# ASSESSING THE BENEFITS OF INTERACTIVITY AND THE INFLUENCE OF LEARNING STYLES ON THE EFFECTIVENESS OF ALGORITHM ANIMATION USING WEB-BASED DATA STRUCTURES COURSEWARE

by

Duane Jeffrey Jarc

M. S. in Computer Engineering, January 1979, Case Western Reserve University
B. S. in Mathematics, September 1970, Case Western Reserve University

A Dissertation submitted to

The Faculty of

The Department of Electrical Engineering and Computer Science
of The George Washington University in partial satisfaction
of the requirements for the degree of Doctor of Science

May 16, 1999

Dissertation directed by
Michael Bliss Feldman
Professor of Engineering and Applied Science

## ABSTRACT

Assessing the Benefits of Interactivity and the Influence of Learning Styles
on the Effectiveness of Algorithm Animation using Web-based Data Structures Courseware

by Duane Jeffrey Jarc

Directed by Professor Michael Bliss Feldman

This study used educational software—courseware—that contained algorithm animations and data structure visualizations that were implemented with the programming language Java, and were embedded in a collection of World Wide Web pages. This courseware differs from most previous algorithm animation systems because it is hosted on the Web and it incorporates a greater degree of interactivity than these earlier systems.

The objective of this study was to determine whether the interactivity provided by this courseware produced a significant learning advantage and whether a student's learning style influenced its effectiveness.

The courseware was used for two years at The George Washington University in the course, *CSci 131 Data Structures*, that was taught using the programming language Ada. Two controlled experiments were conducted during the second year to determine the effect of interactivity and the influence of students' learning style. In addition, students evaluated the courseware each semester to uncover problems in the design and method of its use. Log file data was collected to analyze patterns of use, and selected students were recorded thinking-aloud while using the courseware to determine the understandability of the animations.

The results of this study demonstrated that the students who used the interactive version of the courseware spent significantly more time using it than those who used the noninteractive version. Students who used the interactive version scored better on several of the questions that tested the more difficult lessons, but performed more poorly overall. None of the differences were statistically significant. No significant differences were observed between students of different learning styles. Student assessment of its effectiveness was uniformly high each semester that it was evaluated, which indicates that courseware of this kind is an effective component of an introductory data structures course.

## DEDICATION

To my father, who places great importance in seeing me clothed in a robe with velvet stripes.

> "I have serious reason to believe that the planet that the little prince came from is the asteroid known as B-612.
>
> This asteroid has only once been seen through the telescope. That was by a Turkish astronomer, in 1909.
>
> On making his discovery, the astronomer had presented it to the International Astronomical Congress, in a great demonstration. But he was in Turkish costume, and so nobody would believe what he said.
>
> Grown-ups are like that. . . .
>
> Fortunately, however, for the reputation of Asteroid B-612, a Turkish dictator made a law that his subjects, under pain of death, should change to European costume. So in 1920 the astronomer gave his demonstration all over again, dressed with impressive style and elegance. And this time everybody accepted his report."
>
> Antoine De Saint-Exupéry

## ACKNOWLEDGMENTS

I would like to express my gratitude to the following people, whose help has made the successful completion of this dissertation possible:

To Dr. Michael Feldman, my advisor, whose encouragement and support was critical to my beginning this undertaking and to its successful completion.

To Dr. Rachelle Heller and Dr. Dianne Martin for their multimedia courses that provided an important foundation for this work.

To Dr. John Stasko, a noted expert in the field of algorithm animation, for providing an authoritative review of this work.

To Dr. Melinda Moran for providing a large number of useful references on educational software and learning styles.

To Dr. Richard Felder for permission to use his learning styles instrument.

To the students in CSci 131 during 1997 and 1998, who graciously agreed to provide the comments that were so crucial to this research and to those students who participated in both controlled studies.

To Nayan Patel and Vijay Pandurangan, students in CSci 131 in the spring 1998, who agreed to be tape recorded thinking-aloud while using the data structures courseware.

To Ajay Jayaraj and Iddo Porat, teaching assistants during the fall 1998 semester, for their assistance in conducting the second experiment.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

## GLOSSARY

| | |
|---|---|
| Ada | A general purpose programming language developed by the U. S. Department of Defense, designed primarily for embedded systems |
| almost complete | A binary tree of height n that contains all possible nodes on levels 1 through *n*-1 with the following exception, if a node on level *n* if a node is missing, all its right siblings must be missing also |
| almost heap | An almost complete binary tree with the property that the value in each node, with the exception of the root node, is greater than the value in both children |
| applet | A program, most often written in the Java programming language, that is designed to be embedded in a Web page and executed by a Web browser |
| bandwidth | The amount of information that can be transmitted per unit of time |
| binary search tree | A binary tree, each of whose nodes contains a value that is greater than all the values in the nodes in its left subtree and less than all the values in the nodes in its right subtree |
| binary tree | A tree, each of whose nodes has zero, one, or two children |
| browser | Software used to interpret Web pages, Netscape Navigator and Microsoft Internet Explorer are the most popular ones |
| complete binary tree | A tree of height *n* that contains all possible nodes on levels 1 through *n* |
| courseware | educational software |
| dequeue | An operation that removes an element from either a first-in first-out or a priority queue |
| enqueue | An operation that adds an element to either a first-in first-out or a priority queue |
| heap | An almost complete binary tree with the property that the value in each node is greater than the value in both children |
| height-balanced tree | A binary tree in which the balance factor—the difference between the height of the left and right subtrees—of every node is either -1, 0, or 1 |
| hypermedia | Any of a variety of media, when clicked, that transfers the user to another Web page or to another part of a multimedia program |
| hypertext | Text, when clicked, that transfers the user to another Web page or to another part of a multimedia program |
| interactive video | An interactive multimedia system that uses an analog videodisc player |
| Java | An object-oriented programming language that contains a windows toolkit that was designed to make its applications platform independent |

| | |
|---|---|
| JavaScript | An interpreted language that is embedded in Web pages to add interactivity to those pages |
| meta-analytical study | A study that combines the results of many individual studies to determine an overall effect |
| multimedia | A combination of text, graphics, audio, and video controlled by a computer program |
| Navigator | Netscape's Web browser |
| plug-in | A dynamic code module tailored to a specific platform that enhances the functionality of a Web browser |
| priority queue | A data structure with two fundamental operations, entering the queue called enqueue, and leaving the queue, called dequeue—the criteria for deciding which element to dequeue is based on the priority of the element |
| Prolog | A logic programming language based on the predicate calculus |
| radio button | A button used in groups in user-interface design—only one button in a group can ever be selected |
| server | A computer connected to the Internet that hosts a collection of Web pages |
| toolkit | A collection of classes or subprograms in a programming language that are intended to facilitate easy development of graphical user interfaces |
| traversal | To visit every node of a data structure, such as a linked list or binary tree |
| undirected tree | An undirected graph that is connected and contains no cycles |
| VB Script | A scripting language based on Visual Basic whose code can be embedded in Web pages to add interactivity |
| widget | Any of a variety of different components of a user interface, such as buttons, text boxes, and so on |

## LIST OF SYMBOLS

| | |
|---|---|
| AACE | Algorithm animation for computer science education, multimedia system developed by Gloor |
| ACM | Association for Computing Machinery |
| ANOVA | Analysis of variation |
| ASEE | American Society for Engineering Education |
| AVL | Adelson-Velskii and Landis, a binary search tree that is height balanced |
| CAI | Computer-assisted instruction or computer-aided instruction, instruction hosted on a computer where the presentation and order are determined by a computer program |
| CASE | Computer-aided software engineering |
| CAT | Collaborative active textbooks, Web-based learning environment developed by Brown |
| CBT | Computer-based training, synonym for CAI |
| CD-ROM | Compact disk, read-only memory, an optical storage medium available on multimedia computers |
| CGI | Common gateway interface, an interface used by a Web server to process client requests |
| CHI | Computer human interaction, the field of study concerning interactions between computers and human beings |
| GUI | Graphical user interface, an interface that supports both text and graphics |
| HTML | Hypertext mark-up language, a language used to define the formatting and hyperlinks of Web pages |
| IEEE | The Institute of Electrical and Electronics, Engineers, Inc. |
| JAR | Java archive file, a compressed file that combines many individual Java class files |
| JAWAA | Java and Web-based algorithm animation, system by Pierson and Rodger |
| JCAT | Java implementation of CAT |
| MBTI | Myers-Briggs type inventory, a popular inventory used to determine personality or learning styles |
| MIDI | Musical instrument digital interface, a very compact format for digitally encoding music |
| SIGCHI | ACM special interest group on computer human interaction |
| SIGCSE | ACM special interest group on computer science education |
| SIGGRAPH | ACM special interest group on graphics |

| | |
|---|---|
| TANGO | Transition-based animation generation, Stasko's algorithm animation system |
| UNIX | A computer operating system developed by Bell Laboratories |
| URL | Uniform resource locator, a Web address |
| WWW | World Wide Web, the graphical component of the Internet that allows information to be transmitted in hypermedia formats |
| XTANGO | A version of the TANGO algorithm animation system designed to run under X-Windows |

# CHAPTER 1

# INTRODUCTION

Computer-assisted instruction (CAI) has been used for several decades, but the development of the new multimedia technology and the possibility of creating courseware that can be accessed by students worldwide by hosting it on the World Wide Web presents exciting new opportunities for educators. Before investing considerable time and effort building such multimedia courseware, it is necessary to determine the effectiveness of such materials and how they can be designed and used best. Computer science educators, having the necessary technical skills in the computing field, have a unique responsibility in this effort.

In our research, we consider how educational software that uses some of the new multimedia technology and uses the World Wide Web as a platform should be designed. Specifically, we explore whether adding interactivity enhances learning and whether such software benefits certain types of students more than others. Finally, we consider how such courseware can be incorporated as an effective supplement to traditional classroom education in the lower level computer science curriculum.

## 1.1 Technology and its influence on education

Let us begin with a brief discussion of how technology has influenced higher education in the recent past and specifically what role educational software has played and what role we expect it might play in the future. We also consider the new opportunities and challenges the World Wide Web is presenting for education in the future.

In spite of the dramatic increase in the use of technology in society during the past several decades, the use of technology in the typical classroom has been much less dramatic. Although using computers has certainly been integrated into the curriculum, classroom lecture is still most often conducted using chalk and blackboard. The overhead projector is perhaps the most frequently used technological feature in today's college classroom. The absence of technology is due, in part, to efforts that have not succeeded. In the 1960s and 70s, considerable effort was made to introduce instructional television into classrooms. With the exception of distance education, this effort has not had significant impact. Although it is difficult to predict the future of education, it may be that the classroom will never be the arena where technology has the greatest impact, in fact, the classroom itself may become less important in education in the future. In his vision of the future of electronic education, Dierker [DIE95] predicts that the distinction between formal and informal education will become less clear, that the duration of learning will increase, and that the location of learning may shift, in part, from the classroom to the home.

### 1.1.1 Educational software

Aside from changing the character of the traditional classroom, technology has had the potential to change another facet of education—textbooks. The development of computers made it possible to automate programmed instruction, which marked the beginning of educational software. The development of inexpensive graphic video terminals made it feasible for educational software to include graphics with text. More recently multimedia computers have enabled instructional designers to include audio, video and animation in their products.

Although educational software has been developed for several decades now, it has not eliminated the need for traditional printed materials. Again, it may be that printed materials will remain a key element of education in the future. Nonetheless, the importance of educational software, if only to supplement the classroom and textbooks, is well accepted. Among the challenges identified by Glennan and Melmed [GLE96] in their characterization of a national strategy for educational technology, is the need to train teachers to effectively use technology and to assure a plentiful supply of high-quality educational software. They note, however, that there is currently too little high-quality educational software available—particularly at the more advanced educational levels. Although new technology, such as inexpensive multimedia computers that can reproduce digitized audio and video, has increased interest in developing educational software, one of the key obstacles is the high cost of production. Soloway [SOL98] notes that it is difficult to make a profit developing educational software. The problem is a combination of high cost and low demand. Improved tools for developing educational software should make it possible to reduce production costs. Increasing the demand is a more complex problem, however. The low demand is due, in part, to limited funds available to school systems. It may also be due, in part, to a perceived low benefit compared to the cost. Clearly, finding ways to produce such software less expensively and demonstrating its benefit would both contribute to the greater production of such software in the future.

### 1.1.2 The impact of the World Wide Web

In only a few years the World Wide Web has evolved dramatically. It began as a platform that could be used to host static text and graphics—apart from hypertext, little more than what can appear in print media. Although bandwidth issues still are a limiting factor for full motion video, the Web can now be used to host multimedia applications. Equally dramatic has been the growth of instructional Web sites—courses on the Web. A course on the Web may consist of as little as the syllabus and assignments for otherwise traditional classes to courses that are designed to be distance education courses for which the Web becomes a primary delivery medium. According to Reeves and Reeves [REE97], because Web-based instruction is so new, there is little research that demonstrates its effectiveness. Furthermore, given only several years of experience with such courses, how the Web can be used most effectively for courses along this spectrum has yet to be

determined. Bostock [BOS97] notes that the worst use of this technology, is to replace classroom lecture with printed lecture notes, however, such an approach is undoubtedly the least costly. Yet, there is little evidence that most courses on the Web use its full capabilities. In their assessment of the current state of Web-based instruction, Gillani and Relan [GIL97] believe that the majority of instructional Web sites lack interactive multimedia and fail to adhere to good instructional design principles.

Creating a course on the Web that consists of the syllabus, assignments and lecture notes, requires little effort—if these materials already exist. Most word processors today allow the document to saved in HTML (Hypertext mark-up language) format. Although, such documents sometimes still require minor reformatting and addition of hypertext, the overall process remains relatively simple. Since the inception of the Web, a number of new capabilities have made it possible to create Web-based courses that include something more than linked printed material.

In his recent guide to designing instructional Web sites, Brooks [BRO97] emphasizes the importance of active learning in Web-based instruction, but notes that incorporating the principles of active learning remains an important challenge in their design. A degree of interactivity can be added to Web pages by embedding JavaScript or VB Script code into the HTML. JavaScript and VB Script are scripting languages that are interpreted by the Web browser. They allow the form elements of HTML to act as vehicles for input to and output from this code. A simple example of the kind of interactivity that can be added with these scripting languages is an on-line multiple choice quiz that would provide immediate scoring and feedback. A limited interactive graphics capability can also be achieved. For example, a map could be placed in a Web page that allows students to test their knowledge of states and their capitals. Clicking on a state, they could be prompted for the state name and its capital and be given an immediate response regarding the correctness of their answer. These scripting languages are limited, however, in their graphics ability. What they can not implement is dynamically generated graphics. For example, it would not be possible to prompt the user for a set of numbers, and then generate a bar graph of those values.

Dynamically generated graphics is possible, however, when we use applets written in a programming language, such as Java. Unlike JavaScript, the Java code is not embedded in the HTML. It is instead, partially compiled and the generated byte code is attached to the Web page. Rather than using the form elements of HTML, Java applets create their own widgets that are used to create the user interface. Java currently supports the ability to generate audio, but no support has yet been provided for video. Full multimedia capabilities can be created in Web sites, however. There are several multimedia authoring tools that have this capability. Multimedia applications created by Macromedia's Director can be ported to the Web using Shockwave, which compresses the otherwise very large multimedia files. Asymetrix Toolbook, another multimedia

authoring tool, allows its applications to be run on the Web. Unlike Java applets, however, the user must have the necessary browser plug-ins to run these multimedia applications.

Developing interactive multimedia sites is significantly more labor intensive than developing Web sites with simple text and graphics. Adding interactivity to Web pages with either a scripting language or through applets with a programming language requires some amount of programming, which necessarily adds to the effort, and therefore to the cost. Adding multimedia may also require some amount of programming. It certainly requires added effort, regardless. We are certainly able to produce such interactive multimedia applications for the Web, but it may not be the most effective use of resources. Clearly, given that Web-based instruction is still in its infancy, much research is needed to determine the benefits compared to the cost. Furthermore, investigation is needed to determine what kind of Web-based instruction is appropriate, depending upon whether the Web complements traditional instruction or attempts to completely replace it.

## 1.2 The importance of tailoring teaching to learning styles

Another factor that is often discussed regarding educational software is that it has the potential to benefit students with specific styles of learning more than the student population as a whole. The reasons for this belief are that learning with educational software differs from traditional classroom education is a number of ways. Unlike the classroom environment, learning with educational software is a self-paced type of learning, which enables the student to spend as much time as is needed to master the material. Educational software has the potential to provide an exploratory environment, in which students can learn through experimentation. Although scientific laboratory settings provide a similar environment, because of the lower cost of creating virtual environments, the range of possibilities can be much greater. In addition, if properly designed, educational software can be highly interactive, compared to the usually passive environment that students experience while attending a traditional classroom lecture. Finally, with the development of the new multimedia technologies, educational software can become a highly visual learning environment—one that might include animation.

There are a wide variety of different learning style models that have been developed. They measure a number of different characteristics of how students learn. Among the differences that they measure is the way in which students approach learning and how their understanding develops. The typical classroom lecture presents material in a sequential, orderly fashion. Some students prefer an exploratory kind of learning or to skim the surface of a subject before investigating details. Their understanding may occur suddenly only after the whole topic has been studied. Educational software, containing hypermedia, offers the possibly for such nonlinear explorations, which is not possible in the classroom environment. Another dimension that many of the learning style models assess is the degree to which students need to be active participants in the learning process. For those students whose style of learning lies closer to the active pole of

this dimension, educational software that is designed with a high degree of interactivity might be better suited to their style of learning. Finally, many of the models differentiate between students who are visual learners and those who learn best through verbal or textual communication. New educational software that contains a variety of media, which may include audio, video, text, and graphics, has the potential to satisfy students on any point of this spectrum, provided that it uses the range of media that are available.

The need for studies that explore differing intellectual strengths of students has been recommended by Heller [HEL90a]. Yildez and Atkins [YIL93] have suggested that there is a need for more studies of multimedia that consider the effect of issues such as students' preferred learning styles. West [WES92] has theorized that students who have weak verbal skills, the skills that our educational system has traditionally valued most, may have strong visual capabilities. It may be that including educational materials that are more suited to such students will increase their chances for success. Demonstrating the benefit of educational software, particularly to students who might have difficulty learning in a more traditional environment, should provide an additional incentive for devoting available resources to the development of educational software. Myers and Jones [MEY93] have observed that cultural background influences a student's learning style. If their contention is true and educational software can help those of a particular learning style, it may also be able to assist cultural groups whose educational performance tends to be below average.

The view that learning styles should be considered an important component in the design of educational programs is not shared, however, with equal enthusiasm by all educational theorists. Ellis and Fouts [ELL93] contend that existing research offers little to support this viewpoint.

## 1.3 Algorithm animation in computer science education

The animation of algorithms is one of the most obvious applications of this technology in the area of computer science education. Computer programs and the algorithms that they implement are naturally dynamic. When a program runs or executes, changes occur in the internal state of the program. This state consists of some collection of data structures. Often, these data structures have a very natural visual depiction. Software visualization has been used to assist in the debugging of programs, as in the study by Feldman and Moran [FEL89], and to assist students who are attempting to understand how particular algorithms work. Anyone who has taught an introductory course in data structures has likely felt the desire to be able to show a sorting algorithm in action. With the usual classroom tools of chalk, blackboard or even overhead projector, it is difficult to show any motion, other than with consecutive still frames.

This perceived need led to the development of algorithm animation systems. With these systems, it is possible to create visualizations of the data structures that are manipulated by the algorithm and to watch the changes that occur in these data structures as the algorithm executes. A variety of techniques have been developed to describe how a particular algorithm should be

animated. A given algorithm can be animated in a variety of different ways; some animations clearly might better convey some of the more subtle aspects of a complex algorithm than others.

Although these systems have been used in computer science education for the past decade, their use has been confined to a small percentage of computer science students. The majority of computer science graduates during the past decade have probably never used such a system. We believe that there are several reasons that animation systems have not yet gained more widespread use. First, some these systems were designed to run on high-end platforms, which were unavailable to many students. Another reason may be that some of these systems have not been easy enough to use—at least for inexperienced students. What is more important, however, is that most instructors have not found a compelling reason to include their use in the curriculum. One reason, perhaps, is that, although most computer science educators who have considered this issue, intuitively believe that algorithm animation should be helpful in teaching computer science principles, the research that has attempted to determine whether algorithm animation improves student learning has produced rather mixed results. This fact is particularly surprising given the years of research that have confirmed the favorable effect of computer-assisted instruction, generally.

There have been important developments in technology during the past decade that have influenced both the character and the availability of such systems. The availability of inexpensive computers that provide high-resolution graphics has made the hardware necessary for such systems accessible to more people. The addition of multimedia features to most personal computers has made it possible to incorporate new features, such as audio, in some of these systems. Also, new techniques have been devised to make creating animations simpler. Finally, and perhaps most importantly, the World Wide Web has provided a distribution platform that makes such animation systems available to computer science students worldwide.

Given the current state of educational software and algorithm animation, we felt that it was important to design a study that would add to the important field of investigation that helps us understand how to effectively incorporate this rapidly changing technology into the field of computer science education. Also, because Web-based instruction is so new, and research on how it is best used is so limited, we felt that it was important that our study use educational software that was hosted on the Web. Consequently, we developed Web-based educational software—courseware—that contains algorithm animation and data structure visualizations. We have used this courseware for two years in an introductory data structures course at The George Washington University. This dissertation discusses the results of two experiments that we conducted and the how this courseware was able to be used most effectively in this course. These experiments measured the added benefit of a specific kind of interactivity that was included in the courseware, and the extent to which the learning styles of the students who used it influenced this benefit.

In Chapter 2, we review the research that is related to our area of investigation. We present a description of the multimedia courseware that we used in our study in Chapter 3. Chapter 4 is devoted to a discussion of our overall research plan, a discussion of the hypotheses of our study and the details of the experiments that we conducted. In Chapter 5, we analyze the results of our study, including an assessment of how we have concluded that our Web-based courseware can best be integrated into an elementary data structures course together with an analysis of the results of our two experiments that measured the benefit of interactivity and the relationship of learning styles. Finally, in Chapter 6, we present our conclusions and discuss suggestions of related future work that we believe would be beneficial.

# CHAPTER 2

# REVIEW OF THE LITERATURE

Before outlining the study we have conducted, let us specify the domain of our research. Our research space, illustrated in Figure 2.1, lies in the intersection of three major areas as they apply to computer science education: (1) CAI, (2) learning styles, and (3) algorithm animation.

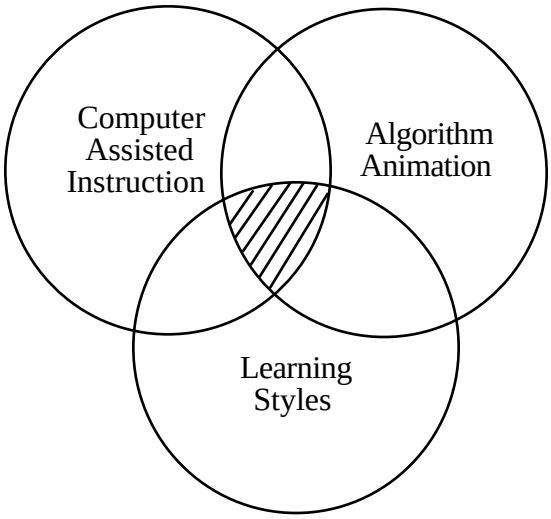



Figure 2.1 Dissertation research space

The courseware that we have used in our study is an example of CAI, consequently it is imperative that we consider the research to date regarding its effectiveness. Second, our study was designed to measure whether there is any relationship between learning styles and the effectiveness of algorithm animation, so it is important to investigate the various learning style models that have been defined and studied in the past to understand why the model chose is most appropriate to our study. Finally, because our courseware contains extensive use of algorithm animation, we need to review the algorithm animation systems that have been implemented and examine the studies of their effectiveness.

## 2.1 Computer-assisted instruction

We begin our review of the literature with CAI. This field is not new, but it is a constantly evolving one. Its roots are generally traced to the work of Skinner in the 1950's. Kulik and Kulik [KUL86] characterize Skinner's work as the first landmark in this development, the individualized instruction systems of the 1960's as the second landmark, and computer-based education as the third. We believe two subsequent major developments have occurred since then. The first is multimedia, which can significantly improve the technical quality and appeal of CAI, and the

second is the World Wide Web (WWW), which provides the ability to distribute CAI without cost. Our literature review includes the final three landmarks, CAI, multimedia, and the WWW. These three are germane because the courseware we have used in our study is CAI that incorporates a variety of media and uses the WWW as its execution platform.

### 2.1.1 Meta-analytical studies

CAI has been used for several decades, and there have been hundreds of studies of its effectiveness. What is more important is that there have been a dozen or more major meta-analytical studies—studies that distill the results of many individual studies. It is these meta-analytical studies that will be the focus of our review to summarize what general conclusions have been reached regarding the effectiveness of CAI.

Because CAI has its roots in programmed instruction, it is interesting to begin this review with a meta-analysis, by Kulik *et al.*[KUL82], performed on the effectiveness of programmed instruction in secondary education. This early study differs from the later ones because they found that programmed instruction neither raised achievement levels nor did it improve students' attitudes about the subject nor the quality of the teaching.

In a later meta-analytical study, by Kulik and Kulik [KUL86], of computer-based education at the college level that included 101 individual studies from 1967 through 1982, the results were more favorable. The measured outcomes of these studies included achievement, attitudes, and required time. Their findings indicate that achievement was raised by 0.26 standard deviations on average. Furthermore, time needed for instruction with computer-based education was one-third less than that required with conventional instruction. Finally, they found a small, but positive effect of computer-based instruction on student attitudes.

Most of the subsequent meta-analytical studies demonstrate similar results. An updated study by the same individuals [KUL91] found slightly better results—examination scores raised by 0.30 standard deviations. Other later studies [LIA92, KHA94] had similar findings. In the most current meta-analytical study, by Fletcher and Gravatt [FLE95], the possibility of improved performance as a result of multimedia technology is raised. They note, however, that their study does not reflect the new multimedia technologies because they found so few studies that incorporated multimedia courseware.

Although multimedia is relatively new, interactive video has been available since the mid-70's. Interactive video requires a special videodisc player that can store about 30 minutes of analog video. Several studies, one by Bosco [BOS86] and another by Pollard [POL92], have reviewed a collection of individual reports and found generally favorable results regarding achievement, average training time and attitude. Two meta-analytical studies of interactive video, one by McNeil [MCN89] and another by Fletcher [FLE89], found overall mean effect sizes of 0.50—a larger effect than the average of previous meta-analytical studies of CAI.

Summarizing the previous studies, we can conclude that CAI significantly reduces learning time and has a moderately positive effect on achievement and student attitudes. These conclusions are vital because they provide a baseline for all further studies, and for our purposes, a baseline for comparing the results of the effectiveness of algorithm animation. It would be unfair to conclude, without one note. In spite of the apparently strong evidence of the benefits of CAI, there are those who dispute the validity of these studies. Clark [CLA85] attributes the achievement gains to better instructional methods embedded in the CAI—not to the CAI itself. In a later assessment of interactive video [CLA91], the favorable results are attributed to interactivity, not the video medium.

This debate has been rekindled recently. Kozma [KOZ94a] challenged the position taken by Clark [CLA83b] a decade earlier. His paper was the catalyst for a special issue of *Educational Technology Research and Development*, which was devoted to the debate about whether it is the media or the instructional method that contributes to the measured learning benefits. Kozma argues that the question should be reframed to ask not whether media *does* influence learning, but whether it *will*. He attributes the failure to establish a relationship between media and learning, in part, to the experimental methods that have been used. He argues that the purely quantitative studies that have traditionally been used are unable to uncover causal mechanisms. He suggests that observational methods, such as think-aloud protocols, eye fixations and log files would enhance the analysis of any such relationship. Another critic of Clark's position, Morrison [MOR94] argues that it is inappropriate to try to make such a distinction because the media and instructional method are so interdependent that it is nearly impossible to separate their individual effects. Reiser [REI94], an instructional designer, makes a similar criticism indicating that certain media attributes are necessary to implement some instructional methods. Clark [CLA94], in response to his critics, holds to his claim that any study that attributes learning benefits to the use of media has failed to control for the instructional method. In a subsequent rebuttal, Kozma [KOZ94b] concurs with Morrison that the fundamental issue is that method and media cannot be separated. This debate will no doubt continue. We believe that the debate underscores the importance of using both quantitative and qualitative methods to gain a better understanding of not only the effect, but how the learning process is affected by the media.

### 2.1.2 Multimedia

Multimedia has been a gradually evolving field whose development has been largely driven by improvements in computer hardware. Text-only systems predate systems containing multimedia because they contain only the single medium. The gradual development of higher and higher resolution monitors made the inclusion of graphics possible. The addition of sound cards and speakers, to even the most inexpensive personal computers, has led to the ability to add audio. Finally, new higher speed processors make digital video a possibility. Multimedia incorporates

some combination of these four media: text, graphics, audio, and video. Heller and Martin [HEL95] have defined a two-dimensional taxonomy that characterizes the full range of possibilities in multimedia applications.

As we have noted in the summary of the most recent meta-analysis of CAI, multimedia is too new to have been included in such a study. Although we found no formal meta-analysis that attempts to quantify the mean effect size produced by multimedia, a recent study, by Najjar [NAJ96], attempts to summarize the effects of using various media—alone and in combination. He concludes that the results are inconsistent regarding whether redundant multimedia produces any learning advantage compared to using a single medium. He qualifies the conclusion by noting that multimedia produce the greatest effect when media are mutually supportive and promote dual-coding of information.

The overall inconsistency suggests that it is worthwhile to examine studies that have focused on individual media and various combinations. We begin with the video medium. Cronin and Myers [CRO97] investigated how important the inclusion of video was to interactive multimedia instruction for learning listening skills. They found no significant difference between the groups that used the version that contained video and the group that did not.

In a study by Barron and Kysilka [BAR93] that measured the effect of adding audio that narrated text already appearing on the screen, the results indicated that the addition did not favorably affect achievement, but did unfavorably affect speed. By contrast several studies by Mayer and Anderson [MAY91, MAY92] that evaluated the combined effects of narration and animation found that their combination did produce positive effects, provided that the narration was concurrent with the animation. Another more recent study by Farday and Sutcliffe [FAR97] also confirms the positive effects of speech to animation as well as the benefit of adding simple captions.

Because animation is of particular interest to our investigation, let us consider studies that have focused on its effects. Reviewing a collection of studies, Rieber [RIE90a] concludes that the results are inconsistent. Next, we examine several individual studies that have attempted tightly controlled experiments. In a study by Rieber, Boyce and Assad [RIE90b] that compared the addition of static and animated visuals to text, the addition produced no favorable effects. In a study by Pane, Corbett, and John [PAN96] that compared the use of animation to still graphics found no added benefit to the animation.

Because multimedia hardware has only recently become a standard feature of all personal computers and because the hardware for digital video is still evolving, it is not surprising that it is not yet possible to form a clear picture of its impact. The lack of sufficient studies of its effects supports the need for more studies of the kind we have undertaken.

### 2.1.3 The World Wide Web

The World Wide Web is a newer, yet equally important development compared to multimedia software. The incorporation of multimedia into CAI has the potential to improve the quality of the presentation. Staging CAI on the WWW greatly facilitates its distribution. We found no empirical studies that have evaluated the effectiveness of Web-based CAI, although a considerable amount of courseware is currently being developed for the Web.

Like multimedia, the Web has evolved in incremental steps. In its initial stages, only documents containing text and graphics could be hosted on Web sites, using hypertext mark-up language (HTML). The interactivity of such documents was limited to the hypertext links. The creation of dynamic documents required common gateway interface (CGI) programs that reside on the server side. In such a configuration, each interaction requires that a message be sent from the client to the server, the server must then dynamically generate a page that is returned to the client. Clearly, such communication adversely affects the response time. Many computer science educators, such as Marshall and Hurley [MAR96] and Carlson *et al.* [CAR96], have noted that using Java can greatly improve the response time of such systems, because the need for client-server communication is eliminated. In their discussion of the use and design of visualization, the working group on visualization [NAP96a] at the recent conference on integrating technology into education notes that Java's abstract windowing toolkit is one of its important advantages. Java is not without its detractors, such as Tyma [TYM98], who cite its failure to achieve its claim of platform independence and its slow speed among its shortcomings.

Clearly, the ability to develop CAI that can be hosted on the World Wide Web offers considerable opportunities for educators in general, and computer science educators in particular. The potential of such CAI provides clear reason for its development and evaluation, which is one central aspect of our study.

## 2.2 Learning styles

Next, we consider learning styles and their relationship to multimedia courseware. We begin with a discussion of the range of learning style models that have been developed and then review the empirical studies that have been conducted exploring the connection between learning styles and CAI and those that examine learning styles and learning computing skills—programming, in particular.

### 2.2.1 Categorizing learning style models

There have been over a hundred different learning style models developed during the last several decades; unfortunately there is no universal agreement upon which of these models is best. Bonham [BON88] notes that this lack of agreement is a key problem for anyone interested in using

learning styles. This problem is one we faced, given that we wished to explore the relationship between multimedia and learning styles.

Before attempting to evaluate these models for selection, let us consider the efforts that have been made to define and order this broad array of different models. Keefe [KEE88], in his definition of learning styles, identifies cognitive, affective and physiological factors as those indicators that learning styles attempt to identify. Curry [CUR83] proposed an onion metaphor consisting of three basic layers, suggesting that every learning style model belongs to one of these three layers. Her innermost layer contains the models of cognitive personality, her middle layer contains models of information processing style, and her outermost layer contains models of instructional preference. We illustrate this categorization in Figure 2.2. Claxton and Murrell [CLA87] have adopted Curry's metaphor, but added a fourth layer, social interaction, which they place between Curry's outer two layers. The construct validity of Curry's system has been examined by Marshall [MAR87] and it has been determined that the outer two layers are independent. Given that Curry's system is one of the few attempts to order this diverse field and that several others have adopted it, we, too, adopt it as our framework for discussing various learning style models.

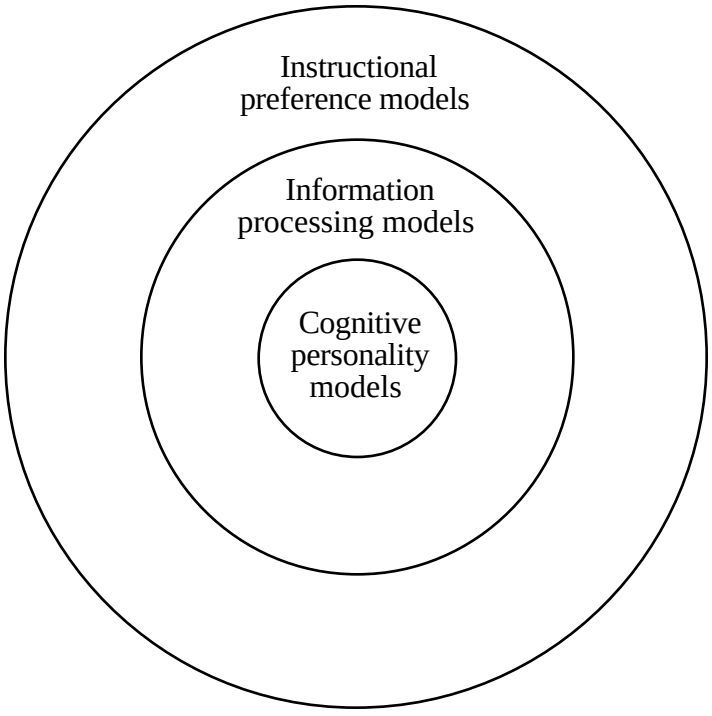



Figure 2.2 Curry's learning style model layers

Because of the vast number of different models that have been developed, we can only consider a small number. Among the criteria we use to select those models for our consideration, is whether a particular model has achieved a high degree of usage by others, whether it has been

used in any empirical study of CAI, and whether it has been tailored to the kind of students that we planned to study.

We begin with Curry's outermost layer, instructional preference. Learning style models in this layer, measure interactions with the external environment. According to Hickcox [HIC95], the learning style model developed by Dunn, Dunn and Price, belongs to this outermost layer. It is the only model in the outermost layer that we consider. It uses a self-reporting questionnaire of true or false questions that measure a total of 24 different parameters. These parameters are grouped into five categories that include emotional, psychological, sociological, environmental, and physical characteristics of student learning.

Next we consider learning style models in the information processing layer. There are three models within this layer that we consider. The first, and perhaps one of the most popular, is the Kolb learning style inventory. Kolb categorizes learners into four learning styles using two bipolar dimensions. The first dimension ranges from concrete experience on one pole to abstract conceptualization on the other. The second dimension ranges from active experimentation to reflective observation. Individuals belonging to the four learning styles are called divergers, assimilators, convergers and accomodaters. The strength of divergers is generating ideas and brainstorming. Assimilators are best at generating theoretical models. Convergers are the opposite of divergers. Convegers prefer to choose a single correct answer, rather than explore the field of possibilities. Finally, accomodaters are the opposite of assimilators. Accomodaters prefer direct experience to theories. We illustrate the Kolb learning styles in Figure 2.3. Howard *et al.* [HOW96] have had success with factoring these different styles into teaching a CS 2 course.

The next learning style model we consider is the one defined by Gregorc. It also uses two bipolar dimensions to define four learning styles. The first dimension is varying degrees from abstract to concrete. The second dimension has *sequential* at one pole and *random* at the other. The four learning styles are the four combinations of the two dimensions.

The final learning style model in the information processing layer that we discuss is the model developed by Entwistle [ENT81]. He categorizes students according to how they approach studying. One style is by first reviewing the overall topic—surface processing. The second style is by immediately investigating details—depth processing.

Finally, we move to Curry's innermost layer—the cognitive personality layer. The most popular learning style model belonging to this layer is the Myers-Briggs Type Indicator (MBTI). The MBTI defines four bipolar dimensions resulting in sixteen unique personality types. The first dimension is extrovert (E), those who are energized by the company of other people, or introvert (I), those who are energized by solitude. The second is sensors (S), who rely primarily on information gathered from their senses, or intuitors (N), who rely more on their intuitive sense. The third is thinkers (T), who make decisions based on rules, or feelers (F) who decide based on

their feelings. The fourth dimension judgers (J), who desire closure, or perceivers (P) who prefer exploration.

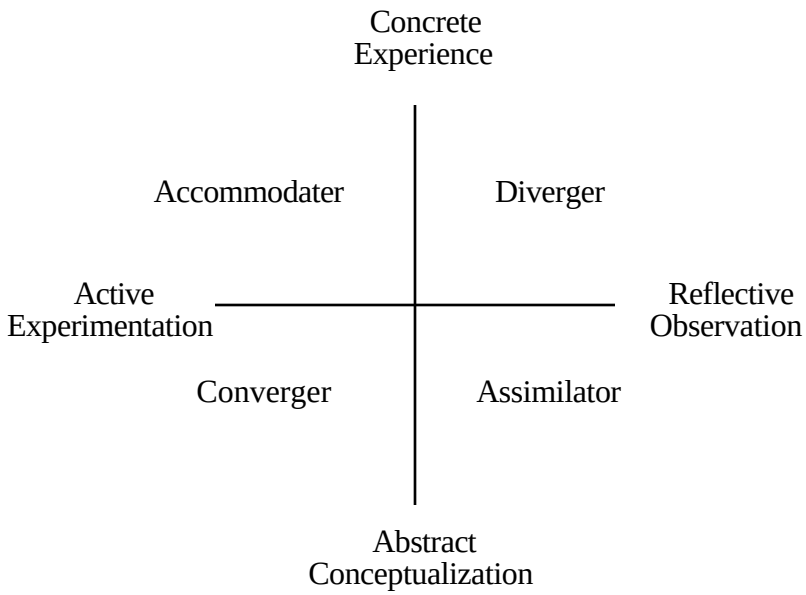



Figure 2.3 Kolb's learning style model

The final learning style model we consider is the one designed by Felder and Silverman [FEL88]. Although this model has not been as widely used as some of the previously mentioned models, it was designed and has been studied primarily with engineering students—a similar group to the kind of students that we have studied. This model does not fit into a single layer of Curry's metaphor, but instead spans both the information processing and cognitive personality layer. It consists of five dimensions, although the forty-four question inventory developed by Felder and Solomon only evaluates four of the five dimensions. It excludes the organization dimension. Figure 2.4 illustrates the five dimensions of this model. It combines aspects of several of the models that we have already discussed. The perception dimension that has sensory and intuitive as its two poles, is one of the dimensions of the Myers-Briggs inventory. The fourth dimension—processing, which has active and reflective as its two poles—is a component of Kolb's learning style model. Two of these dimensions are particularly germane to our investigation. The input dimension that differentiates between visual and auditory is pertinent because visualizations are a key component of our courseware. Felder observes that one of the flaws of most engineering education is that although most students are visual learners, most of the teaching is done in a verbal manner. The processing dimension is also important to our study, in which we have assessed the impact of interactivity. Felder describes active learners as ones who prefer to do something, other than listen and watch.

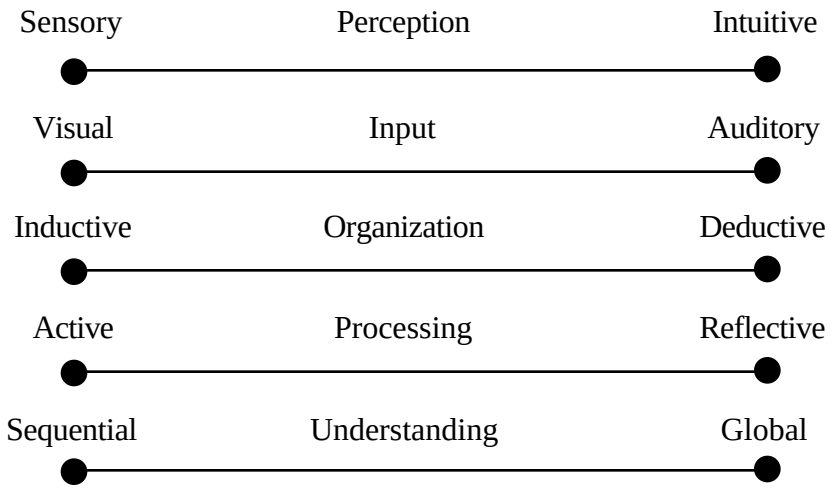



Figure 2.4 Felder-Silverman learning style model

### 2.2.2 Empirical studies of learning styles and CAI

There have been some empirical studies that have attempted to determine whether there is any relationship between learning styles and performance using CAI. The number of such studies is far fewer than the studies that have assessed the benefit of CAI overall. Consequently, we do not have the benefit of meta-analyses to help answer this question, but instead we must consider the individual studies.

The first study that we review is a study by Billings [BIL92] that used the learning style model of Dunn, Dunn and Price. This study involved a group of nursing students who used CAI that used an interactive videodisc. No significant effect among different learning styles was found when measuring achievement, but student attitude was found to be a strong predictor of performance.

We have located two studies that seek to determine whether there is any difference in the performance of students of the four different Kolb learning styles.

In the first study, conducted by Cordell, [COR91] two different implementations of CAI were used—one was linear, the other used hypermedia. Significant differences were found in student performance between the two types of instructional design. No main effects were found for the four Kolb learning styles. None were expected, however, because the goal of the study was to match learning styles with instructional design.

The second study, by Larsen [LAR92], involved a course on data communication concepts using courseware that incorporated interactive video. The effectiveness of the interactive video and its level of acceptance were both measured. No significant differences were found among the four different Kolb learning styles. Larsen speculates that there is no difference because such instruction accommodates the needs of learners of all different styles.

We have identified one study, performed by Ester [EST94], that categorized students according to the four Gregorc learning styles. Ester compared the effectiveness of using CAI with ordinary lecture for teaching vocal anatomy to a group of undergraduate music students. His findings indicated that there was no significant difference in achievement when concrete or abstract learners were given the same instructional technique, but the abstract learners performed significantly higher with the lecture approach than with the CAI.

Another study, by Allinson [ALL92], categorized students into Entwistle's learning styles. Although there were no differences between the two groups regarding learning outcome, there was a significant difference between the two groups regarding navigational style. The high reproducing group preferred a linear approach, studying screens at a slower rate. The high meaning group demonstrated a higher degree of navigational activity and a greater use of hypertext linkages.

Carlson [CAR91] used a self-designed learner survey that divided students into two groups—inductive and deductive learners. This dichotomy does appear in other learning style models. It is one of the five dimensions of the Felder-Silverman model [FEL96], for example. Two instructional designs were used in the study—one designed for each type of learning style. Her results indicated that the students whose learning style was matched to the instructional design had significantly higher achievement.

Given the small number of studies that have been performed and the diversity of learning style models that have been used, it is difficult to form any definitive conclusion.

### 2.2.3 Empirical studies of learning styles and computer skills

The second type of empirical studies that are of interest, are those that assess whether there is any relationship between learning styles and students' ability to learn computer skills—programming skills being our greatest interest.

The relationship between the Kolb learning styles and novice students' ability to learn spreadsheets and electronic mail has been studied by Bostrom [BOS90]. The students with the abstract styles (assimilators and convergers) had significantly higher scores on comprehension tests and significantly higher accuracy than those with the concrete styles (accomodators and divergers).

Davidson *et al.* [DAV92] studied the relationship between learning styles and the performance in a course on computer applications. Students were categorized according to the four Gregorc learning styles. Students with the abstract sequential style scored significantly higher than average, while students with the abstract random style scored significantly lower.

Next, we consider several studies that have investigated the relationship between programming performance and learning styles. In the first study, conducted by Corman [COR86], the predictors for the success of beginning programming students were investigated. Students

were categorized according to the Kolb learning styles and with the Myers-Briggs Type Indicator. Neither categorization was related to their performance in a beginning COBOL class.

Cavaiani [CAV89] investigated whether there is a relationship between the cognitive style of student programmers as measured by the Group Embedded Figures Test and their performance on two different programming tasks. He found that students with an analytical cognitive style performed significantly better than those with a global style on program comprehension and debugging. He also found no significant difference between the two groups on the task of finding and correcting compilation errors.

In another study, by Gordon [GOR90], the relationship between left-brained and right-brained individuals regarding success in computer programming was investigated. Three groups were measured: a computer class, a university computer center staff of computer professionals, and a group of bank employees. The university computer center staff members were found to be significantly right-brained, the bank employees significantly left-brained. Among the students in the computer class, those who were more right-brained performed significantly better on programming projects than the left-brained students, but no difference was found between the two groups on examination scores.

Bishop-Clark and Wheeler [BIS94] investigated whether there is a relationship between the Myers-Briggs personality type and performance in an introductory programming course. Of the four dimensions of this indicator, only the dimension that differentiates between sensing and intuition showed any significant difference. Furthermore, it was only on programming assignments, not examination scores, where sensing students performed significantly better than intuitive students.

In a recent study, by Wu, *et al.* [WU98], that used Kolb's learning style inventory to separate abstract learners from concrete learners, the abstract learners performed significantlybetter on both the posttests and the two retention measures. In addition, concrete learning models, used to teach recursion, were found to be superior to abstract ones. What the study failed to show, however, was that abstract learners performed better when given abstract learning models or that the performance of concrete learners improved when they were taught using concrete models.

Given the small number of studies and the differences in learning style models, it is difficult to make any definitive conclusion.

## 2.3 Algorithm animation

Finally, let us consider the research that has been completed to date, in the area of algorithm animation. First, we consider the problems inherent in algorithm animation, next the various algorithm animation systems that have been built, and finally, we review the various studies that have attempted to determine whether such systems favorably affect the learning process.

### 2.3.1 The problem of algorithm animation

The idea of animation is not new. Cartoon animation has existed for many decades. It is well known that the cost of such animated productions was very high before computers assisted the animation process. Animations had to be created by individual drawings for each frame. Although the use of computers in animation reduces the effort, animation is still an expensive process compared to video production.

The animation of computer algorithms would seem a simpler problem than cartoon animation, because we are already in the computer domain. The ideal situation would be to totally automate the process of algorithm animation—create software whose input is any program and whose output is its animation. There are many obstacles to achieving such a totally automated process. One obstacle is that what is interesting to animate differs from one algorithm to the next. So, the input to any such animation program must include not only the program, but what to animate and how to animate it.

Price *et al.* [PRI94] have developed a taxonomy of software visualization. They define software visualization as "the use of the crafts of typography, graphic design, animation, and cinematography with modern human computer interaction technology to facilitate both the human understanding and effective use of software." Clearly algorithm animation is only one component of this broader field. Nonetheless, certain aspects of their taxonomy clarify the technical challenge of algorithm animation. For example, among the measures they use to categorize elements of this domain are generality and scalability. Both highlight some of the technical challenges of an automated algorithm animation system. To make any such system work on any program, written in any programming language, and run on any system, presents many technical challenges. Large programs and large data sets present other technical challenges. Techniques have been devised to address some of these issues. Stasko and Muthukumarasamy [STA96] have developed a technique they call semantic zooming to address the issue of large data sets. Finally, there is yet another technical challenge that is currently receiving considerable attention—the feasibility of making such systems available on the World Wide Web.

Clearly no algorithm animation system can ever totally automate the process for all programs and without some additional input regarding what to animate, but the more we broaden the scope of any system, simplify its use, and improve the quality of animation, the more progress we have made with the technical challenges.

### 2.3.2 History of algorithm animation

Next, let us consider what progress has been made in solving these technical challenges. The generally accepted beginning of algorithm animation is the video tape entitled *Sorting Out Sorting* presented by Baecker [BAE81] at the ACM SIGGRAPH conference in 1981. This videotape shows the animation of nine different sorting algorithms. Anyone who has taught a data

structures course realizes that certain sorting algorithms, such as the quick sort, are particularly difficult to explain. This videotape allows students to watch the behavior of the algorithm rather than try to imagine its actions from a verbal explanation or from several static images. The concept of algorithm efficiency as measured by the big-O can be a rather abstract idea for many students, yet it is important for students to understand the important difference between the $O(n^2)$ and the $O(n \log n)$ algorithms. The videotape shows several different sorting algorithms running at the same time to enable students to understand the significance of algorithm efficiency.

The first major interactive algorithm animation system, Brown's [BRO88a] Balsa, was developed at Brown University. This algorithm animation system was used in introductory programming courses to allow students to watch already scripted algorithm animations. Students were able to control the animation by starting, stopping and replaying the animations. The system was also used in advanced courses as a high-level algorithm animator. The technique that was used to implement these animations was to annotate the program to be animated using the *Interesting Event* manager of Balsa. A later version Balsa-II [BRO88b] added color and some rudimentary sounds. A more recent animator named Zeus [BRO91] improved upon the sound capabilities by adding a MIDI synthesizer.

Stasko [STA90a], also at Brown University at the time, created another algorithm animation system called TANGO. The technique used by TANGO is called a path-transition paradigm [STA90b]. Because TANGO does not erase and redraw each image as the previous animation systems did, TANGO is able to produce smoother more cartoon-like animations. A later version of this system, XTANGO, was developed to run under the X11 Windows system that runs on most UNIX workstations. More recently Stasko [STA97] has been experimenting with having students build their own animations in an advanced algorithms course.

A variety of techniques have been used to simplify the implementation of algorithm animations. Roman and Cox [ROM93] have defined four different methods for specifying software visualization. They include (1) predefinition, (2) annotation, (3) declaration, and (4) manipulation. Predefinition is used in software visualization systems, such as CASE tools, but is not practical for algorithm animators, because the variety of what might be animated is too great. All the previously mentioned systems specified their animations by annotation—annotating the source code of the algorithms by specifying the interesting events. One extension of the annotation method was an interpreted programming language called GEF developed by Glassman [GLA93] that was designed to work with the algorithm animation system Zeus. The intent of this system was to reduce the time necessary to develop animations.

Next, we consider declarative specifications. The difference between declarative and annotative specifications is similar to the difference between declarative and imperative languages. In declarative languages, the programmer defines what the program does, but not how. Several algorithm animation systems have been developed that use declarative specifications. One is a

system called Pavane, which performs three-dimensional visualizations of concurrent programs, that was developed by Roman *et al.* [ROM92b] at Washington University. Another is a system developed by Takahashi *et al.* [TAK94] that incorporates what they call their bidirectional translation model. Their declarative mappings are defined by Prolog statements. They claim their approach is more abstract than the technique used in Pavane.

The final method for specifying animations is by manipulation. Helttula *et al.* [HEL90b] implemented a system called ALADDIN, designed as a teaching tool for data structures and algorithms, that integrates a graphical editor with the animation system to make the system easier to use for the unsophisticated programmer. Duisberg [DUI90] has experimented with an alternate interface that is gestural. Similarly, his goal was to design an interface that would make the task of animating an algorithm both simpler and quicker for novice users.

Much of the focus of algorithm animation has been for the benefit of students who are learning data structure algorithms. Recognizing that the needs and abilities of students differ depending upon their experience, McWhirter [MCW96] developed a multi-tiered algorithm animation system named AlgorithmExplorer. The first tier, intended for students in the first and second year of a computer science program, is designed to allow algorithm animation with minimal effort. The second two tiers are designed for more advanced students, consequently these tiers provide greater flexibility, but they also require more knowledgeable users.

### 2.3.3 Algorithm animation on the World Wide Web

Next, we consider algorithm animation systems that use the World Wide Web as a platform. The exponential growth of the Web and how quickly it has become widely known, cannot be ignored. The obvious advantage of putting algorithm animation on the Web is the ease of distribution and the vast audience. The development of the Java programming language has also sparked interest in the development of interactive applets that can be run from a Web page. Hartley [HAR96] has investigated making algorithm animations written with XTANGO available on the Web by adding an external viewer to the Web browser. His work with animation has been primarily in the context of teaching operating systems courses. Naps [NAP96b] who developed an algorithm animation system called GAIGS, has investigated five different methods of Web-based animation. The different methods differ depending upon where various parts of the system reside—on the client or on the server. Due to response time, his experience has shown only the methods where the animation system and the algorithms it animates reside on the client are currently feasible.

In the past several years considerable attention has been given to developing Web-based animation systems and porting existing systems to the Web. Brown and Najork [BRO96] have developed a Web-based algorithm animation system they call CAT, collaborative active textbooks. The unique features of their system are that it provides multiple views of an animation and it

provides support for collaboration. An instructor and a group of students can be viewing the same animation, each on their own machines. A significant limitation of this system was that it was written using Obliq active objects. Consequently, it could only be accessed from Web browsers that supported Obliq. To improve accessibility, a modified system called JCAT [BRO97] was developed that used Java instead of Obliq. A group of individuals, Haajanen *et al.* [HAA97], at the University of Helsinki, has developed a Web-based algorithm animation system called Jeliot. Their system is based on a theater metaphor and uses the concept of self-animation for selected data types. It animates programs written in a dialect of Java they have developed called EJava. Brummund [DER98] has compiled an index of Web animations that is available at `www.cs.hope.edu/~alganim/ccaa/ccaa.html`. It contains a large number of different algorithm animations and animation systems. We have included our system among the links.

Others have also created Web-based animation systems recently. Michail [MIC96] has developed a Java applet that implements a visual language designed for teaching binary tree algorithms. Naps and Bressler [NAP98] have ported their GAIGS system to the Web. Finally, Pierson and Rodger [PIE98] have developed a Java-based animation system entitled JAWAA, that contains graphical objects that are created in manner patterned after Stasko's Samba animation system.

### 2.3.4 CAI containing algorithm animation

The majority of algorithm animation systems have focused on developing an animation tool that is as general as possible, which as we have noted, is a challenging technical problem. Our research area, however, is in the intersection of CAI and algorithm animation. We found much less activity in this area. The one notable exception is a system developed by Gloor [GLO92] called AACE, Algorithm Animation for Computer Science Education. He developed an interactive version of a popular textbook on algorithms, by Cormen *et al.* [COR90]. This textbook is aimed at an advanced undergraduate or graduate data structures or algorithm course—somewhat different from our intended audience, however. Unlike any of the other systems we have discussed, this system was constructed from a multimedia authoring tool. He considered several tools, but he chose HyperCard. He rejected Macromedia Director, a popular multimedia authoring tool, as unsuitable for computer scientists developing algorithm animations. He paid considerable attention to the design of the user interface and regarded interactivity as one of his critical interface design requirements. His work was eventually transferred to a CD-ROM.

One other aspect of Gloor's work that we find particularly noteworthy is that he classifies algorithm animation into two categories: unified-view based and structured-based. The primary goal of animation systems that fall into the former category is to animate similar algorithms in similar ways. The advantage of this approach is it increases the generality of the animator. The majority of animation systems that we have discussed fall into this category. With his structure-

based method, each animation is developed independently to focus on the specific characteristics of that algorithm. We believe that this dichotomy is significant, because it highlights an important tradeoff in algorithm animation between the generality of the animation tool and the understandability of the generated animations.

### 2.3.5 Empirical studies of the effectiveness of algorithm animation

Although much research has been done in solving the technical aspects of algorithm animation, less research has been done to assess whether it is really an effective tool in computer science education. Next, we consider the empirical research.

Perhaps the earliest study, conducted by Badre *et al.* [BAD91], was an exploratory study that began with a faculty survey to determine the methods most faculty members used to teach algorithms and the type of graphic diagrams they used. In addition, they observed several students viewing an animation of the Shell sort algorithm with the intent of gaining insight in how to conduct more formal studies. We find it interesting that among their conclusions they suggest that it is important to consider the spatial abilities of students. They conjecture that poor visualizers might benefit most from animation systems. Despite this observation, we find no subsequent study that has considered this factor.

The first truly empirical study of algorithm animation was conducted by Stasko, Badre and Lewis [STA93]. Their study used a priority queue implemented with a pairing heap and involved two groups of students. One group was given a textual description of the algorithm. The second group was given text and the animation. Although their overall results did not suggest that animation helped significantly, they noted that the animation seemed to help more on the declarative and analytical questions, but not on the procedural questions.

In a subsequent study by Lawrence, Badre, and Stasko [LAW94], a minimum spanning tree algorithm was used. They found that viewing animations in either a classroom or laboratory setting did not significantly improve students' understanding. They did find, however, that there was a significant difference depending upon whether students were an active participant in the laboratory or not. Those participating in an active laboratory setting performed significantly better than those who participated in a passive laboratory. In addition, their results indicated that students who attended a laboratory session involving animation performed significantly better on the free-response test than on the questions that measured factual knowledge.

The animation of sorting and searching algorithms was the subject of a study by Crosby and Stelovsky [CRO95]. They divided questions into two categories that they referred to as textual or graphical. They concluded that the graphical questions better match multimedia instruction than do textual ones. In addition, they divided students into two groups according to their cognitive learning style as measured by the Myers-Briggs Type Indicator. Although this indicator contains four orthogonal dimensions, they considered only one of the four—the one that separates students

into sensing or intuitive. Their results indicated that students categorized as sensing were helped significantly by the animation, while the other group of students did not gain nearly the same level of benefit from the animation.

The benefit of algorithm animation with prediction was the subject of two experiments performed by Byrne *et al.* [BYR96b]. In their first experiment they used an animation of the depth-first search of a graph in a two-factor experiment, in which the presence or absence of animation were the two conditions of the first factor, and prediction or no prediction the two conditions of the second factor. They found some, but not a statistically significant, benefit to both animation and prediction, but they found no interaction between factors. Speculating that the first algorithm may have been too simple, they repeated the experiment with the animation of a more difficult algorithm—a priority queue implemented with a binomial heap. The second experiment did not yield the expected improved result.

Kann *et al.* [KAN97] conducted a study in which students were asked to implement the knapsack problem as a part of a lesson on recursion. One group of students was permitted to view an animation of the problem before attempting to implement it. The results indicated that those who viewed the animation demonstrated a significantly better ability to recognize recursive problems as measured by a posttest. They also concluded that integrating the use of animation into the overall learning experience is an important factor in its effectiveness.

Most recently, Kehoe *et al.* [KEH99] measured the effectiveness of algorithm animation in an open homework-style learning environment, in which students were provided the questions at the beginning of the experiment and there was no time limit. The animation that was used was the priority queue implemented with a binomial heap. The students who used the animations scored significantly higher than those who did not.

Considering these studies together, we can conclude that it appears that algorithm animation may influence learning certain kinds of knowledge more than learning other kinds or may help certain kinds of students more than others, but given the limited evidence, these conclusions are at best tentative. Most of the studies show that algorithm animation has no significant effect when type of knowledge and type of learner are undifferentiated. These results are difficult to reconcile with the strong positive evidence for the effectiveness of CAI in general. Gurka and Citrin [GUR96] speculate that many of the experiments that have been conducted have produced false negative results. Among the reasons they offer for these results include poor experimental design and poorly designed animations.

# CHAPTER 3

# INTERACTIVE DATA STRUCTURE VISUALIZATION COURSEWARE

The cornerstone of this research study is courseware entitled *Interactive Data Structure Visualizations*. The prototype of this courseware was developed during the spring of 1997 by the author. This Web-based courseware focuses on several key areas in the typical introductory data structures course. Its current Web address is `www.seas.gwu.edu/seas/eecs/research/visualization`.

## 3.1 Courseware development

We begin our discussion of this courseware by considering several developmental issues that we considered before its implementation. We begin with our requirements and then discuss our implementation considerations, and finally some high-level design decisions.

### 3.1.1 Key requirements

There were three key requirements that constrained the design of this courseware. The first requirement was that the courseware run on the World Wide Web. The benefit of such a requirement is clear—distribution problems are eliminated and accessibility is dramatically increased. By imposing this requirement, the possibility of extending its use and evaluation beyond students at The George Washington University became more readily feasible.

The second requirement was that the courseware should incorporate those features that have been found to be most effective in algorithm animation, in particular, and in multimedia, generally. The study by Lawrence, Badre, and Stasko [LAW94] suggested the importance of interactivity to the effectiveness of algorithm animation. Furthermore, interactivity has become widely recognized as a critical component to multimedia software generally. In her study of the educational use of virtual reality, Byrne [BYR96a] found that immersion, which is generally regarded as an essential component of virtual reality, did not significantly influence student performance, but interactivity did. For these reasons, we considered interactivity to be essential. Some of the studies we reviewed found narration concurrent with animation to be effective. Time constraints prohibited the inclusion of any audio in the prototype, but as we discuss later, the possibility of confounding effects led us to exclude it from the courseware used for our study.

The third requirement was that the courseware be easy to use. Although our intended audience consists of computer science students, who have a reasonably high degree of computer literacy, we still regard ease of use as an important requirement. A well-designed user interface

contributes to software that is easy to use. Consequently, we felt considerable attention should be given to the design of the user interface.

The final requirement was that the courseware have sufficient breadth to enable it to be used by students for more than a single session—preferably over a period of several weeks. As previously indicated, the three most fertile topics in data structures were chosen. These topics comprise about one fourth of the typical introductory course in data structures. In particular, it represents Chapters 10-12 and 14 of the textbook by Feldman [FEL97] that is currently used in CSci 131 at The George Washington University.

### 3.1.2 Implementation considerations

There was one important implementation consideration that needed to be made before this courseware was developed. It was what kind of software development tool to use. In any software development project, using the highest level tool possible is desirable because it reduces the amount of effort, therefore time and cost. There is a tradeoff to using such tools, which is loss of control. Capabilities that are not provided are generally very difficult to implement. Given that this courseware was intended to be multimedia CAI that uses algorithm animation, there were two possible high-level development tools we might have chosen.

Because our courseware was intended to be multimedia CAI, one possibility would have been to use an authoring tool, such as Authorware or Asymetrix Multimedia Toolbook CBT. Such tools have the advantage of providing many built-in capabilities useful in the construction of multimedia courseware, such as easily constructed animations, inclusion of audio and video clips, automatic tracking of correct and incorrect responses. In addition, most provide some capability for staging the resulting product on the Web. The drawback to using an authoring tool for our purposes is that although simple animations are easily implemented with them, complicated animations of the kind we required are much more difficult. In addition, such tools generally require plug-ins in the user's Web browser and such plug-ins are not always provided for all platforms.

The other option we considered was using an existing algorithm animation system. The advantage to such an approach is the process of designing the animations would be greatly simplified. There are several disadvantages, however. Most of these systems were designed for high-end work stations because of the need for speed and high resolution graphics, although some are being modified to be more platform independent. The other consideration was the design of the user interface. Using an existing system would constrain the user interface to be that provided by the animation system. To maximize ease of use, we felt a more customizable interface was needed. Moreover, the degree of interactivity and feedback we desired would have been unavailable. Furthermore, not all of our lessons are animations—several are nonanimated visualizations. Finally, the ability to tally correct and incorrect responses would have been unavailable.

The courseware we developed was somewhere between typical CAI and algorithm animation, so neither tool was a good fit. Consequently, our choice for the animations and visualizations was a general purpose object-oriented language that provided a platform independent GUI toolkit—Java. Besides the animations, a tutorial—primarily text—was required to explain the animations. Hypertext mark-up language (HTML) was the obvious choice for implementing this tutorial. Finally, to integrate the HTML tutorial with the Java applet, JavaScript—a scripting language—was chosen.

Our final consideration was how general to make the courseware we were developing. It was not our intent to write another general purpose algorithm animation system, since numerous such systems had already been developed. At the time we began our research, however, none had been ported to the Web. As expected, many have been since that time. Our primary intent was to create Web-based courseware that animated specific algorithms and most importantly incorporated a kind of interactivity that no existing algorithm animation system currently supports. Developing sufficient courseware to test our hypotheses was a lower cost solution than attempting to incorporate this feature in existing algorithm animation systems, and certainly less costly than developing a new system that incorporates it.

### 3.1.3 High-level design

One of our key requirements was developing easy-to-use courseware. As we noted, a well-designed user interface is an essential component of easy-to-use software. Because of its importance, we have given considerable attention to the user interface design. Our outermost interface has evolved through three different designs. In our initial version, used by students in CSci 131 during the spring 1997 semester, the courseware consisted of a single Java applet running in a separate frame. High-level control resided in the menus associated with that window. The second iteration, that incorporated the initial tutorial and the corresponding Ada code, required the high-level control be moved outside the applet to the Web pages. A multiple frame interface was selected, so that all possible navigation is clearly present. This approach had one drawback—requiring a higher resolution monitor to view all frames in their entirety. This design consists of two physically orthogonal menus. Vertically, the eleven different courseware lessons are listed. Horizontally, the menu lists the tutorial, the visualization, and the Ada code. The orthogonal form of the menu was chosen because it reflects the function—all combinations of the horizontal and vertical menus are meaningful. Our final refinement combines the best of both. It uses multiple frames in the main browser window, but displays the visualization applet in a separate window. This approach makes the entire applet window visible in low resolution monitors and makes it possible to view the visualization and either the text or code simultaneously on high resolution monitors. This final interface design is shown in Figure 3.1.

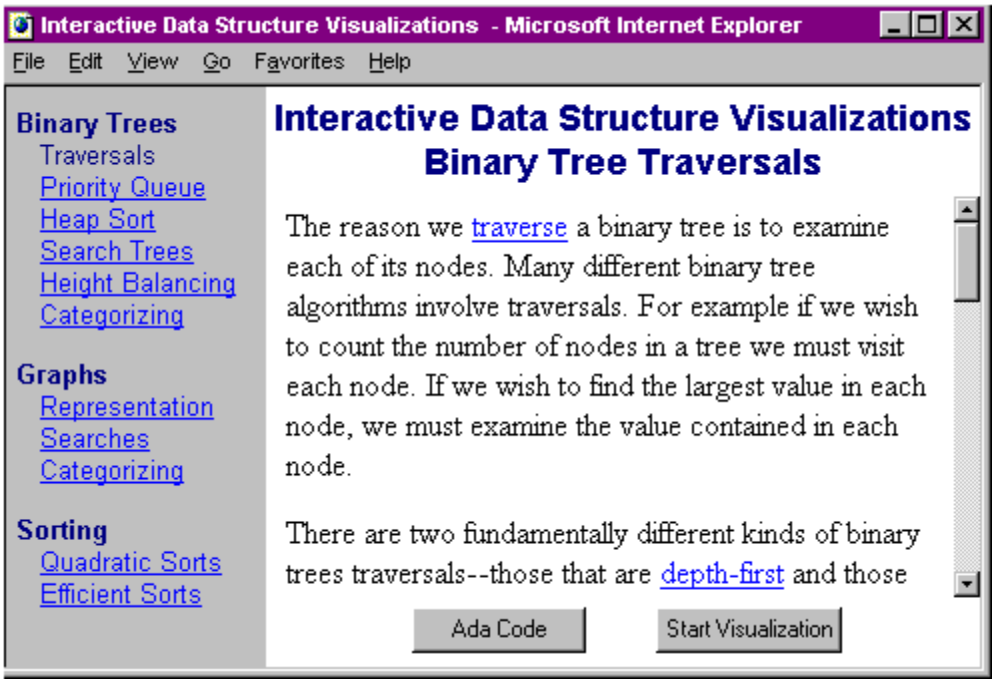



Figure 3.1 Overall user interface design

Besides carefully considering the design of the outermost interface, we also gave considerable thought to the user interface of the applet that performs the visualizations. The concept of abstract data types is an important topic of most CS 2 courses—particularly those taught in Ada. Ada packages allow abstract data types to separate the specification from the body. We attempted to capture this separation in the design of the applet. The buttons reflect the specification—each button initiates one of the operations of the abstract data type. The visualization represents the implementation. Figure 3.2 illustrates the applet-level interface for the binary tree traversal lesson.

The final high-level design feature we discuss is interactivity. The adjective *interactive* is included in the title of our courseware because it incorporates various aspects of interactivity. The most basic aspect of interactivity is the navigational interactivity. Students are allowed to select an operation from buttons along the right side of the applet window. At the bottom of the applet window are controls to allow the speed of the animation to be controlled. It can be paused, resumed and run in single-step mode.

Another aspect of interactivity is the ability to be shown solutions with its *Show Me* mode and to interact with the system and try to replicate the steps of an algorithm in the *I'll Try* mode. In a recent study, by Byrne *et al.* [BYR96b], the value of students predicting the next step of an algorithm was shown to be of some benefit. Rather than stopping an animation and guessing the next step, our *I'll Try* mode allows students to enter a purely interactive mode. A pair of radio buttons allows students to switch between modes. The type of interaction that is expected in the *I'll Try* mode, differs depending upon the visualization. In the binary tree traversal or graph search

lessons, students are asked to click the node that would be enumerated next in the traversal or search. In the sorting algorithms that are implemented with swaps, students are asked to click the pair of nodes or bars to swap in the next step of the algorithm. In the lessons that do not involve animations, the interactions are different. In some cases, such as the recognition of AVL trees or undirected trees, students are asked to choose from a pair of buttons, one that indicates that the data structure shown satisfies the property or a second button indicating that it does not. In other cases, such as the recognition of complete trees, students are asked to either click a missing node if the tree is not complete or to click a button if it is complete. The final type of interaction involves data input. In the height-balanced tree lesson, students are asked to input the height of a subtree or the balance factor of a node. Aleem [ALE98] has developed a taxonomy of interactivity with several levels: passive, reactive, proactive, and creative. The *I'll Try* feature constitutes reactive interactivity, because the student is expected to react to the prompt that asks for the next step of the algorithm.

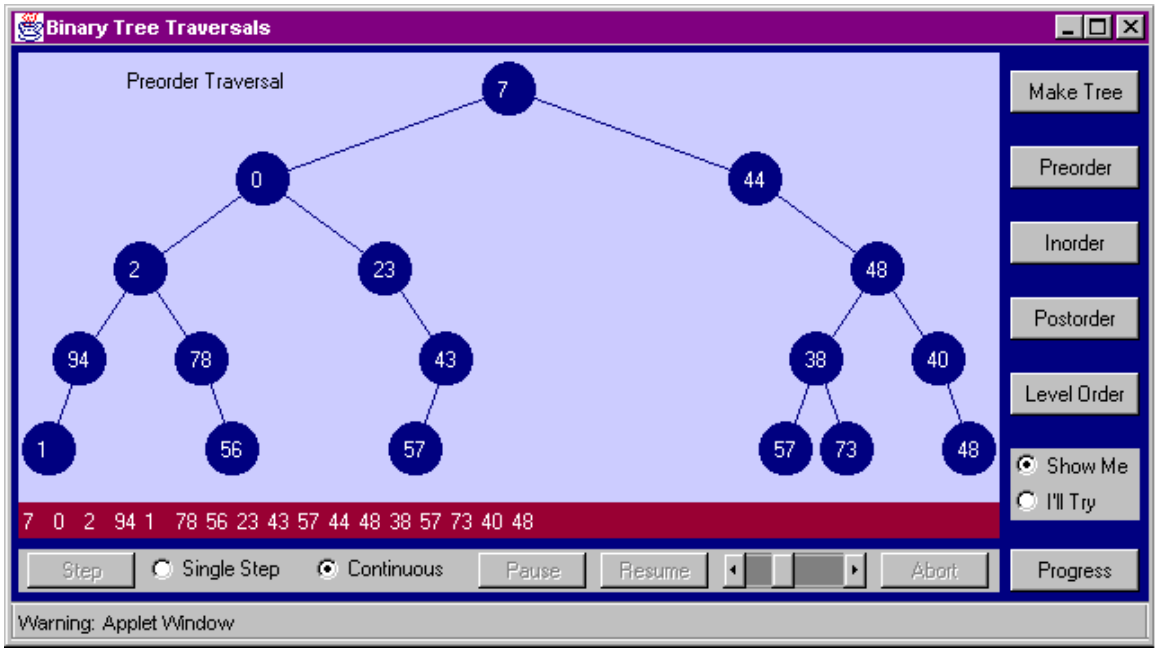



Figure 3.2 Binary tree traversal lesson

An additional aspect of interactivity is provided by a second pair of radio buttons for selected lessons. This pair of buttons allows the choices of *Random* or *I'll Pick*. It is provided in the priority queue, binary search tree and graph search lessons. The *Random* selection causes the data values to be randomly selected in the first two lessons and the *I'll Pick* selection allows students to choose the data value. This additional control allows students to enter an exploratory learning mode. In the case of the graph search lesson, *I'll Pick* allows students to pick the starting node of the search. This capability would be considered proactive according to Aleem's

taxonomy, because a degree of learner construction is involved. The student can create the binary search tree by choosing its values.

## 3.2 Courseware content

The subject areas included in our courseware are binary trees, graphs and sorting. The term visualization is used rather than animation because it includes both animations and nonanimated but highly visual lessons. One example is a lesson designed to assist students in identifying different kinds of binary trees, such as complete trees, binary search trees, heaps and so on. This lesson contains no animation, but is highly visual, nonetheless. Eleven lessons are included in the prototype—six on binary trees, three on graphs, and two on sorting.

### 3.2.1 Binary tree lessons

Binary trees are a visually rich topic. The six lessons that we have chosen encompass all the major aspects of this topic. Four of the six binary tree lessons are animations. For simplicity, whenever data values are required in the binary tree nodes, nonnegative integers of at most two digits are used.

The first animation illustrates four binary tree traversals. Trees of random shapes containing random values are generated. The traversals include preorder, inorder, postorder, and level order. The traversals are animated by a circle that traces the path of the traversal. When the data value of a node is visited, its value moves to a list of values at the bottom of the screen. All the recursive traversals follow the same path, so understanding the difference among the three requires understanding that in a recursive traversal each node is touched three times, and in a preorder traversal that value is visited the first time, for inorder the second, and postorder the third. In the version used by students during the spring of 1997, we used three colors to indicate the three times it is touched.

The second animation illustrates a priority queue using a heap. The two operations that are provided are enqueue and dequeue. An important idea that students must understand to grasp all uses of heaps is that the array representation corresponds to the level order enumeration of the binary tree heap representation—a topic from the previous lesson. Consequently, both representations are illustrated. The sifting process up or down the tree is illustrated by the swapping of values in both representations when a dequeue or enqueue operation is performed. Random values are chosen, but the option is provided for the student to pick a value for the enqueue operation.

The third lesson is an animation of a binary search tree. The operations provided are the most basic ones—insert, delete and find. The animation of the insert operation uses a circle to trace the path followed in the process of insertion. Because a fixed-sized graphic is used, tree

height is limited. If the node to be added would exceed the maximum tree height, an appropriate message is displayed. Similarly, because duplicates are prohibited, an error occurs when a value already in the tree is to be inserted. The find operation traces the search path in a similar manner—a message indicates whether the value was found. The delete operation is algorithmically the most complicated of the three operations. Deleting a leaf node, a node with one child or two children require separate cases. Leaf nodes are the simplest, while nodes with two children are the most difficult. To allow the student to experiment with various cases, the option to have data values randomly generated or selected by the student is provided for all operations.

The fourth animation is the heap sort, which we choose to include among the binary tree lessons, although conceptually it fits equally well among the sorts. As with the heap used for the priority queue lesson, both the tree and array representations are illustrated. The heap sort algorithm has two distinct phases—transforming the initial tree into a heap, then successively extracting the largest value and transforming the resulting *almost heap* into a heap. Visually conveying these steps is the one of the key challenges of this animation. We have used color for this purpose. We did not attempt to determine whether such color coding was effective in conveying this information in the first semester's evaluation, because we did not believe students had used the courseware enough to make such judgments. We did, however, assess its effectiveness in subsequent semesters.

The remaining two binary tree lessons are visualizations rather than animations. The first of these two is designed to assist students in understanding the concepts of tree height or depth, and the concept of balance factors and height-balanced trees—a topic often introduced, but not fully explored in an elementary data structures course. Like the previous binary tree lessons, trees of random shape are generated, but no data values are included, because they are not required for these topics. Clicking the height button causes each node to be labeled with the height of its respective subtree. When students access the height operation in the *I'll Try* mode, they are prompted for the subtree heights in postorder—to help them understand that the height of a tree depends upon the height of its subtrees. Clicking the button for balance factors causes each node to be labeled with its balance factor. Finally, there is a button that determines whether a tree is an AVL tree and one that determines whether it is height-balanced *n*, for a value of *n* that the student supplies. If a tree does not fulfill the definition, the unbalanced nodes are indicated.

The final binary tree lesson, which is the second of the two visualizations, involves categorizing binary trees. There is a button to test whether the tree on the screen qualifies for the following binary tree categories: almost complete, complete, binary search tree, heap and AVL tree. If the tree meets the definition an appropriate message is displayed, otherwise an indication of the reason for failing the definition is provided. If a tree fails to meet the definition of an almost complete or complete tree, all the missing nodes are shown in yellow. If a tree fails to meet the binary search tree definition, the lowest subtrees that are not binary search trees are highlighted. If

a tree fails to meet the definition of a heap all offending branches are highlighted. For AVL trees, an appropriate message is displayed indicating whether the tree is an AVL tree.

### 3.2.2 Undirected graphs

There are three lessons on undirected graphs, two animations and one visualization. The first lesson is an animation illustrating the adjacency representation of graphs. A graph with exactly twelve nodes is used. Both the graph and the matrix are displayed. The two provided operations are named *Make One* and *Create Other*. In addition, there is a pair of radio buttons labeled *Matrix* and *Graph*. If *Matrix* is selected, *Make One* randomly generates a matrix, and *Create Other* transforms the matrix into its corresponding graph. If the radio button *Graph* is selected, the action of the buttons is reversed. The applet window for the graph lesson is shown in Figure 3.3

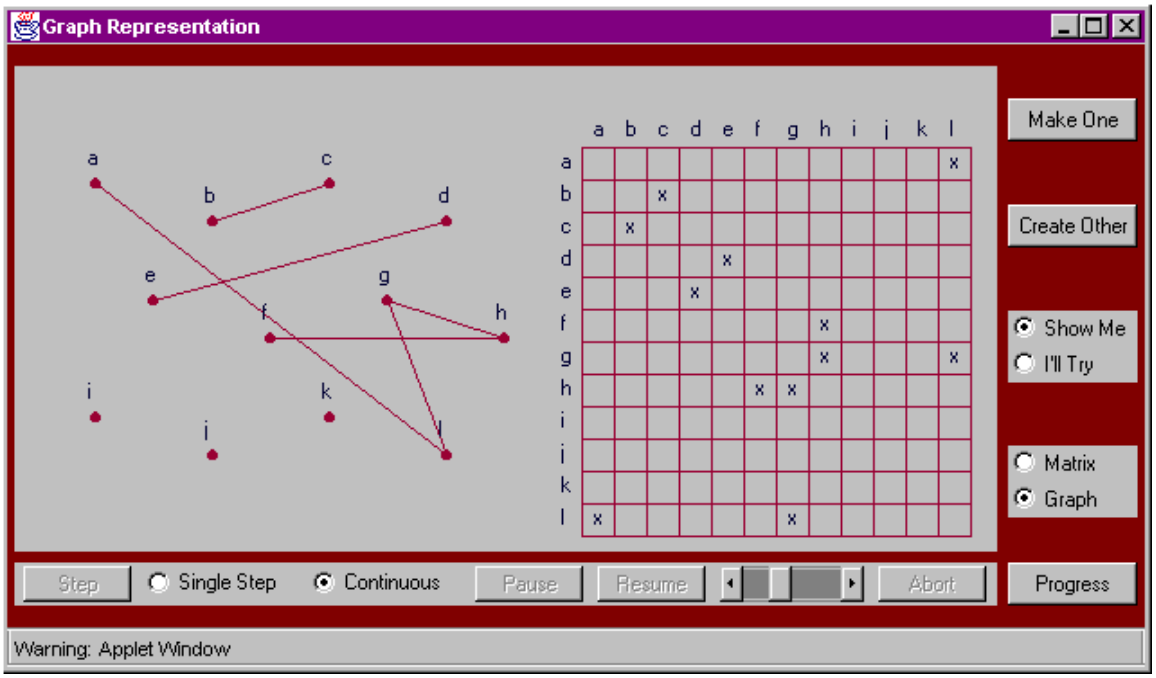



Figure 3.3 Graph representation lesson

The second lesson is an animation illustrating depth-first and breadth-first graph searches. Graph searches are similar to binary tree traversals, but there are some important differences. The starting point for a tree traversal is always the root. Graph searches have no fixed beginning so students are permitted to select the starting node. Because graphs can contain cycles, it is necessary to mark each node as it is visited to avoid endless loops. We illustrate this marking with a black X. A depth-first search is naturally recursive like the recursive tree traversals, but unlike a tree, graphs lack orientation, making it more difficult to see when the algorithm is moving into or out of the recursion. We regard this difference as important. In the version used during the spring

1997 semester, we attempted to distinguish between the directions with different colors. One final difference exists between tree traversals and graph searches. In a tree traversal, it is customary to always visit the left child before the right, with a graph search, there is no established order for visiting neighboring nodes. Consequently, a breadth-first graph search, unlike a preorder tree traversal, is not unique. Because we need to allow the student to choose any adjacent node in the *I'll Try* mode, the underlying algorithm for the breadth-first search must be implemented using a queue of queues, rather than the simple first-in first-out queue required for the standard graph search algorithm. The applet window for the graph search lesson is shown in Figure 3.4.

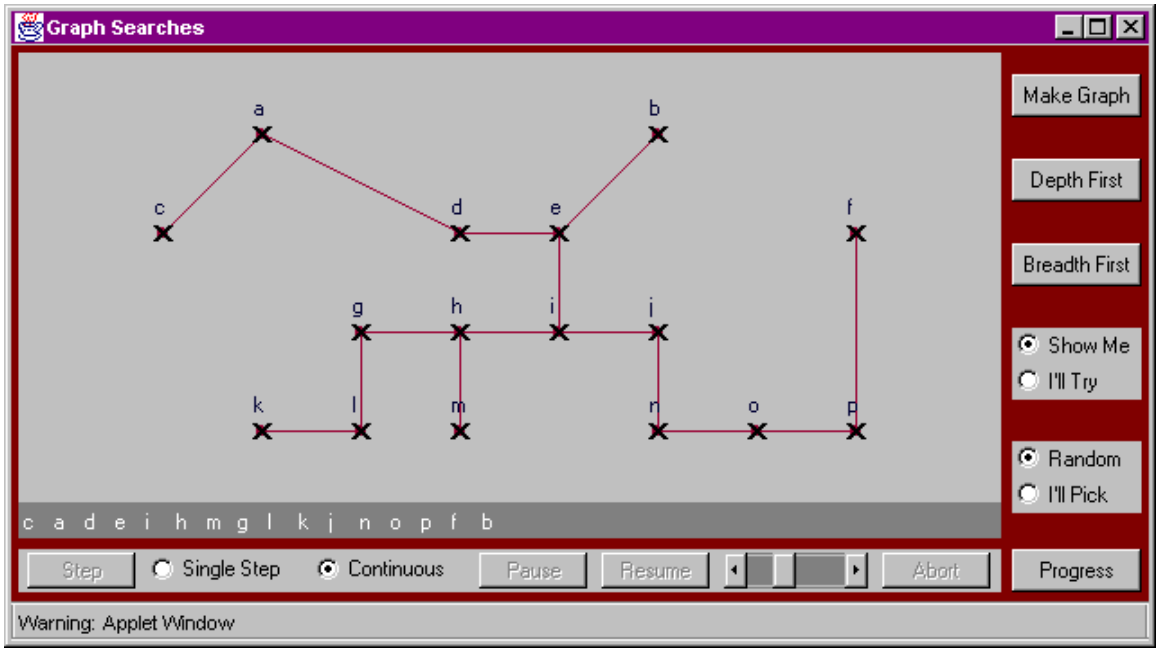



Figure 3.4 Graph search lesson

The last of the three lessons is a visualization that illustrates three graph properties: whether it is connected, whether it contains cycles and whether it is an undirected tree. Disconnected graphs are indicated by highlighting all nodes that are disconnected to one designated node. Graphs that contain cycles are indicated by highlighting the edges of the cycles in white. Graphs that are not undirected trees are indicated by an appropriate message. To properly generate an interesting random selection of graphs for this lesson, a random set of nodes was generated, a spanning tree was created, then randomly either one or two edges were removed—producing a disconnected graph, it was left unchanged—producing an undirected tree, or one or two edges were added—creating a graph with cycles.

### 3.2.3 Sorting

Finally there are two animation lessons on sorting. Sorting algorithms have been animated with either scatterplots, horizontal bars, or vertical bars. A study of user preferences, by Morris *et*

*al.* [MOR], has shown that there is a strong preference for bars over scatterplots and a slight preference for vertical bars compared to horizontal. Consequently, we adopted vertical bars together with the numerical value beneath the bar that its length represents.

The first lesson is an animation of three quadratic sorts: the insertion sort, the selection sort and the bubble sort. We animate the movement of both the bars and their corresponding numbers to emphasize their connection. Color is used to distinguish sorted values from unsorted and to highlight the values that are being swapped. The applet window for the quadratic sort lesson is shown in Figure 3.5. The second sorting lesson illustrates three of the more efficient sorts: one $O(n\sqrt{n})$ sorting algorithm—the Shell sort, and two $O(n \log n)$ sorting algorithms, the merge sort and the quick sort. The animation of the merge sort is the most unusual of the six. To visualize the merging step, we use a temporary array to copy to while merging the subarrays. Animation of sorting is nothing new, but the *I'll Try* mode, as in all the previous lessons, offers students the opportunity to reproduce the steps of the algorithm as a method of testing their knowledge.

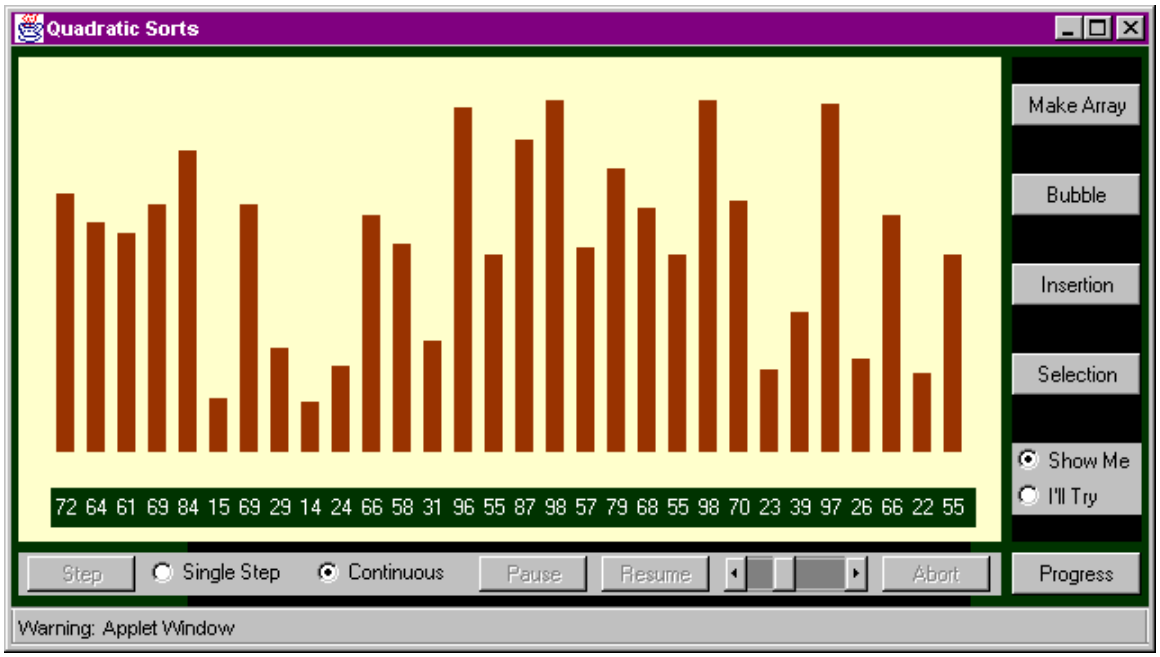



Figure 3.5 Quadratic sort lesson

## 3.3 Incremental courseware modifications

After the courseware was used during each semester, changes were made as a result of evaluations by students, student observations, logged information of usage, and audio recordings of a student thinking aloud while using the courseware. The details of this data are discussed in Chapter 5. Next, we consider the software changes that occurred at each of these intervals as a result of this input.

### 3.3.1 Modifications following spring 1997 semester

The primary problems with the courseware experienced by the students during the spring 1997 semester were the speed issues—response time and download time. Further investigation confirmed that the response time problem was a problem with version 3 of Netscape Navigator on the PC—the browser almost all students used. This problem was not present in the earlier 2.1 version of Netscape Navigator, nor is it present in the subsequent version 4. All versions of Internet Explorer executed the courseware without problem. The download speed problem was resolved by using a Java archive (JAR) file. A JAR file is created by zipping all the class files into a single file. In this way, only one JAR file needs to be downloaded instead of over 60 class files. Initially, Netscape Navigator supported the use of these archive files, but Internet Explorer 3 did not. Now, both browsers support it.

Two major enhancements were deemed necessary based on the suggestions of the students who experimented with the first version of the courseware. The first of these enhancements was to include more controls that regulate the animations. Such controls are typically a part of most algorithm animators. The initial version had only speed control. A pair of radio buttons was added to select between a single step mode and continuous mode. In the single step mode, a button allows the user to step through the animation. In the continuous mode a pair of buttons allows the user to pause and then to resume the animation. A button that aborts the animation is accessible in either mode. All these controls together with the speed control were placed in a control panel below the animation panel. The illustrations in Figures 3.1-3.4 are all from this modified version—consequently they contain this control bar.

The second major enhancement is the ability to submit on-line homework assignments. It is important to give students a reason to use courseware. In a recent study of the effectiveness of intelligent tutoring systems by Tjaden [TJA98], less than half the students used the courseware. During the spring 1997 semester, we had a comparable experience. To improve participation, the homework assignments for the last four weeks, which had been submitted on paper were replaced with this on-line homework for the fall 1997 semester. The on-line homework makes it possible for each student to have unique problems, because they are randomly generated. Having chosen the Web as the platform for our courseware enables us to have the homework electronically submitted and recorded by our Web server. To implement this enhancement, an additional button was included in the lower right corner of the visualization window labeled *Progress.* This button alternately shows or hides an additional window. This window contains a table showing the user's progress. The operations associated with the lesson form the rows of the table. The columns consist of the number of examples shown, the number of correct responses, the number of incorrect responses, and the number needed before the assignment can be submitted. The progress window is shown in Figure 3.6. When the needed column is all zero, the submit button

is enabled. In the *I'll Try* mode, access to operations that have already been shown or tried is prohibited.


Figure 3.6 Progress window

Besides the major enhancements, several minor modifications were deemed appropriate after the spring 1997 semester. Like the enhancements, the first of these modifications was also in response to the evaluations by students. Several students had remarked that the graphs looked weird. Purchase [PUR97] has studied graph aesthetics and developed several metrics for judging the aesthetic appeal of graphs. They include minimizing edge crosses and edge bends and maximizing symmetry—ideally by fixing the edges to an orthogonal grid. Eades and Xuemin [EAD89] have also proposed aesthetic criteria that include the even distribution of the nodes. The students were right. According to these metrics, the graphs did look weird. A new algorithm was developed for generating the graphs for the graph search and graph categorization lessons. First, the nodes were aligned to an orthogonal grid. Second, the algorithm for adding edges beyond the minimum spanning tree was modified to eliminate crossing edges. The graph representation lesson was modified slightly to ensure that no edge bends were needed, by ensuring that no edge would intersect any nodes other than its endpoints. These graphs could be as small as an empty graph or as large as a complete graph, so minimizing edge crossing was not considered.

The second modification arose as a result of the enhancement to allow students to use the courseware for on-line homework assignments. In the first version, the graph representation lesson did not have the *I'll Try* and *Show Me* modes. This dichotomy was essential to the implementation of on-line homework facility. This lesson was redesigned to be more consistent with the other ten lessons.

The final modification resulted from the review of the proposal for this dissertation. In the initial version of the courseware, color was used to convey the various times a node is touched during the recursive tree traversals. Check marks were recommended to replace the color scheme. In addition, color was initially used to convey the direction—into or out of recursion—in the graph search lesson. In place of color, an arrow was used to illustrate direction.

### 3.3.2 Modifications following the fall 1997 semester

The evaluations by students again provided the basis for the major changes that were made to the courseware after the fall 1997 semester. Three major changes were made at this point. The first change was to improve the user interface by adding a help facility. A button was added to the control panel to turn on the help feature. With help turned on, whenever the mouse is moved over any of the controls, a message appears on the screen describing the function of that control. It is state sensitive, so, for example, if a control is disabled, a message explains why it is disabled. Different messages also are displayed for some of the controls depending upon whether the *I'll Try* or *Show Me* is selected. Figure 3.7 illustrates this help facility in action.

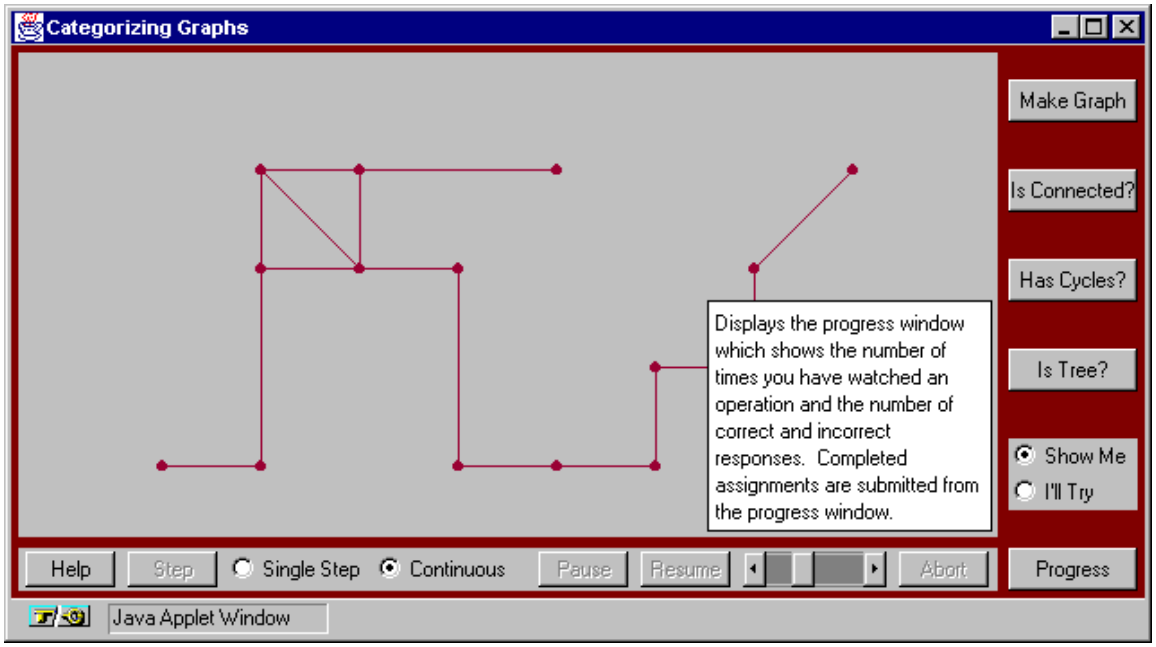



Figure 3.7 Applet window showing help feature

The second major change that was made to the courseware was to improve the understandability of the sort animations, which several students had suggested. Clearly, the efficient sorting algorithms are less intuitive. These algorithms were the ones that needed some additional visual cues. To the merge sort, we added lines beneath the bars to illustrate the levels of recursion. Cox and Roman [COX92] have categorized the representations that appear in algorithm animation according to their degree of abstraction. The bars that represent the values of an array

are examples of the most concrete category they term direct representation. The lines that represent the levels of recursion are more abstract. They categorize such representations as structural representations, which is their second level of abstraction. We made several changes to the quick sort. The bar that was the pivot was highlighted in white. A line was drawn horizontally at the height of the pivot bar. It crosses all bars to which the pivot is compared. Two arrows were placed below the bars to represent the position of the successive comparisons. Figure 3.8 illustrates the quicksort in action. Finally, we decided to remove the Shell sort for two reasons. First, it is less important since its efficiency is only $O(n\sqrt{n})$, and second it was difficult to find good visual cues to describe its behavior.

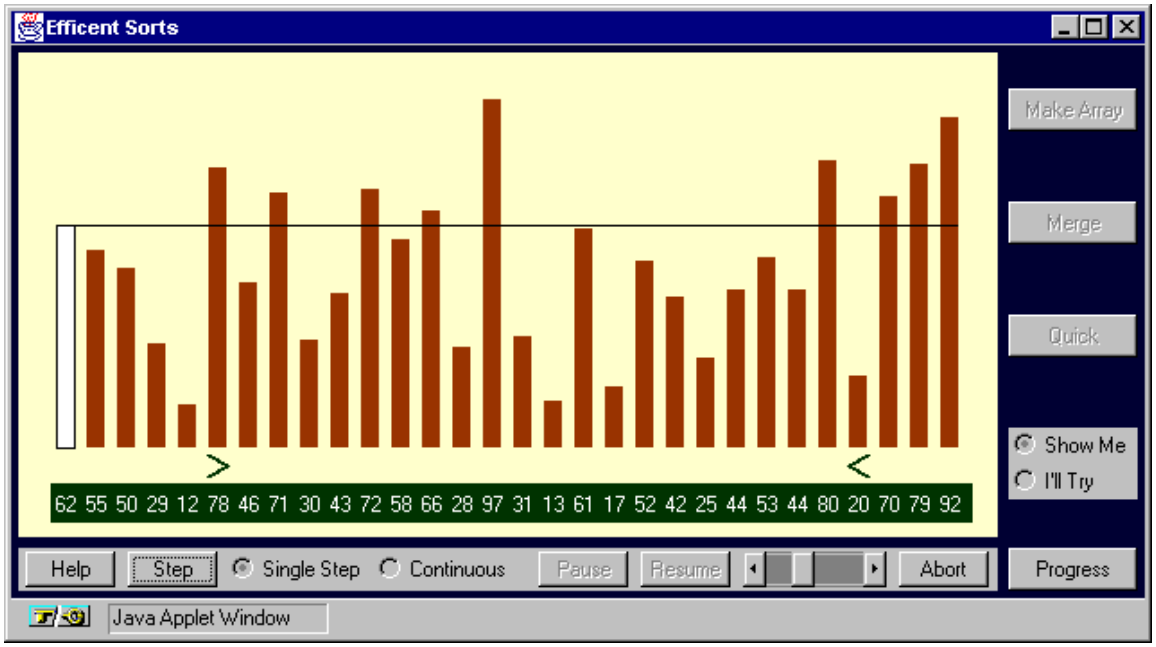



Figure 3.8 Quicksort with visual semantic cues

The final major change was the addition of a feature that logs all button clicks. We felt that a log of these clicks would be the most reliable measure of the depth of use, more reliable than the questions we had added to the evaluation given in the fall 1997 semester. Furthermore, it would describe how the courseware was used. Each button click was time stamped, and periodically these transactions were sent back to the server and saved in a log file.

Several minor changes and corrections were made at this juncture. Although, some changes were made after the first semester, students still felt that they needed to click too exactly for a response to be considered correct. Changes were made to increase the tolerance. The single step mode had been used more extensively this semester than the first, so some problems were discovered. These were also corrected.

### 3.3.3 Modifications following spring 1998 semester

Observing students using the courseware in the laboratory provided the basis for most of the changes that were required following the spring 1998 semester. One major change was made as the result of these observations. From observing the students it was clear that they were confused by the expected interaction in the nonanimation lessons. To correct this, the user interface was modified one final time to add an interaction panel at the top of the applet window. This panel contains three elements. The left-most element displays a prompt instructing the student what action is expected. If keyboard input is required, a text box appears within the prompt element. The middle element is used to display answers in the *Show Me* mode and buttons to click as responses in the *I'll Try* mode. The right-most element is used to display messages—such as those confirming a response is correct. As a part of this user interface redesign, the help button was moved from the control panel to a more appropriate location in the upper right hand corner of the applet window. This final user interface design is illustrated in Figure 3.9

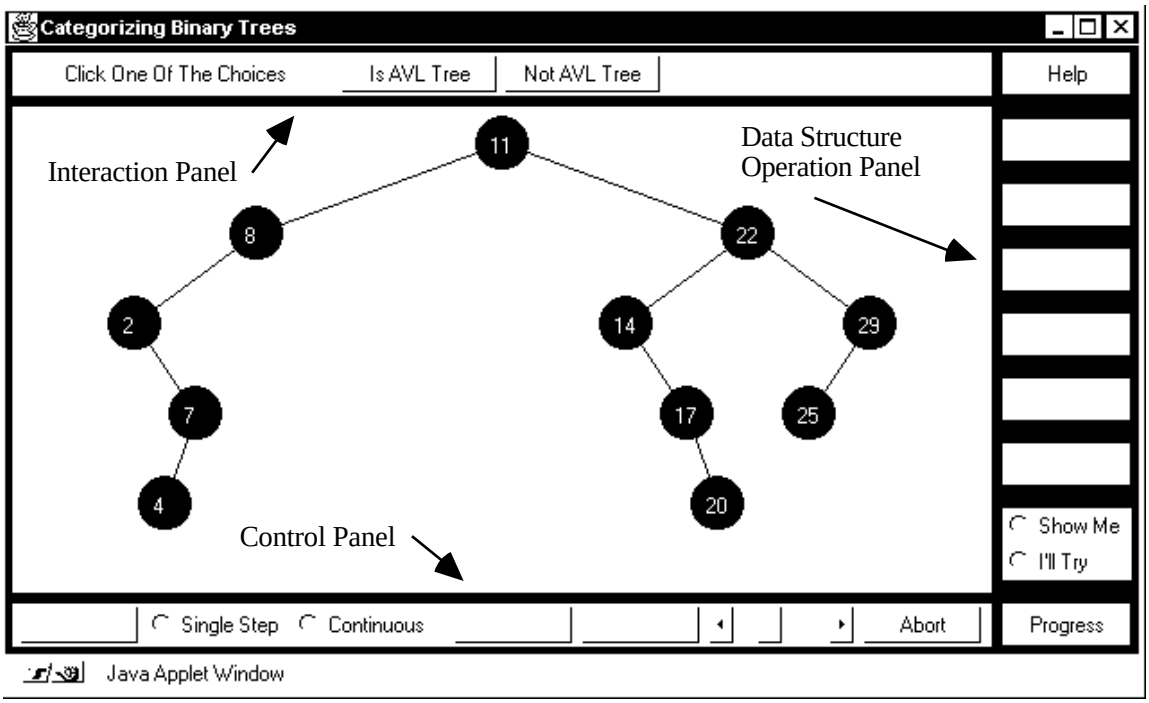



Figure 3.9 Applet window containing interaction panel

Several minor changes were also made as a result of the observations. Numerous students were frustrated by having to complete what they felt was an excessive number of correct responses on certain lessons. The courseware was modified so that number of correct responses could vary from lesson to lesson and could be adjusted to a level that demonstrated mastery of the topic. To prevent guessing, the algorithm for submitting a completed assignment was modified so that every incorrect response increased the amount needed by two.

The feedback given when clicking on any part of the data structure was improved, so that not only does a message appear indicating whether it is a correct response, but when correct responses are clicked a more physically immediate change occurs, such as changing the color of a tree node or a number. This modification is most helpful when the interaction required is a swap and requires two clicks. After the first click, the clicked object is highlighted by the color change.

One change was made as a result of the think-aloud protocol that we will discuss further in Chapter 5. This change was to help clarify the delete operation in the binary search tree lesson. The algorithm for deletion of a value from a binary search tree is most complicated in the case where the node containing the value to be deleted has two children. In this case, the value in its inorder successor must be moved into that node, and the node that contained the inorder successor must be deleted. To help make this relationship more apparent, the branch between the node containing the value to be deleted and its inorder successor is highlighted at the time the value is moved.

The final change that was made was a change to the logging feature that had been incorporated after the previous semester's usage. After analyzing the logging data collected during the spring 1998 semester, we felt that too much detail had been collected. The logging feature was modified to tally the button clicks as they occurred and log the total rather than each click.

### 3.3.4 Modifications considered but rejected

There was one enhancement that was suggested by several students in the evaluation during three of the four semesters. This enhancement was the addition of audio. Although our students suggested adding music or narration, our preference would have been to use sound to convey meaning by using auditory icons. According to Gaver and Smith [GAV90], auditory icons can be used for three purposes: confirmation, state information, or navigation. We believe the first two of these three possibilities could have been applied to our courseware. Clearly confirmatory sounds could have been used in the *I'll Try* mode to confirm correct responses. What is more important is that there are numerous opportunities to convey state information. In the depth-first binary tree traversals, we wish students to understand that each node is touched three times. Sound might have enhanced the understanding of this topic. In the depth-first graph search we wish to distinguish between processing that moves into the recursion and processing that moves out. Again sound might have assisted in understanding this difference. Gaver [GAV93] suggests that information can be encoded into auditory icons through parameterization, for example, large sounds for large objects. It has been shown by Brown *et al.* [BRO89] that users can rapidly extract multiple pieces of information from sound. Blattner and Greenberg [BLA89] believe that another benefit to the use of sound to convey information is its ability to transcend specific languages. According to Minghim and Forrest [MIN95], the greatest difficulty with using sound to convey information is that auditory perception has been studied much less than visual

perception. Brown and Hershberger [BRO92], who have incorporated sound into their animations, found sound much more difficult to use than color. Because of this difficulty and the possible confounding effect that sound might have had on our experiment, we chose not to add it. Nonetheless, we do believe that audio has an important role to play in algorithm animation and we shall discuss this topic again when we discuss the future work that might be done.

# CHAPTER 4

# RESEARCH PLAN AND EXPERIMENTAL DESIGN

We regard the question of whether algorithm animation is an effective tool for computer science education as an open question. There have been too few studies thus far to warrant a definitive answer, and as often happens in such research, the results have been mixed. It is our hope that our study has moved us one step further toward an answer to this question—a question we regard as an important one, particularly in view of the new possibilities that multimedia and the World Wide Web present to computer science educators.

In their assessment of the problems associated with testing the effectiveness of algorithm animation, Gurka and Citrin [GUR96] note that a judicious combination of quantitative and qualitative approaches is likely to be the most effective. In our study, we have followed this recommendation. We have collected many kinds of data—both quantitative and qualitative. Furthermore, our study consists of a set of primary questions that are objective in nature, and a set of secondary questions that are more subjective. The primary questions regarding the influence of interactivity and learning styles will be answered mainly by quantitative data—posttest scores and time spent using the courseware. To gain a deeper understanding of the results, however, we have also gathered qualitative data, in the form of written evaluations and transcriptions of students thinking aloud while using the courseware. The secondary questions that assess how well the courseware is accepted, whether the Web is an effective platform, and how understandable the animations are, will also be answered by a combination of quantitative and qualitative data.

## 4.1 Primary research questions

Let us begin with a discussion of how we formulated our primary research questions and how previous studies on CAI and algorithm animation influenced this decision. In our earlier discussion of the previous research we noted that the meta-analytical studies of CAI have shown a demonstrated positive, but admittedly moderate, effect. Such a clear pattern has not yet emerged, however, in the empirical studies of algorithm animation. We cannot hope to definitively reconcile this difference, but we offer several possible explanations. The first possibility, perhaps the most likely, is simply that the small number of studies has been inadequate to perceive a clear effect. We must remember that the meta-analyses we reviewed in many cases summarized the results of several hundred individual studies. Another possibility is that we are seeing a manifestation of the observation that CAI is significantly less effective in the hard sciences, an effect noted by Kulik and Kulik [KUL86]. The final possible explanation we offer is that algorithm animation alone—outside the context of CAI is not sufficient to produce a favorable result. There is some evidence

of this possibility. The study by Crosby and Stelovsky [CRO95] showed the most favorable results did not use a general purpose algorithm animator, but used algorithm animation in the context of multimedia courseware. The earlier studies do not.

Before discussing our primary research questions and experimental design, there is one important characteristic of most prior studies that warrants further discussion. In the majority of these studies, CAI has been compared to traditional classroom instruction. We find several problems with such comparisons. The first problem is the presumption that CAI is intended to replace traditional classroom instruction. One clear opinion that emerged from the evaluations of our courseware, that we discuss later, was that although students liked it and indicated they would like to see more such courseware, most did not think such courseware should replace classroom instruction. Tjaden's [TJA98] students expressed a similar opinion. We believe that the primary role of CAI is similar to that of textbooks—to augment not replace traditional instruction. The second problem with comparing CAI with traditional instruction is the difficulty of ensuring a fair comparison. In his discussion of the confounding factors in the meta-analyses of CAI, Clark [CLA85] observes that when we consider only the studies in which the CAI is designed by the same individual that provides the classroom instruction, the differences are negligible. The final problem with comparing classroom instruction with CAI concerns studies that attempt to identify differences among students of different learning styles—an issue of particular importance to our study. Let us reconsider the study by Crosby and Stelovsky [CRO95] that found that the Myers-Briggs category of sensors benefited significantly more than the intuitors. One cannot definitively conclude that the animation contributed to this observed difference. Another possible explanation might be that sensors generally have difficulty with classroom instruction and learn more from self-paced individual instruction. Although comparison of CAI with classroom instruction has been the norm, we reject it in our study for these reasons. There are simply too many potentially confounding factors when making such disparate comparisons.

Consequently, we did not compare our courseware with classroom instruction. Instead, we compared two versions of the courseware—a version that contains the interactivity provided by the *I'll Try* feature and a version that did not include it. The *I'll Try* feature is the unique aspect of our courseware. It is the feature that no general purpose algorithm animation system provides.

Finally, McWhirter [MCW96] observed that there are three ways that algorithm animation can be used in an educational setting (1) in the classroom lecture (2) in laboratory exercises and (3) by having students implement algorithm animations. In our research, we chose to focus on the second use. We avoided the first option because most classrooms still lack the necessary hardware and bringing such equipment to the classroom can be problematic. Although having students implement algorithms has been shown to be effective in advanced data structures courses [STA97], we did not believe it was suitable for students in an introductory data structures course. Our intent was to explore the use of algorithm animation courseware in the laboratory setting.

### 4.1.1 Research hypotheses

Let us now state our primary research questions more explicitly. We begin with our first null hypothesis:

$H_{01}$: Students who use the version of our courseware that incorporates the interactivity provided by the *I'll Try* mode will perform equally well as those students who use the version that omits it.

We have discussed a variety of different learning style models that have been applied in educational research and have reviewed numerous studies that have attempted to correlate differences in styles with performance on CAI and with computer skills—programming, in particular. We have conducted an investigation similar to some of those that we have reviewed. Next, we state our second null hypothesis:

$H_{02}$: The benefit of incorporating interactivity into the courseware will be no different for students whose learning style is active compared to those students who are reflective.

Our final null hypothesis is:

$H_{03}$: There will be no significant interaction between the presence of interactivity and the type of learning style.

### 4.1.2 Experimental design of the first study

Having formally stated both of our hypotheses, let us now describe the design of our first study. In our study we used a true experimental design—specifically the posttest-only control group design.

The subjects of our first study were students enrolled in CSci 131, *Data Structures*, at The George Washington University during the spring 1998 semester. Our experiment included two independent variables: (1) the instructional method, either with interactivity or without and (2) the learning style of the students, either active or reflective. The only dependent variable was the scores of students on the posttests.

The instrument that we used to measure learning styles was the Felder-Silverman learning style inventory. This instrument produces a learning style that has four dimensions: (1) sensing or intuitive learners, (2) visual or verbal learners, (3) active or reflective learners, and (4) sequential or global learners. We chose this learning style model because the first three dimensions are all of interest. Previous studies have demonstrated a difference between the benefit of algorithm animation among sensors compared to intuitors. The second dimension dividing learning styles into visual and verbal learners is pertinent because of the highly visual nature of our courseware. Finally, the third dimension is the focus of our second hypotheses, which is that, active learners, those who learn by trying things out, will benefit more from the interactivity than the reflective learners. In her study of the relationship between learning styles and multimedia, Montgomery

[MON95] chose Solomon's Inventory of Learning Styles, which measures the same four dimensions as the Felder-Silverman inventory, rather than Kolb's or Myer's-Briggs. Her reason for rejecting the Kolb inventory was that its questions contained excessive jargon and were hard to answer. She rejected the MBTI because it was focused too heavily on personality.

The other instrument that we used was a posttest that was developed by the researcher and validated by the course director for CSci 131. The posttest consisted of several questions on the final examination for the course.

In many of the previous studies on algorithm animation a variety of different kinds of questions were used in the posttests. Stasko, Badre and Lewis [STA93] divided questions into declarative and analytical categories. Lawrence, Badre and Stasko [LAW94] used a different division—free-response and factual questions. Crosby and Stelovsky [CRO95] divided their questions into textual and graphical groups. No two studies used the same division. We chose no such division. The intent of our courseware is to help students either understand the steps of an algorithm, as with the heap sort lesson, or to understand a particular data structure definition, such as the definition of an almost complete binary tree in the binary tree categorization lesson. Our posttests measured precisely the skill that the courseware was designed to convey—can students reproduce the steps of a heap sort given a new heap, for example.

Clearly procedural knowledge as reproducing the steps of an algorithm is only the first level necessary for understanding an algorithm. At this level, students understand how the algorithm works, but do not necessarily have the deeper conceptual understanding of why it is implemented as it is. It may be interesting to determine whether watching an animation better enables students to answer questions about the efficiency of an algorithm, but using such questions introduces potential confounding factors. Consequently, we have excluded them from our posttests.

In our first study, 33 students used the courseware in a closed laboratory setting, which was the recitation session of the CSci 131 course. The recitation session is normally used for work on programming projects. During the last four sessions that semester, students were expected to complete several of the lessons of the courseware. Our schedule describing which lessons were used during which weeks is shown in Table 4.1.

Our procedure was to randomly divide the students into two groups. The two treatments—interactive or noninteractive—were rotated among the two groups so all students received one treatment for two of the weeks and the other treatment for the other two. Our schedule for rotation is shown in Table 4.2.

Because our courseware was hosted on the World Wide Web, we needed to limit access to it during the period of the study. We used password protection during that time. When students accessed the courseware, they were asked to provide their name and password. Depending upon the student's group and the week of the study, the student was automatically given the interactive

or noninteractive version, as appropriate. Students who received the interactive version of the courseware were expected to provide twenty correct answers for each operation of each lesson in the *I'll Try* mode, before they could submit a completed lesson. Students who received the noninteractive version were expected to watch twenty correct actions in the *Show Me* mode.

| | |
|---|---|
| Week 11 | Graph Representations<br>Graph Searches<br>Categorizing Graphs |
| Week 12 | Binary Tree Traversals<br>Binary Search Trees |
| Week 13 | Height-Balanced Trees<br>Priority Queue with Heap<br>Categorizing Binary Trees |
| Week 14 | Heap Sort<br>Quadratic Sorts<br>Efficient Sorts |

Table 4.1 Schedule for use of courseware lessons

| | Group 1 | Group 2 |
|---|---|---|
| Week 11 | Noninteractive | Interactive |
| Week 12 | Interactive | Noninteractive |
| Week 13 | Noninteractive | Interactive |
| Week 14 | Interactive | Noninteractive |

Table 4.2 Rotation of treatments among groups

### 4.1.3 Problems in the initial experimental design

After the first run of the experiment was completed, the results were analyzed. On average, students performed better on the problems for which they did not have the *I'll Try* mode, although the difference was not statistically significant. This result was somewhat surprising. In Chapter 5, we elaborate on these results.

We considered the possibility that these results were due to several possible problems in the design of the first experiment.

(1) Using the final examination as a posttest may have been a poor measurement of the learning that occurred while using the software. Students had too many other opportunities to learn the same material from other sources, such as the lectures and the textbook. Consequently, it is difficult to know to what extent the software influenced their learning.

(2) Running the experiment over such an extended period introduced major problems in the consistency of the data. There was one student who completed the learning style questionnaire, hence was entered into one of the two random groups, but never completed any of the visualizations. There was one student who completed all the visualizations, but he did not take the final exam. There were many students who only completed some of the visualizations. These inconsistencies complicate the analysis. Of the 33 students who participated in the experiment in some form, only 24 completed all components of the study.

(3) Asking students to use the software until they showed or tried a specific number of tasks was not an ideal objective to have given. Had we instead instructed students to use the software until they felt they understood the topic, the amount of time they used it would have been a more meaningful statistic.

(4) Interleaving the *I'll Try* and no *I'll Try* modes had the undesirable side effect that after having used the *I'll Try* mode, some students would imitate it when it was not present. This effect was observed during the think-aloud protocol, which we discuss in Chapter 5.

(5) Each recitation section contained students from both groups. As a result, a student in one group might be sitting next a student in the other group. This often raised concerns about the fact that one student had a different version from his neighbor.

There were other problems that resulted from this experimental design. Some students were reticent to participate believing their performance would influence their grade. Restricting access to the software, necessary to ensure experimental control, was perceived as unfair by many students. They believed that they should be given access to the software, especially to study for the final exam.

### 4.1.4 Experimental design of the second study

The experiment design of the second study was modified in several ways to address our concerns in the design of the first experiment. The subjects of our second study were 52 students enrolled in CSci 131 during the fall 1998 semester. The recitation laboratory had been expanded from one hour to two hours. The longer sessions made it possible to condense the entire study into three weeks. Previously, the consent form and learning style questionnaire had been administered a week before the study. The study itself was four weeks, and the final examination was given two weeks after the last laboratory. The entire study spanned a seven week period, increasing the possibility of student absences. In the second study, the consent form and learning style questionnaire were administered the first week. The consent form was put on the Web as a

precursor to the learning style questionnaire to ensure its review by all participants. The three graph lessons were also included in the first week. The second week all five binary tree lessons were completed. The third week students studied the three sorting lessons and completed the courseware evaluation. The longer laboratories also made it possible to add a quiz at the end of the laboratory each week. These quizzes were used as the posttests instead of the final examination as had been done previously.

The second major change to the experimental design was that students were not rotated. Instead the four recitation sections were divided into two groups of two sections. This division was done based on the mid-term examination scores. Each of the three possible pairs was considered. A comparison of the three pairs is shown in Table 4.3, which shows the average, range and standard deviation of the mid-term examination grades. In addition, the size of the groups is shown and the teaching assistants that correspond to each section. The first pairing was chosen because it was the one that provided the two groups whose performance on the mid-term examination was most comparable. Because one of the three sections was much larger than the other three, no pairing would have provided two groups of equal numbers. Because several students never attended the laboratory during the three weeks of the study, the mid-term examination averages for the final two groups differ slightly from the values in this table. This change eliminated the possible transfer effect of rotation that we described previously and it eliminated the possibility that a student would be seated next to another student who was using a different version of the software.

| | Average | Range | Standard Deviation | Count | Teaching Assistants |
|---|---|---|---|---|---|
| Section 30 & 32 | 43.8 | 19 - 62 | 11.3 | 34 | Same |
| Section 31 & 33 | 43.6 | 19 - 60 | 12.8 | 21 | Same |
| Section 30 & 31 | 44.2 | 19 - 60 | 11.1 | 23 | Different |
| Section 32 & 33 | 43.3 | 19 - 62 | 11.5 | 32 | Different |
| Section 30 & 33 | 42.1 | 23 - 55 | 9.3 | 24 | Different |
| Section 31 & 32 | 44.9 | 19 - 62 | 12.5 | 31 | Different |

Table 4.3 Possible pairing of recitation sections

The final modification was changing the objective of each laboratory. In the first study, the objective was to use the courseware until they completed twenty correct responses for those with the interactive version or until they watched twenty actions for those with the noninteractive

version. In the second study, students were instructed to use the courseware until they felt they understood the material, at which time they were given the quiz. This last change enabled us to add a new hypothesis to our investigation based on a second dependent variable—the amount of time that students used the courseware.

$H_{14}$: The students who used the interactive version of the courseware would use it significantly longer than those who used the noninteractive version.

### 4.1.5 Experimental validity

We have designed our experiments to try to minimize confounding factors. As previously noted, by comparing two versions of Web-based courseware rather than comparing courseware to classroom instruction, we have eliminated many potential threats to the validity of our results. In the first experiment we intentionally rotated the control group with the treatment group each week. Such a rotation had two benefits. It eliminated the confounding influence that by chance we divided the students into groups where one group performed significantly better than the other, which did occur in the first experiment. In addition, because we used students in a real classroom setting, we needed to be sensitive to fair treatment of all students. If the treatment had proven beneficial in the first experiment, the students that received the treatment might have performed better on the final examination and consequently improved their course grade. The rotation ensured that any benefit provided by the treatment was distributed equally among all students.

Although this rotation provided certain benefits, it introduced another potential threat, which was the transfer effect of having used the other version of the courseware. We observed this effect during the think-aloud protocol. That student, having used the interactive version one week, imitated the *I'll Try* mode the next week, while using the noninteractive version. In the second experiment we eliminated the rotation. To avoid the possibility of two groups of unequal ability, we selected the groups based on their mid-term examination performance. The possible threat with this approach is that the mid-term examination may not have been a good predictor of student performance on the courseware.

There were several other potential threats to the validity of our experiment. First, if we had decided to allow students to use the courseware outside the supervised laboratory, then it would have been possible that someone other than the actual student would do the work. By conducting both experiments in a supervised laboratory, we minimized this possibility.

The next threat was that students were not required to complete laboratory assignments. Because we were conducting our experiment in a real classroom environment, we encouraged students to attend the recitation session and complete the assignments. A small part of the final grade included their participation. This issue proved to be a problem in the first experiment, since a significant number of students did not complete all the assignments. In our redesigned second

experiment, we minimized this problem by conducting the experiment in a shortened period of three weeks.

Another confounding factor was that students were able to learn the material contained in the lessons from other sources—classroom lectures and the textbook. This proved to be a problem in the first experiment, because we used the final examination as the posttest. We minimized this problem in the second experiment by requiring students to take the posttest directly after they used the courseware.

One other threat to the validity of both of our experiments remained. As previously noted, we chose the posttest-only experimental design to decrease students' focus on their participation in an experiment. Requiring students to complete a pretest as part of a laboratory assignment is not customary. By including it before each use of the courseware, students would have been reminded that they were participating in an experiment. Such awareness can lead to a variety of confounding factors, such as the Hawthorne effect. Omitting the pretest did, however, eliminate one threat to internal validity—mortality—the effect of losing subjects between the pretest and posttest.

## 4.2 Secondary research questions

Our primary research question concerns whether interactivity and learning styles affect students' learning. Several secondary questions, questions of a more subjective nature, are a natural complement to this research. The first of these questions is whether courseware of this kind is a useful component of an elementary data structures course. To be effective, courseware must be used. Students have many demands upon their time and some will avoid activities they perceive as not directly related to their goal—a good grade in the course they are taking. In addition, courseware will have limited effectiveness if it is not easy to use. Because students themselves are one of the best judges of the effectiveness of various educational tools or techniques, we expected to be able to answer this question from their comments.

Although the educational use of algorithm animation is not new, Web-based implementations are relatively new, so another of our secondary research questions is how well the Web would perform as a platform for algorithm animation courseware. Being hosted on the Web, the boundaries of the laboratory are no longer necessarily confined to the physical boundaries of the university laboratory, but they can extend to students' own computers, without even explicitly providing students with a copy of the courseware. The potential benefits of the Web can only be realized if the Web is stable enough, so that such courseware can be reliably used from different types of computers and Web browsers. It was our expectation to be able to answer this question, and as a result, conclude how such courseware might be best integrated into an introductory data structures course.

The final of the secondary questions is an attempt to gain some insight into how animations can be made most understandable. Algorithm animation courseware can only be effective if the

animations are easily understood. The basic features that should be animated in standard algorithms, such as sorts, are well understood. What is not well understood is what visual cues are needed to enhance basic animations to make them more understandable. Through feedback from our student users, we hoped to be able to provide some preliminary answers to this question.

### 4.2.1 Research plan

Unlike the primary questions, no specific experiments were designed to answer the secondary questions. While our primary questions were addressed in successive experiments in two semesters in 1998, the data collected to answer the secondary questions spans a two year period beginning with the spring 1997 semester. Additional data was collected to answer these secondary questions. Each of the four semesters the level of participation was measured and feedback was gathered from students. Given the feedback, appropriate changes were made from semester to semester. During the last two semesters, log file data was collected. During the spring 1998 semester, selected students were recorded thinking-aloud while using the courseware.

We used the courseware for four semesters, in three different formats. In the spring of 1997, the courseware was provided to students as an optional learning tool. Students were given a small amount of extra credit for using the tool and evaluating it. The most important outcome of this first trial was the recommendation from the students that it should be modified to be on-line homework exercises. The presence of the *I'll Try* feature made this transformation a relatively simple one.

The second trial was during the spring 1997 semester. This trial was intended as the pilot study. During this semester, the courseware was used as on-line homework assignments as had been suggested by the students the previous semester. The primary purpose of the pilot study was to validate the usability of our courseware. Students during the previous semester had made limited use of the courseware, so additional testing was needed. Heller [HEL91] has noted the importance of evaluating multimedia courseware and the variety of different methods that have been used. The method we used was to give evaluations to our students to complete again. The evaluation was similar to the one used during the previous semester. Because the students used the courseware during a four week period and spent several hours using it, they were able to make a more informed assessment of its usability than the students who used it during the previous semester. In particular, several additional questions were included. These questions attempted to measure whether students understood the intended meaning of various features of the animation, such as color changes and check marks. The questions included a series of statements that students were asked to rate on a Likert scale. In addition, several open-ended questions were included that attempted to elicit any other comments regarding changes that students felt needed to be made to the courseware.

The third trial of the courseware was during the spring 1998 semester. This trial was the first experimental study. For that reason, we needed to maintain a more controlled environment than we had the previous semesters, so students were asked to use the courseware as laboratory exercises rather than on-line homework. Although the courseware was on the Web, access was disabled during times other than scheduled recitation sessions. During this semester the researcher was the teaching assistant in the laboratory. In previous semesters, he had been the course instructor. Being present in the laboratory while the students used the courseware provided a deeper insight into the usability of the courseware than the evaluations of the previous two semesters because the feedback was more immediate. Also, two students were recorded thinking aloud while using the courseware to better determine how easy the animations were to understand. As we had done during the previous two semesters, students were asked to complete evaluations after they had used the courseware.

The fourth trial was during the fall 1998 semester. This trial was the second experimental study. As had been done the previous semester, access was restricted during the three week study. Students were only permitted to use the courseware in the scheduled recitation sessions. Unlike the previous semester, the courseware was made freely available to students between the final week of the study and the final examination. Unlike previous semesters, the researcher was neither the instructor nor teaching assistant for the class, but was present for half of the recitation sessions during the period of the study. Again, students were asked to evaluate the courseware during the final week of the study.

Our varied experiences using our courseware have enabled us to provide some assessment of how such courseware can be integrated into an introductory data structures course and the advantages of each approach. We will elaborate on these conclusions in Chapter 6.

### 4.2.2 Data collection

To determine students' perception of the value of the courseware, its ease of use, how it was used, the effectiveness of the Web, and how understandable the animations were, we collected several kinds of supplemental data.

First, as previously mentioned, at the end of each semester, students were asked to complete evaluations of the courseware. These evaluation questionnaires, like the courseware, were Web-based. The questionnaires that were used are in Appendix B. They included two types of questions providing both quantitative and qualitative data for analysis. The first questions were ones that could be answered by choosing from a Likert scale. The topics of these questions included the effectiveness of the courseware as a learning tool, its ease of use, the effectiveness of the Web as a platform, among others. In addition, each evaluation included some open-ended questions, in which students were asked to provide written comments about their experience using the courseware.

Second, we collected detailed information about the depth of use. In the fall 1997 questionnaire, we included questions asking how frequently they had used some of the different controls. The responses suggested that, in some cases, students did not remember clearly. Rather than rely on their recollection, we decided to collect detailed information that recorded how the courseware was used. Nebesh [NEB97] successfully used a similar technique to track the navigation of Web pages. In the spring 1998 version, we included a logging feature that logged and time stamped each button click. This approach provided an overwhelming amount of data that required condensation before it could be interpreted, so in the final fall 1998 version, rather than logging each button click, we counted the button clicks, and logged a summary. This data provided a clear picture of how deeply the courseware was used. It also highlighted specific controls that were rarely used, suggesting a need for a better introduction on how to use the courseware. We should also note that the log file data provided the measurement of time the courseware was used—a second dependent variable in our second experiment.

Finally, we collected data to determine whether the animations were understandable. In the fall 1997 questionnaire, we included questions asking students whether specific aspects of the certain animations were understandable. Because students were completing these questionnaires a week or more after using the courseware, in most cases the students did not remember the aspect of the animation in question. In the spring 1998 semester, we used a more effective method to answer this question—recording students using the courseware, thinking aloud as they used it. Kehoe and Stasko [KEH96] successfully used a similar technique—observing students while they used algorithm animations.

# CHAPTER 5

# ANALYSIS OF RESULTS

Our research consists of the two experimental studies designed to answer the primary questions of whether interactivity provides a significant learning advantage and whether students' learning styles influence that benefit. In addition, we collected supplemental data to help us answer several secondary questions regarding the benefit of such courseware, the role of the Web as a platform, and the question of how to best make animations understandable. First we analyze the results of the two experiments, then we discuss the implications of the supplemental data.

## 5.1 Experimental results—first study

Our first experiment was conducted during the spring 1998 semester. The experiment spanned a seven week period at the end of the semester. The learning style questionnaire was given the first week, the students used the courseware for the next four weeks and they were given the final examination, which was the posttest, in the seventh week. A total of 33 students participated in the study, initially.

### 5.1.1 The learning style data

The Felder-Silverman learning style questionnaire that we used assesses four dimensions of a student's learning style. Recall that these dimensions are processing, which is either active or reflective, perception, which is either sensing or intuitive, input, which is either visual or verbal, and understanding, which is either sequential or global. Table 5.1 contains the results of the learning style questionnaire.

| | | |
|---|---|---|
| Processing | 18 Active | 15 Reflective |
| Perception | 16 Sensing | 17 Intuitive |
| Input | 29 Visual | 4 Verbal |
| Understanding | 13 Sequential | 20 Global |

Table 5.1 First experiment—learning style summary

The processing dimension favored active learners, the perception dimension split rather evenly, input strongly favored the visual mode of learning and the understanding dimension strongly favored global understanding. These results are consistent with Felder's observations, with the exception of the understanding dimension.

### 5.1.2 The posttest data

Of the 33 students who began the study, all but one student took the final examination. We used three of the problems on the final examination as our posttest. These three problems are in Appendix D.1. Each problem had two parts. The first problem tested the material in two of the graph lessons in the courseware—categorizing graphs and graph searches. The second problem tested material in two of the binary tree lessons—tree traversals and categorizing trees, and the last one tested two of the sorting lessons—the selection sort and the quick sort. Recall that the students were randomly divided into two groups. The results of the individual posttest problems by group are shown in Table 5.2. The shaded cells indicate that the group had the interactive version of the courseware for the lesson pertaining to that problem.

| | Group 1 average | Group 2 average | Overall average |
|---|---|---|---|
| Graph categories | 74.51 | 93.33 | 83.33 |
| Graph searches | 48.24 | 66.67 | 56.88 |
| Tree traversals | 79.78 | 84.58 | 82.03 |
| Tree categories | 68.24 | 78.67 | 73.13 |
| Selection sort | 83.53 | 80.67 | 82.19 |
| Quick sort | 47.65 | 56.00 | 51.56 |
| Overall average | 66.99 | 76.65 | 71.52 |

Table 5.2 First experiment—posttest results by group

The results indicate that the quick sort was the most difficult and the graph search problem was the second most difficult. These results were expected. What was not expected was the large—almost ten point—difference in performance between the two groups. Although we randomly formed these groups, by chance, one group performed much better than the other. This occurrence underscores why it was important to give both treatments to both groups. In this way we are able to factor out the group differences. The students in group 1 had the *I'll Try* mode for both graph lessons and the tree category lesson and did not have it for the other three lessons. The arrangement was reversed for the students in group 2. In Table 5.3 we compare the performance based on the version of the courseware that was used. The results are the opposite of what we would have expected. Both groups performed better on the lessons that did not incorporate the *I'll Try* feature. The difference is not, however, statistically significant using a paired two tailed t-test ($t(31) = 1.60$, $p = 0.119$). We should note that the above data includes all 32 students who both completed the learning style questionnaire and took the final examination. Of these 32 students,

one did not use the courseware at all and another nine did not use it for some of the weeks. We examined the same data, eliminating these ten students. It is not appreciably different. Given these results, we did not examine the possible influence of the learning styles.

| | Group 1 average | Group 2 average | Overall average |
|---|---|---|---|
| *I'll Try* lessons | 63.7 | 73.8 | 68.4 |
| *Show Me* lessons | 70.3 | 76.7 | 73.3 |
| Difference | -6.7 | -2.9 | -4.9 |

Table 5.3 First experiment—posttest results comparing *I'll Try* and *Show Me*

In our earlier discussion of our research plan, we described several problems in our experimental design that may have contributed to this result. The most important of these was that the students' performance on the final examination may have been influenced by what they learned from other sources—the classroom lectures and the textbook. Consequently, we found it necessary to perform a second redesigned experiment, in which the posttest was givenimmediately following the use of the courseware.

## 5.2 Experimental results—second study

Our second experiment was conducted during the fall 1998 semester. The experiment consisted of three laboratory sessions that spanned a four week period at the end of the semester. In the second experiment, the posttest was a set of quizzes given at the end of each of the three sessions. Also, rather than rotating groups, one group had the interactive version for the whole study—the other group did not. We ensured the two groups had comparable ability based on their mid-term examination performance. A total of 52 students participated in the second experiment—a larger sample than we had for the first experiment.

### 5.2.1 The benefits of interactivity

The results of the second experiment were the same as the results of the first experiment. The group that used the interactive version of the courseware performed marginally, but not significantly, worse than the group that used the noninteractive version. Finding the same result in both experiments increases our confidence that this result is correct and not the result of poor experimental design as we speculated following the first experiment. A summary of the results of the posttests is shown in Table 5.4. The noninteractive group performed better on each of the three quizzes, although in no case significantly so, using a two-tailed t-test. Although the noninteractive group scored better on all three quizzes as a whole, the interactive group did perform better on

three of the eleven individual questions, the graph search, binary tree traversals and the heap sort. These three are among the most difficult of the eleven lessons. The three lessons that do not involve any animation, the graph categories, tree categories and height balancing, are among those that the interactivity appeared to have the benefited the least. This data suggests that interactivity may help most on the more difficult questions and the ones that involve animation as opposed to just visualization. As the data in the table indicates, the differences are not statistically significant on any of the eleven questions. The questions that correspond to each of the eleven lessons are in Appendix D.2.

| Quiz | Lesson | Noninteract | Interactive | Difference | df | t | p |
|---|---|---|---|---|---|---|---|
| Q1-1 | Graph categories | 91.95 | 81.67 | 10.29 | 47 | 1.62 | 0.11 |
| Q1-2 | Graph searches | 62.07 | 63.75 | -1.68 | 47 | -0.20 | 0.84 |
| Q1-3 | Graph representation | 94.25 | 86.67 | 7.59 | 47 | 1.49 | 0.14 |
| Q2-1 | Tree traversals | 33.87 | 39.47 | -5.60 | 48 | -0.50 | 0.62 |
| Q2-2 | Binary search trees | 82.26 | 78.95 | 3.31 | 48 | 0.40 | 0.69 |
| Q2-3 | Priority queue | 70.97 | 47.37 | 23.60 | 48 | 1.68 | 0.10 |
| Q2-4 | Height balancing | 43.55 | 42.11 | 1.44 | 48 | 0.14 | 0.89 |
| Q2-5 | Tree categories | 78.49 | 68.42 | 10.07 | 48 | 1.31 | 0.20 |
| Q3-1 | Selection sort | 66.67 | 47.37 | 19.30 | 44 | 1.53 | 0.13 |
| Q3-2 | Quick sort | 52.78 | 50.00 | 2.78 | 44 | 0.22 | 0.83 |
| Q3-3 | Heap sort | 32.10 | 40.35 | -8.25 | 44 | -0.64 | 0.53 |
| Quiz 1 | Graphs | 80.34 | 77.00 | 3.34 | 47 | 0.74 | 0.46 |
| Quiz 2 | Trees | 62.90 | 57.37 | 5.53 | 48 | 1.00 | 0.32 |
| Quiz 3 | Sorts | 49.26 | 46.32 | 2.94 | 44 | 0.33 | 0.74 |
| | Average | 64.64 | 60.42 | 4.22 | 50 | 0.91 | 0.37 |

Table 5.4 Second experiment—posttest results by group

We also examined the final examination results for the fall semester. Students were permitted to use the courseware between the end of the experiment and the final examination. All students were permitted to use the interactive version during that time. The group that had used the interactive version during the experiment scored better on the first three problems of the examination that pertained to the courseware lessons than the group that had used the noninteractive version, and about the same on the remaining three problems. The differences were not, however, statistically significant.

Our second experiment was also designed in such a way that the amount of time students spent using the courseware could be a meaningful second dependent variable. In the second

experiment, both groups were instructed to use the courseware as long as they felt necessary to understand the lessons each week. The amount of time that students used the courseware was measured using the log file data. Although students were required to enter their names when they began using the courseware, a small percentage of the log file records did not contain identifiable student names. Because all records were time-stamped each record could be identified by group. The average times for each group are shown in Table 5.5. Also included in the table are the numbers of examples each group watched in the *Show Me* mode.

| Group | Average time used | Average watched |
|---|---|---|
| Noninteractive group | 1 hour, 2 minutes, 28.5 seconds | 619.44 |
| Interactive group | 1 hour, 19 minutes, 52.5 seconds | 862.20 |

Table 5.5 Average time used—second experiment

The average for the interactive group was approximately 30% higher than for the noninteractive group. Distributing the unnamed records equally among students in the appropriate group, the result is statically significant using a one-tailed t-test ($t(50) = 1.87$, $p = 0.034$). Even with a highly uneven distribution, assigning all unnamed records to a single student, the probability value is within the range of significance.

A second measure of increased use by the interactive group was that they watched more examples in the *Show Me* mode than the noninteractive group. The difference is again statistically significant using a one-tailed t-test ($t(50) = 2.28$, $p = 0.030$).

### 5.2.2 The impact of learning styles

As had been done in the first experiment, the Felder-Silverman learning style questionnaire was again used to assess student learning styles. Table 5.6 contains the results of the learning style questionnaire.

| Processing | 31 Active | 21 Reflective |
|---|---|---|
| Perception | 32 Sensing | 20 Intuitive |
| Input | 44 Visual | 8 Verbal |
| Understanding | 31 Sequential | 21 Global |

Table 5.6 Second experiment—learning style summary

Comparing the survey results for this semester with the previous, the results were comparable in the processing and input dimensions—more active and more visual students both

semesters. The other two dimensions differed somewhat from the prior semester to the current. Our prime concern, however, is to determine whether there were any significant differences in the performance on the posttests between students of different learning styles. Although the processing dimension is of prime interest, we examined all four dimensions. The results are shown in Table 5.7.

| Dimension | | Average | | Average | df | t | p |
|---|---|---|---|---|---|---|---|
| Processing | Active | 6.52 | Reflective | 5.98 | 50 | 1.16 | 0.25 |
| Perception | Sensing | 6.14 | Intuitive | 6.57 | 50 | 0.93 | 0.36 |
| Input | Visual | 6.41 | Verbal | 5.71 | 50 | 1.12 | 0.27 |
| Understanding | Global | 6.37 | Sequential | 6.25 | 50 | 0.26 | 0.80 |

Table 5.7 Second experiment—learning style posttest results

In no case is any of the differences significant. In each case, a two-tailed t-test was performed. The processing and input dimensions have the lowest probabilities.

In the case of the processing dimension, the active students performed better than the reflective students. We also examined each of the eleven questions individually. On the graph search question, the active students performed significantly better than the reflective students using a two-tailed t-test ($t(47) = 2.86$, $p = 0.010$). This was also one of the three questions on which the interactive group scored better than the noninteractive group.

As it had happened the previous semester, the visual students performed better than the verbal students. Given the visual nature of the courseware, this result is not unexpected. On one of the eleven questions—the one concerning binary search trees—the difference was statistically significant, using a two-tailed t-test ($t(48) = 2.08$, $p = 0.043$).

### 5.2.3 The interaction of interactivity and learning styles

Although we found no significant differences between the interactive and noninteractive groups and no significant differences between active and reflective learners, it would still be possible for a significant interaction to exist between the two factors. Before presenting the numerical results of the two factor-two group per factor-ANOVA, we consider the graphical depiction of the interaction as shown in Figure 5.1. The orientation of the lines of the graph does deviate from parallel, suggesting a possible interaction. The results indicate that active learners were the ones hurt by the interactivity, while the reflective learners were helped slightly. In our post hoc analysis, we offer a possible interpretation of this result. Our prime concern now is whether this interaction is statistically significant.

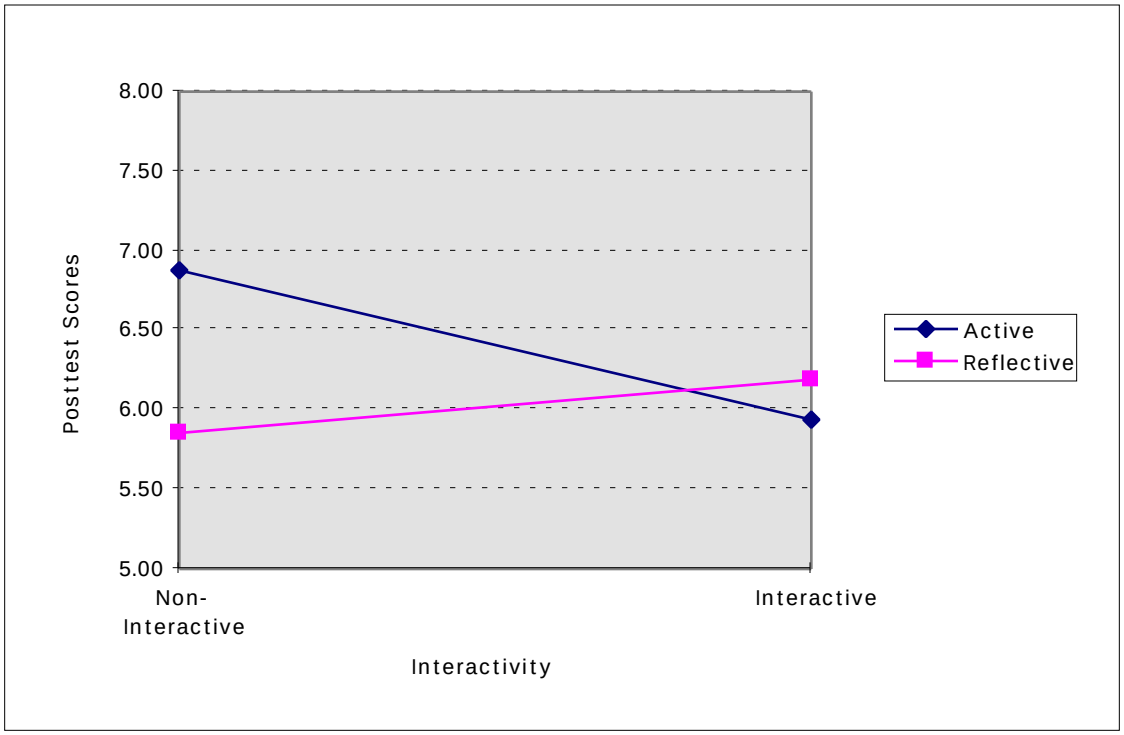



Figure 5.1 Graph of interaction between interactivity and learning styles

Table 5.8 contains the numerical results of the two factor ANOVA with independent measures on both factors. The table indicates that the interaction between factors is not statistically significant. The table also reiterates the nonsignificance of each of the individual factors, which we have already noted by the prior t-tests.

| Source of variation | Degrees of Freedom | Mean Square | Variance ratio (F) | Probability > F |
|---|---|---|---|---|
| | | | | |
| Active / Reflective | 1 | 3.543 | 1.37 | 0.2480 |
| Interactive / Noninteractive | 1 | 2.223 | 0.86 | 0.3589 |
| Interaction | 1 | 4.711 | 1.82 | 0.1838 |

Table 5.8 Interaction between interactivity and learning styles—numerical results

### 5.2.4 Post hoc analysis

Let us now attempt to explain the meaning of our results. The result that requires further interpretation is that interactivity significantly engages student interest as measured by the time spent using the courseware, but this added time does not translate to improved understanding as

measured by the posttest results. In fact, both our studies indicate poorer results for the interactive group, although not significantly so. The inclusion of interactivity appears to encourage ineffective use of time. The explanation we offer to explain this result is that some students may be encouraged to treat the courseware as a video game when interactivity is present and focus on being entertained but not on learning. Although further studies would be needed to confirm this claim, we may be able to find some supporting evidence for it by performing some post hoc analyses of our data.

To try to gain a better insight into which students might be hurt by interactivity, first we consider differences between high achieving and low achieving students as measured by their performance on the mid-term examination—the measure we used for forming groups of equal ability. The high scorers are those whose score was above the median. The low scorers are those whose score was below the median. Figure 5.2 illustrates the relationship between the performance on posttests and the courseware version for these two groups.

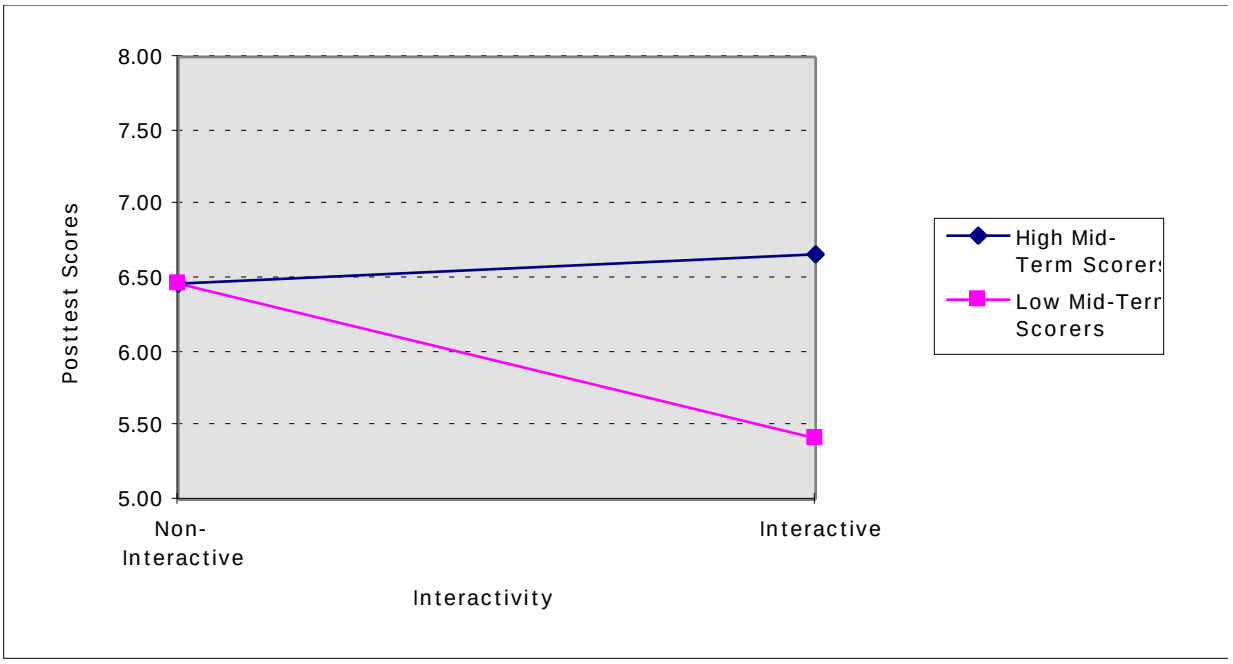



Figure 5.2 Interaction between interactivity and mid-term scores

The poorer students were hurt by the interactivity while the better students were helped. The inference that might be drawn from this relationship is that students whose understanding of the material is poorer might have a greater tendency to treat the courseware as a video game when interactivity is present. Note that the difference between these two groups is not statistically significant, nor do we make any claim to statistical significance of the interaction. Our examination of this data is to try to gain some insight into which students are hurt most by the interactivity.

Next, let us refer back to Figure 5.1, which illustrates the interaction between the processing learning style dimension and the presence or absence of interactivity. The active students were hurt by the interactivity and the reflective students were helped slightly. Our suggestion is that the active students would be more likely to treat the courseware as a game, while the reflective students, those more likely to be contemplative, would not rush to use the interactivity until they understood the material. Again, as we stated in the previous section, these results lack statistical significance, but may offer hints, nonetheless, into the reasons for our overall finding.

Next we restrict our attention to only those students who used the interactive version of the courseware to determine whether there is any relationship between their performance and the degree to which they used the *I'll Try* mode. Students who were given the interactive version had the option to use *I'll Try* mode as little or as much as they chose. Their instruction was to use the courseware until they felt they understood the material. All students who were given the interactive version were made aware of the capability to ensure none would inadvertently overlook it. For each of these students, we computed an interactivity ratio, which was the sum of correct or incorrect responses to the number of watched responses. The higher the ratios the more each used the *I'll Try* mode compared to the *Show Me* mode. This data was extracted from the log file data. A scatter plot of this data is illustrated in Figure 5.3.

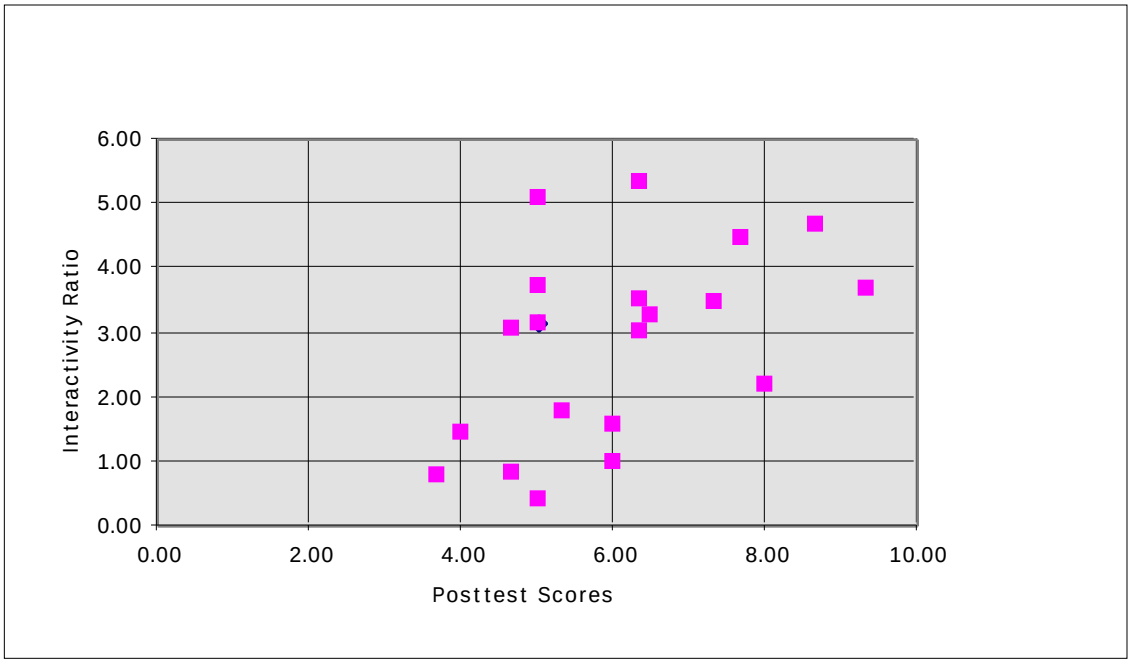



Figure 5.3 Scatter plot of interactivity ratio and posttest performance

Numerically the correlation between the interactivity ratio and posttest scores is quite low, but if we exclude some of the outliers, the scatter plot does seem to suggest that those with the higher ratios did perform better.

Finally, let us again consider only the students who used the interactive version to see whether we can find any evidence that interactivity is helpful to those students who understand the material—those who are not using the interactivity as a guessing game. We use the log file data to divide the students into two groups—guessers and nonguessers. We computed a different ratio to form this division—the ratio of correct to incorrect responses. Those whose ratio was above the median were deemed nonguessers—those below were guessers. As we would expect, there was a statistically significant difference between the posttest performance of the guessers compared to the nonguessers on a one-tailed t-test ($t(18) = 2.14$, $p = 0.023$). This evidence serves to validate our posttest. We now compare the interactivity ratio discussed earlier with posttest performance for the guessers and nonguessers. Figure 5.4 illustrates this relationship.

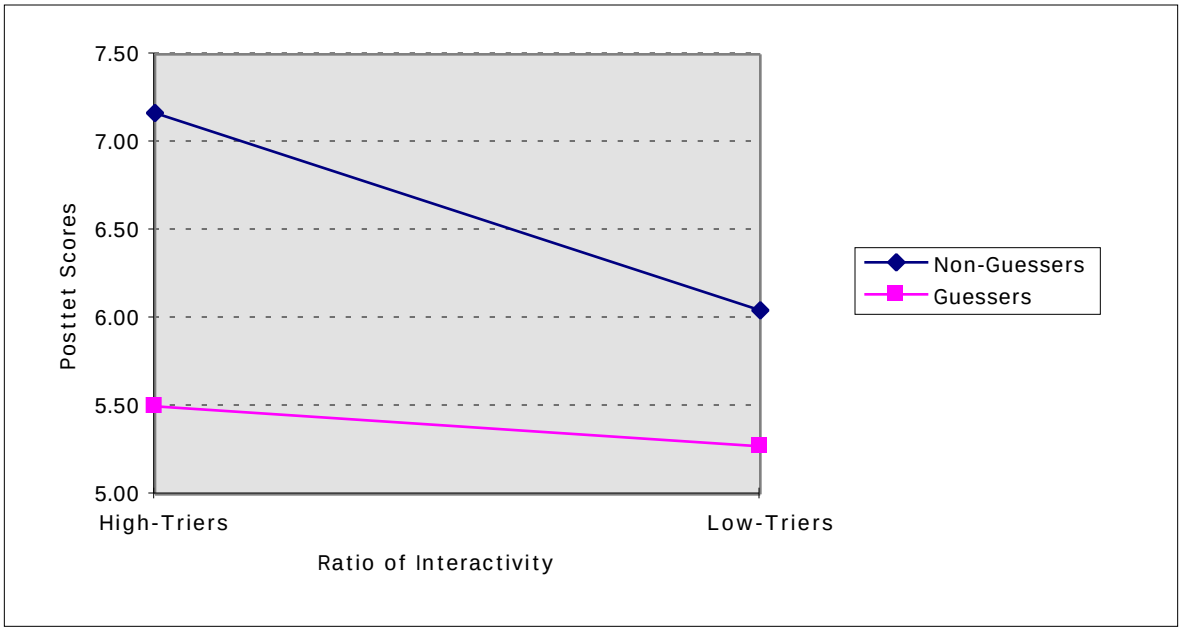



Figure 5.4 Interaction between interactivity ratio and guessing

This graph suggests that using the *I'll Try* mode more, a high interactivity ratio, helped those who used it to confirm their understanding—those who were not just guessing, however increased use of the *I'll* Try mode did little to improve the performance of those who were just guessing.

To summarize what our post hoc analysis suggests is two issues. First, certain students, when given the interactive courseware, may be more likely to treat it as a video game. Our conjecture is that these students are the weaker students and the active students. Our second conjecture is that interactivity may be helpful when it is properly used—to confirm understanding—and not used as a guessing game.

## 5.3 Analysis of evaluations by students

There were three major types of ancillary data that we collected during our research to answer our secondary research questions and to provide additional insight to our primary questions. The first type of ancillary data we collected was the evaluations by students that were given at the end of each semester. The second type of ancillary data was the logging data that provided detailed information regarding the extent to which students used the courseware. The final type of ancillary data was the recording of a student thinking aloud while using the courseware.

Students were asked to complete evaluations after using the courseware for four semesters: spring 1997, fall 1997, spring 1998 and fall 1998. Although the evaluation questionnaires differed somewhat, from semester to semester, there were some questions that were used for all four semesters. Each semester, the evaluations were Web-based forms. The evaluation questionnaires used during each of the semesters are in Appendix B.

### 5.3.1 Analysis of evaluations for spring 1997

The first semester, spring 1997, students were asked to use the courseware independently, so among the information gathered was information concerning which platform they used, what browser they used and what type of connection to the Internet they had. Finally they were asked how frequently they used the Web. Fifteen students completed the questionnaire, which represented about a half of the students who completed the course. All the students used Netscape Navigator, six students used a Macintosh system, one used a UNIX workstation and the rest used a PC. All but one student used the Web at least weekly.

The major goal of this first evaluation was to determine the feasibility of using such Web-based courseware and to uncover any technical, logistic, or pragmatic problems and to ascertain whether there were any modifications that needed to be made.

The major technical problem revealed by this evaluation was the issue of speed. There were two aspects of this problem—the time required to download the courseware and the slow response time. The first aspect was a greater concern for students accessing the courseware remotely. The later aspect was more pronounced on low-end machines, but surprising was most affected not by the power of the machine, but by the browser that was used. Although all students who completed the questionnaire used Netscape Navigator, one student reported by e-mail that he had tried both Netscape Navigator and Internet Explorer, and that the system ran fine with Internet Explorer, but not with Netscape Navigator. Subsequent investigation proved this claim to be correct. Clearly, platform independence proved less of a problem than browser independence. The prototype was developed under Windows 95. It was subsequently tested under Windows NT, X Windows and on the Macintosh System 7. Several minor problems were noted and corrected, such as the color of buttons.

The major logistical problem identified by the first evaluation was that the machines that students had most convenient access to were machines running Windows 3.1. Although this courseware requires no plug-ins, it does require a Java-enabled browser, which was unavailable on these machines. Most students were, nonetheless, able to locate a machine on which the courseware could be run. Because most machines on campus were upgraded to Windows 95 by the next semester, this problem quickly diminished in importance.

The major pragmatic problem revealed by the first evaluation was that students did not have a compelling reason to use the system. Although they were given a small amount of extra credit for using the system and completing the evaluation, this motivation led most students to use the courseware for somewhere between one and five hours, by their own reporting. We suspect the usage for most of these students was closer to one hour than five.

There are several key issues that the individual questions of the evaluation were designed to assess. The most important of these was whether students felt that the courseware was a useful tool for understanding the key topics related to graphs, binary trees and sorting. The second key issue was to determine how deeply they used the courseware. The third issue was how easy the courseware was to use. This issue included the clarity of the user interface and the adequacy of the instructions provided. The final issue that the first evaluation was designed to assess was whether the World Wide Web was a suitable platform. Specifically, it attempted to determine whether there were any problems related to the use of specific browsers and whether speed was a problem on any platforms. Each of these questions was answered using a Likert scale that ranged from 1 to 5. Rather than numbers, we used a menu selection of choices that included: Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree. Table 5.9 contains a summary of the responses to these questions. Appendix B.1 contains the numeric averages for the individual questions.

| Average | Issue |
|---|---|
| 4.25 | Effectiveness as a learning tool |
| 3.90 | Depth of use |
| 3.92 | Ease of use |
| 3.52 | Platform effectiveness |

Table 5.9 Spring 1997 evaluation summary

The responses suggest that students had a favorable reaction to the effectiveness of using such as tool. Although not quite as high, the questions that attempted to gauge ease of use also produced a reasonably high score. The questions on platform effectiveness received the lowest response, primarily because of the questions that asked about availability and speed.

Finally, as a part of the evaluation, students were asked three open-ended questions intended to elicit written responses. The first question asked them to list three things they liked about the system. Interestingly, ease of use was the most frequent comment mentioned by seven respondents. Six mentioned the visual aspect of the courseware. A few comments that address the visual aspect follow:

> It is helpful to visualize some programmming [sic] tools in order to have a better understanding of the algorithm. Such a program would be enormously helpful for anyone who doesn't comprehend the use of some concepts.
>
> this educational tool is very essential for students who depend on visual images to understand different fields of sciences specially computer science. i [sic] hope that it will be available for me in all other courses as csci120 for example.
>
> The student can learn at a much faster rate than just reading from a book. Most students don't like reading. Viewing and listening to lecture provide a much faster learning curver [sic]. Pictures, graphs, and examples help reenforce [sic] learning and retaining the information much longer than reading the information from a chapter.

The next open-ended question asked students to list three things they did not like about the system. The need to improve the speed was dominant among these responses. Ten students cited the need to improve the speed and six students commented that it was hard to find a machine powerful enough to run the courseware. The following comment was typical of the responses and captures both the speed and platform issues.

> the system i to [sic] slow. even if I tried it on different platforms -Was hard to find browsers that had the the [sic] add-ins that are required -speed -also make the system work on any brower [sic] and plattform [sic].

The final question asked students to list three things they would change about the system. Several other noteworthy suggestions were made. The first suggestion was modifying the courseware to provide a test or to allow students to do on-line homework assignments that would be electronically submitted. Interestingly, students did not perceive the *I'll Try* capability as a test. This recommendation was adopted and added for the version that was used the following semester. There were several important benefits to its implementation. First, it gave students a more compelling reason to use the system. These homework assignments, unlike the handwritten ones, could be automatically graded—that is they could not be submitted until a satisfactory level of performance was demonstrated. Furthermore, because of the random generation of problems, no two students would have the same assignment.

The next suggested modification was the inclusion of audio. This change was suggested by several students—some suggested music, while others suggested narration. As mentioned earlier, we chose not to add audio because we felt that it might confound other aspects of our analysis.

Although the visually richest areas of data structures have already been included, other topics in the course might be included. Several students recommended that the scope of the courseware be expanded to include more topics. There are several potentially visually rich topics in this course that might have been added, such as linked lists, stacks and recursion, but we felt that the usage for four weeks was sufficient for students to perceive the courseware as an integral part of the course, rather than a novelty.

Several students thought the graphs looked weird because they contained more nodes and edges than the graphs used in class, as we have previously mentioned. Finally, several new controls were suggested by students, such as the ability to pause the animation and stop an animation.

### 5.3.2 Analysis of evaluations for fall 1997

During the fall 1997 semester, students were again asked to use the courseware independently, however, this semester it was assigned as on-line homework problems that applied to the last four lectures of the semester, as had been suggested by the students the previous semester in their evaluations. The homework problems for the earlier lectures were submitted handwritten. Two sections of this course were offered that semester. Students from both sections were assigned the homework problems. At the end of the semester, the students were again asked to complete an on-line evaluation questionnaire. Because they were again using the courseware independently, the questionnaire included questions regarding the browser and platform that they used. A total of twenty-two students completed the questionnaire. Four had used Internet Explorer; the rest had used Netscape Navigator. Students were explicitly advised to use version 4 of Netscape Navigator, if they choose to use that browser. The problems experienced the previous semester, were a result of a problem of version 3 of Netscape Navigator and had been corrected in the later version. Only one student used a Macintosh that semester; the rest used a PC.

This questionnaire included many of the same questions to assess effectiveness as a learning tool, ease of use and platform effectiveness. Because students were expected to use the courseware more extensively this semester than they had the semester before, additional questions were added to questionnaire to gauge the effectiveness of the animations. Specifically questions were included that asked whether the size of the data structures was too small, too large or about right and questions were added to determine whether specific aspects of the animations were clear. Additionally, we now felt questions regarding the frequency of use of various controls would be meaningful. A summary of the responses to these questions appears in Table 5.10. Appendix B.2 contains the numeric averages for the individual questions.

Comparing these three results with the previous semester is instructive. The most noticeable difference between the responses was to the questions regarding ease of use. The responses during this semester were about a half point lower. We attribute this difference to the

fact that the students' depth of use had increased significantly, so while superficially it had seemed easy to use, once they needed to use it more extensively, it became more difficult to use. The written responses confirmed that problems remained with the user interface.

| Average | Issue |
|---|---|
| 4.38 | Effectiveness as a learning tool |
| 3.38 | Ease of use |
| 3.29 | Platform effectiveness |

Table 5.10 Fall 1997 evaluation summary

This questionnaire included questions in three categories that had not been included the previous semester. The first category included questions about the frequency of use of certain controls. Specifically, we asked about how frequently the *I'll Pick*, single step, abort, and speed controls were used. The results indicated that almost half the students never used the single step feature and over half never used the speed control. The other two controls were used with somewhat greater frequency. The written responses suggested that the lack of use was due in part to not recognizing the purpose of the controls. The second category of questions asked students how they felt about the size of the trees, graphs and arrays—whether they were too small, about right or too large. The responses indicated that vast majority thought the sizes were about right. Finally, we used this questionnaire to attempt to determine whether specific aspects of the animations were clear, such as the significance of the check marks on the binary tree traversals, and the arrows on the depth-first graph search. In the majority of cases, the students did not recall. Evidently, they were completing the questionnaire too long after they had used the courseware.

As we had done the previous semester, we included several open-ended questions in this questionnaire to elicit written responses. The questions we asked this semester were somewhat different from the previous semester. One question asked students to comment on whether they felt that the courseware had been a helpful addition to the class. Another question asked them to comment on the future of such educational tools. In these comments, five students explicitly mentioned the value of the interactivity. Let us consider excerpts from two students who highlight the importance of interactivity.

> The on-line homeworks were GREAT! Definitely use them in the future. There will be fewer problems in the future if more people use Netscape 4.04 or higher. About the format, I would say that interactive learning is ALWAYS better than any other type (memorizing, or practicing on a piece of paper). That's why I like this way of learning. The great thing about the Internet, and Java, is that it allows people to do it all remotely, at the best possible personal convenience. In a nutshell, use this

> technology in classes, period. But also, use it as a PRIMARY too for students, and not as an experiment, or as a secondary medium. It works, and works well.
>
> I understand that this is a new thing that you have developed, it would be good to have an online part to all of the homeworks. It helps to go through something yourself - kinda hammers home the point better than notes or a lecture can. This is a good thing. It helps to try something yourself after watching a demo. It also forces you to DO it when you have to submit the results. It would be great to have more of this sort of thing.

Eight students identified the visual aspect as one of the key beneficial components. Because we believe that the students own words best describe their reaction to using the courseware, we have included below are several additional excerpts from the written responses from several of the students who were the most enthusiastic users.

> Overall, the animations were VERY helpful in understanding the algorithms. There is still a problem with robustness and access to computers capable of running the animations—I had to reinstall my operating system before I could get the animations to work. Thanks for all the work putting these animations together!
>
> The on-line homeworks were very useful. The animation showing deletes in binary tree helped clarify that. The most useful part for me, though, was the efficient sort excersizes [sic]. I was feeling very unsure about how they worked, and I am sure that if I had to work it out on paper, it would have taken a lot longer to figure out. Within a few minutes of looking at sorts and trying them myself, I understood the sorts completely. I don't think I could have achieved that level of understanding without the program.
>
> It is like Seeing Is Believing. Other than listening the concepts in class, it is essential to visualize the real graphs and diagrams online. In doing so, students understand much better on the subject. The online assignments are just HomeRun!!! To a certain extent, this kind of educational tool should be employed in the future as well. It is nice to explore these stuff other than writing long & tedious programs for hours.

The final question asked them what things they would change about the courseware. The responses to this question indicated that a variety of problems remained. About a third of the respondents mentioned problems related to the software not working properly on all platforms or with all browsers or mentioned that they had difficulty finding a suitable platform. Approximately half of the students noted some problem with the user interface. These comments suggested clarifying the instructions, making the interface more user-friendly, reorganizing the main page and reducing the need to click on objects with such precision. A few students mentioned problems with the controls. Some students did not recognize the speed control, another indicated that the buttons needed clearer names, one other said that the single step control did not seem to work properly. The following comment expresses the difficulty of finding a suitable platform and the problem with recognizing the speed control.

> It was difficult to find a machine that had Netscape 4.0 or higher. I finally went out to Netscape and downloaded the new version to my computer at work. I never

> found a machine at school that had it. ... I wasn't even aware that there was a speed adjustment on the demo. Where was it? If it was the little bar at the bottom, it didn't produce any effect for me.

The need to reorganize the main page is clearly expressed by the following comment.

> The categories of java applets (ie quadratic sort, binary sort, heap, etc...) need to be additionally labelled [sic] to show what homework assignment they correspond to. I can't say how many times I went to the Homework link, and then forgot which applet tool I had to use!

The written responses also indicated how understandable the animations were. Several students felt that the sorting algorithms needed further clarification. Finally, some students recommended that all the homework assignments be made on-line, as they had in the previous evaluation.

### 5.3.3 Analysis of evaluations for spring 1998

Students used the courseware in the recitation session held in a computer laboratory during the spring 1998 semester, so all students used similar machines and the same browser. For that reason, such questions were excluded from the questionnaire. There were nineteen respondents that semester. The questionnaire was made as similar as possible to the previous semester for purposes of comparison. Two groups of questions were removed, however. The questions regarding appropriate data structure size were removed, because the responses from the previous questionnaire suggested that they were about right. The questions that attempted to ascertain whether certain aspects of the animations were meaningful were also removed, because these questions were determined to be ineffective after reviewing the results of the questionnaire the previous semester. Table 5.11 contains a summary of the results of the questionnaire for the spring. Appendix B.3 contains a summary of the individual questions.

| Average | Issue |
|---|---|
| 4.47 | Effectiveness as a learning tool |
| 3.67 | Ease of use |
| 4.21 | Platform effectiveness |

Table 5.11 Spring 1998 evaluation summary

The responses to questions, regarding the effectiveness as a learning tool, were high, as they had been in both of the previous semesters. Of the three questions that asked specifically about the importance of the animations, the interactivity, and the random generation of problems, the question that concerned interactivity received the highest score, but only by a slight margin.

The response to the ease of use questions was slightly higher than the previous semester, but not as high as the first. It may be that this increase reflected some of the improvements that had been made to the user interface, or it may have been that students, using it in the computer laboratory, were given additional guidance. The average for the platform effectiveness was a full point higher that the previous semester. We removed one of the questions from the previous semester regarding the ease of finding a machine on which it would run, since this question was no longer appropriate. This large difference was primarily a result of the question regarding whether the software worked properly. The previous semester the average for this question had been 2.6; this semester the average was 4.1. Although some improvements had been made since the previous semester, we attribute this large difference to the fact that students were working in the computer laboratory on machines with a browser that was known to run the software properly. We believe that this result supports our decision to have students use the courseware in an environment where it has been tested.

We asked students the same three open-ended questions that we had asked the previous semester. In response to the question about whether the courseware was a beneficial addition to the class, two students specifically mentioned the interactivity as a key feature. One student provided the following comment that highlights the importance of the *I'll Try* mode.

> The ability to read about something and to try it out at the same time would help greatly in all educational environments, not just CS courses. I paid closer attention to the sections which included I'll try, therefore I think it would help the student learn and focus on the material if they had to participate as well.

Six students mentioned that the visual aspect was an essential reason that the courseware was helpful. Two of the comments that mentioned the importance of the visual aspect follow:

> I believe it will help such students like me who learn better visually.

> They were definitely a beneficial addition to this class. It not only provided more information but also consolidated the information recieved [sic] in the lectures. It was a lot easier to understand concepts visvualized [sic] than it would be to follow them through notes and lectures.

Finally, one student remarked that using the courseware helped him learn the material faster.

In response to the second open-ended question asking them what they would change about the system, several logistical issues were mentioned this time. Two students mentioned that the lectures should precede the use of the courseware, although other students felt it prepared them for the lectures. One commented that it should be available outside the laboratory. We should add that numerous students had made this comment verbally during the month that the courseware was in use. Finally, one student noted too many repetitions were required for certain lessons. Again, numerous other students expressed the same comment verbally. User interface problems were still among the suggested changes, although there were only six such comments, about half as many as

the previous semester. The following comment expresses one of the remaining user interface problems.

> I would change the positioning of the buttons such as the choice buttons e.g. It is unbalanced or it is balanced. There is not enough seperation [sic] between the two choices, so buttons for these would be appropriate.

In spite of having made some changes to clarify the sorting algorithms, one student commented that they still require better explanation. Finally, two students suggested that sound or music be added, a suggestion that had been made the first semester the courseware was used.

### 5.3.4 Analysis of evaluations for fall 1998

Students used the courseware in the recitation session held in a computer laboratory during the fall 1998 semester, as they had the previous semester. There was a total of 46 responses to the questionnaire—the largest sample of the four semesters. Three of the recitation sections were held in a laboratory whose computers ran Windows 95 and used Netscape as the browser. This laboratory had been used the previous semester. Newer, faster computers had been installed since the first experiment. Unfortunately, these machines were configured so that Netscape loaded from the network, not the individual machines, which had not been the case the previous semester. The fourth recitation section was held in a laboratory whose computers ran Windows NT. Although these machines were initially configured to run Netscape, the courseware ran so slowly with it, Internet Explorer was installed. With Internet Explorer, the courseware ran satisfactorily. The questionnaire was identical to the one used the previous semester. Table 5.12 contains a summary of the results of the questionnaire for the spring. Appendix B.4 contains a summary of the individual questions.

| Average | Issue |
|---|---|
| 4.10 | Effectiveness as a learning tool |
| 3.96 | Ease of use |
| 3.22 | Platform effectiveness |

Table 5.12 Fall 1998 evaluation summary

The responses to questions, regarding the effectiveness as a learning tool, were high, although not quite as high as they had been in the previous semesters. The response to the ease of use questions was again slightly higher than the previous semester, improving two semesters in a row. We believe that this increase reflected some of the improvements that had been made to the user interface. The written evaluation comments confirm this belief. The average for the platform effectiveness was a full point lower that the previous semester. We believe that this large

difference was primarily a result of Netscape being loaded from the network in the one laboratory that was used by the majority of students. This problem is very evident in the written comments.

We asked students the same three open-ended questions that we had asked the previous two semesters. In response to the question about whether the courseware was a beneficialaddition to the class, the vast majority indicated that they felt it was. Nine students specifically mentioned the interactivity as a key feature and ten mentioned the importance of visualization. One student mentioned hat such courseware enables faster learning. The high number of students mentioning interactivity was somewhat surprising since over half did not have the interactive version. The following comment captured both the interactive and the visual aspect.

> Yes, they were a definitely a benefit to learning the material. Actual hands-on and visual experience makes learning far more interesting and we are more apt to learn and remember the material than if we were to read it from a textbook.

Two students did not respond favorably to the use of the courseware in the laboratory. They felt that it took time away from their project work, which they deemed to be more important.

In response to the second open-ended question asking them what they would change about the system, many comments about the downloading speed, platform independence, and Java appeared. Seventeen students mentioned the downloading speed as a problem. Eleven mentioned Java as the source of the problem. Eight students cited the need for browser or platform independence, while seven students deemed the courseware unreliable. The vast majority of these comments were from the students who used the lab in which Netscape was downloaded from the network. We believe this problem was the cause of most of these comments. The increased level of sophistication of student comments compared to previous semesters was noteworthy. The following two comments are examples of this understanding.

> Greater platform independance [sic] (Java's strong suit) is a necessity if this program is to be implemented on a large scale.
>
> browser/platform independence - hey wait isn't that what java is based on?

Our constant improvement of the user interface in response to student suggestions was apparent from the numerical responses to the ease of use questions and in the written comments. This semester only three comments appeared regarding the need for more instructions and one comment about the need for a better user interface. Having the help feature on by default likely contributed to this favorable feedback. Some students, however, never discovered how to turn off the help feature as reflected by the following comment.

> take away the "this button is inactive during use" lable [sic]...it cover the graph

What emerged as a strong theme in the written comments this semester was the need for some type of concurrent explanation, either written or verbal, with the animation. Again their increased sophistication of how to deliver information over the Web was apparent—in contrast to prior semesters. Twelve students made comments of this kind.

> Work on delivery of information. Web based content shouldn't consist of lengthy paragraphs, but should instead be broken down into smaller parts to ensure readability (see wired.com news stories as an example)
>
> Better connection between the description and the visualization.
>
> Maybe there could be brief explanations while the animation was was [sic] running

We believe that this emergence of new themes reflects the importance of repeated evaluations. Once major problems in the interface were corrected, students were able to focus on more subtle improvements that would enhance the usability of the courseware.

Among the other suggested changes, seven students mentioned the need to improve the speed control. This problem has arisen as computer speeds have increased causing the animations to run too fast. One student mentioned the need for a smoother transition between the single-step and continuous mode—a comment that would only arise from someone who had developed a high skill level in the use of the courseware. Two students mentioned a need for better explanations of the sorting algorithms, one mentioned the addition of audio, and one mentioned that difficulty levels should be added. Finally, this was the first semester that some students never used the version of the software that had the *I'll Try* feature. Six students suggested adding more interaction.

> The only thing I can think of would be to build in a quiz type thing that would continously [sic] ask questions and tell u if u [sic] are right or wrong. If you are wrong it can make you try again. There should however, be a practice button to allow a user to practice before it asks questions.
>
> Let there be more student interaction in the visualization

Lastly, five students commented that the courseware should be available outside the laboratory.

> They're beneficial, but i'd [sic] rather do them on my own time. I feel somewhat rushed while in class. If would be more beneficial if I could just use it when I needed it.

Although students were restricted from using the courseware outside the laboratory as they had been the previous semester, they were given access to the complete version during the two week period after the study was completed and before the final examination. Although the final examination was not planned as the posttest in the second experiment, students were advised that the final examination would contain questions related to the courseware. From the log file data, we were able to gauge the extent to which they used the courseware during this period. We discuss those results in the next section.

### 5.4 Analysis of log file data

The log file data we collected in the fall 1998 semester provided the data necessary to measure the amount of time students used the courseware—data required for answering one of the primary questions of our study. We also turned to the data in the log file as a part of our post hoc

analysis. In this section, we consider how the log file data helped us understand the use of the courseware and how analyzing the data helped us improve the user interface design.

The versions of the courseware that were used the last two semesters contained a feature that logged student activity. The version of the courseware that was used during the spring 1998 semester contained a logging feature that logged every button click with a time stamp. This method provided a complete but overwhelming amount of data that required extensive distillation, consequently in the fall 1998 version, the modified the logging feature provided a single log record each time a student used one of the lessons in the courseware.

### 5.4.1 Analysis of log file data for spring 1998

In the spring 1998 semester, one of our prime reasons for incorporating the logging feature was to enable us to better measure the depth of use of various features to enable us to make any necessary improvements in the user interface design. This semester we also included questions on the evaluation questionnaire regarding the depth of use of various controls. Having both the questionnaire data and the actual usage data gave us the opportunity to determine how accurate the self-reported data was. We measured the strength of the correlation between what students reported regarding the use of various controls with what the log file indicated about their use. Table 5.13 contains the correlations for the three controls, for which both log file and questionnaire data were available.

| Buttons | Correlation |
|---|---|
| I'll Pick | 0.23 |
| Abort | 0.05 |
| Step, Pause, Resume | 0.51 |

Table 5.13 Log file, evaluation correlations

Although all the correlations are positive, only one of three is above 0.5. This result is perhaps not too surprising for two reasons. It is no doubt difficult for students to accurately remember such information and no two individuals likely have the same perception of frequent use. We believe these statistics are significant only to underscore the importance of collecting actual objective data whenever possible rather than relying on self-reported data, which is what we had done with the logging capability. Clearly, subjective data, like the perceived ease of use, must be gathered with evaluation questionnaires.

One of the most important statistics that the log file data provided, was the degree of use of the various controls. Numerous controls were added in response to evaluations from the first

semester. The help feature was added after the second semester. The logging data revealed that only about half the students ever turned on the help feature. The default state of the system was designed to have help off initially. They either did not believe they needed it or they did not notice it. The transcriptions of the think-aloud sessions discussed in the next section suggest the later possibility to be the most likely. Consequently, the initial state was modified to have the help feature initially on.

Next, we consider the four controls that affect how the animations run. These include *Step*, *Pause*, *Resume* and *Abort*. The button most frequently used was *Abort*. Over 90% of the students used this button at least once. Clearly this button is essential because it allows lengthy animations, such as the bubble sort, to be stopped once they are understood. By contrast, the *Pause* and *Resume* buttons were essentially unused. Only about 10% of the students ever used these buttons. One student remarked, in the evaluation from the spring 1998 semester, that he felt there were too many controls. The value of these two buttons needs further evaluation. Excessive controls may obscure those that are most important. Finally, we consider the single step feature. Not surprisingly, this button had the highest average number of clicks. What was disappointing, however, was that it was only used by about 65% of the students. One of the evaluations and the think-aloud transcripts suggest that some students may have been unaware of this capability.

As expected the *I'll Try* button was used by most students. What was less predictable was that the number who used the *I'll Pick* button was almost 80%. This feature, which is available in selected lessons, put students in an exploratory learning mode. We were pleased that its usage was so high.

In the explanatory text, implemented in HTML, that accompanies each lesson, certain terms are hypertext. Clicking them produces a small window containing their definition. About 62% of the students referenced these definitions at least once. The statistics for all these buttons are shown in Table 5.14.

### 5.4.2 Analysis of log file data for fall 1998

The log file helped us answer one additional question in the fall 1998 semester. The previous semester, we restricted students from using the courseware outside the laboratory before the final examination. Many students objected. We provided a four hour period when students could use the courseware in the laboratory before the final examination. Only a few students chose to use it. The following semester students again expressed the desire for freer access to the courseware in their evaluation comments. Because the quizzes given after each laboratory were the posttests, we were able to make the courseware available to students during the two week period after the final laboratory and before the final examination. Students knew that the final examination would contain questions similar to the questions they had been given in the laboratory quizzes. The log file provided a measure of the level of use of the courseware during that two week period.

All users of the courseware are asked to provide their name before beginning use, but users can elect not to enter their name. The log file during that two week period indicates that fourteen class members used the courseware and entered their complete names. Four other entries are first names of students in the class. That fact that students elected to use the courseware confirms the evaluation results that expressed their view that the courseware was helpful for learning the topics of graphs and trees.

| | Average | Standard Deviation | Percentage Who Used |
|---|---|---|---|
| Help | 1.9 | 2.6 | 53.1 |
| Step | 77.8 | 186.5 | 65.6 |
| Pause | 0.2 | 0.6 | 12.5 |
| Resume | 0.1 | 0.4 | 9.4 |
| Abort | 11.1 | 10.3 | 90.6 |
| Show Me | 5.5 | 4.5 | 93.8 |
| I'll Try | 11.6 | 5.0 | 96.9 |
| Random | 1.6 | 1.6 | 65.6 |
| I'll Pick | 2.8 | 2.6 | 78.1 |
| Definitions | 4.5 | 6.0 | 62.5 |

Table 5.14 Control button usage

Another measure of the usage of the courseware can be readily seen in Figure 5.5, which shows the number of log file entries during the two week period before the exam and the two weeks following it. The exam was on December 17. On that day there was almost 200 entries in the log file. The day before it, there were about 50 entries. This graph also reveals a well-known fact—most students study for exams immediately before them. The activity during the remainder of the period before the exam was only slightly higher than it was during the two week period following the exam. The courseware has several external links to it that generate about ten log file entries per day on average.

We also examined whether those students who used the courseware during the period between the end of the study and the final examination benefited from its use. In Table 5.15, we compare the students who used the courseware during this period with the group that did not. Note that some log file entries during this period did not contain names.

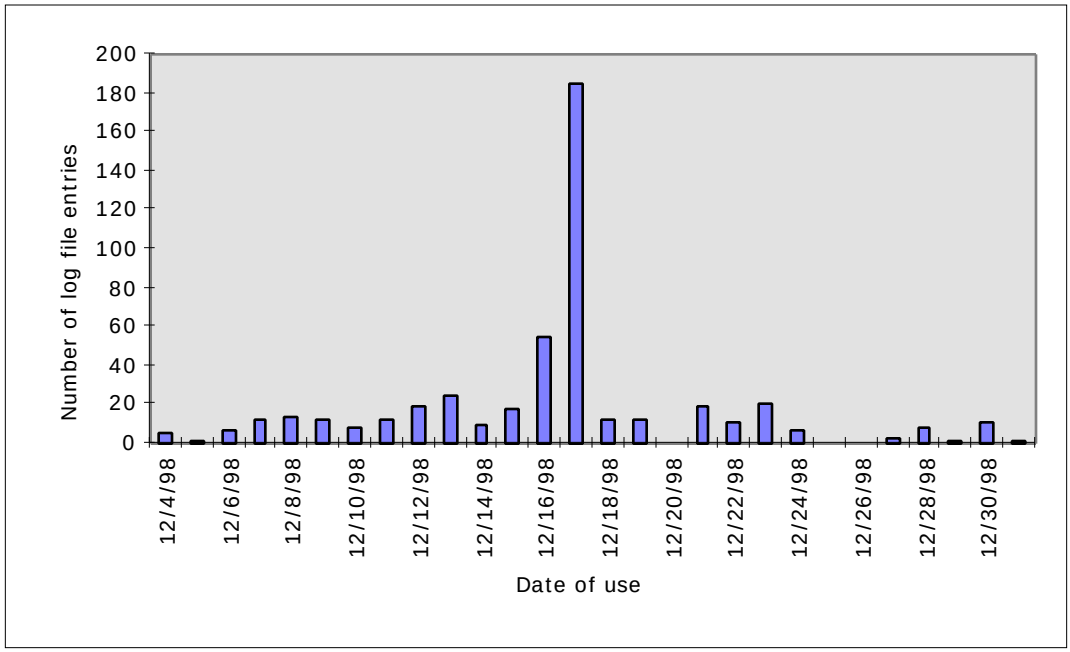



Figure 5.5 Courseware use before and after final exam

Those who used the courseware to study for the final examination had a greater improvement in their final examination scores compared to the scores they received on the posttests that followed the study, than those who did not use the courseware during that period. The improvement is not statistically significant, using a one-tailed t-test ($t(50) = 1.55$, $p = 0.064$).

| | Final Exam | Posttest | Improvement |
|---|---|---|---|
| Nonusers | 69.8 | 63.6 | 6.2 |
| Users | 75.7 | 61.9 | 13.8 |

Table 5.15 Effect of courseware use prior to final examination

We conclude the discussion of the log file data with one final observation. A slight modification to the collection of logging data would have been helpful during the fall 1998 semester. We previously noted that we reduced the logging from a log record for each button click in the spring 1998 log file to one record per lesson in the fall 1998. In retrospect, it would have been better to have logged a new record each time students changed from the *Show Me* to the *I'll Try* mode. Although we recorded the number of button clicks in each mode and the number of times that students switched between modes, this additional information would have given the time spent in each mode and helped better characterize the usage behavior of students who seemed to be guessing.

## 5.5 Analysis of think-aloud transcriptions

The think-aloud transcriptions are the final ancillary data we analyze. During the spring 1998 semester we asked students, whom we felt might be adept at thinking aloud, to be recorded while using the courseware. The recording from the first week was largely inaudible for a variety of reasons, so was not transcribed. The transcriptions for the final three weeks are in Appendix C. Because the student who was recorded the second week was so skilled at thinking-aloud—a skill that is not shared by all students—we requested that he be recorded for the remaining weeks. This student was also one of the best students in the class. His final examination score was the second highest in the class. His learning style was visual and reflective.

We regard the think-aloud transcriptions as one of the most important pieces of data that we gathered during the study. They provide some important insights into several important areas. Problems with both the user interface and the clarity of the animations become apparent from these sessions. Furthermore, we gained a deeper insight into the overall process students use and the stages of understanding that occur during its use.

Let us consider the individual sessions. The first transcription was of the binary tree traversals. It is clear from the transcriptions that there are two important things that the student was trying to understand. The first was the meaning of the check marks. To help distinguish among the three recursive traversals, a check mark is added to each node each time it is touched. He mentioned the check marks eight times. His final assessment of their meaning was

> Maybe the check mark determines or represents which path it's taking, I mean the possibilities which is probably what it is doing

Clearly, we have succeeded in conveying the importance of the check marks. What is less clear is whether they conveyed the intended meaning. We believe that small additions to animations such as these check marks greatly influence the understandability of animations. We call these features *visual semantic cues*. What we have not explored is whether a different visual semantic cue might have been more effective. Instead of check marks, three arrows, one that pointed down, one that pointed up from the left and up that pointed up from the right might have added greater meaning. We should note that the text that accompanied the lesson explained the meaning of the check marks, but was either not read or not remembered.

The other idea that the student tried to understand was the order in which the nodes are visited. He was able to articulate this behavior with reasonable clarity.

> It starts at the top and then takes the left-most branch and then once it's exhausted, once it reaches the end it goes back up until it finds another branch and then from there it goes to the left-most and then it continues along that way until it's all done.

The level order traversal, as expected, presented little difficulty. After he believed that he understood the algorithm, he switched to the *I'll Try* mode. This was the first time he had used it. He had no difficulty with the preorder traversal, but when he began the inorder he became confused. In spite of having said several times during the *Show Me* mode that “this seems to

make sense," he later commented: "I'm confused. I don't understand that at all. I didn't get that at all." We believe that this observation is significant. It illustrates that watching the animation, one may believe that he understands it, but only when one is made to replicate the behavior can his understanding be confirmed.

The second lesson was binary search trees. Understanding the structure of these trees was the student's first task, which he accomplished rather quickly:

> Oh, I see. This is sort of sorted. It starts off with the first number and if it's greater, it goes to the right and if it's less it goes to the left.

This lesson is the first one that contained the *I'll Pick* feature. He used it very early and said "interesting thing would be to see if it's between like 15 and 89 what would happen. I'll pick." This feature is another aspect of interactivity—one that many algorithm animation systems have included in some form. His comment indicates that he is about to perform an experiment. Clearly the ability to pick puts the student into a discovery or experimental mode of learning, which is favored by some learning theorists. In this mode, he begins making predictions:

> Let's try 5 means—it should probably put it—it should probably put it to the right, if I'm not mistaken.

He was able to understand the insert operation with little difficulty. The delete operation, as might be expected, presented much greater difficulty. The algorithm for deleting a node from a binary tree differs depending upon the number of children of the node whose value is to be deleted. The two-child case is the most difficult and this case was the one with which he had difficulty. He was unable to understand why the value was moved.

> Why did it move 43? I have no idea. This is neat stuff, but really weird. Why did it do that?

He made a valiant effort to understand this part of the algorithm and spent a great deal of time on it. When told that there was Ada code for the algorithm, he commented "The code I'll be able to deal with." After extensive tracing through the code, he still did not understand the algorithm. Finally, with a single sentence verbal explanation, he understood. The explanation was that it is the inorder successor whose value is moved—that node is located one right and all the way to the left. The difficulty in understanding this algorithm clearly highlighted a problem in the animation, but also returns us to the importance of visual semantic cues. In the final version of the courseware, we highlighted the path from the node containing the value to be deleted to the node containing its inorder successor.

The following week the same student was recorded thinking aloud while using three more of the binary tree lessons. This week he had the version of the courseware that did not have the *I'll Try* mode implemented. The first lesson was the priority queue implemented with a heap. He began with the enqueue operation. His first observation was to note the location where new nodes are added. He comments: "Oh, it goes to the right-most possible, the right-most possible." Although the log file indicated that only slightly over half the students ever used the single step

mode, this student was quick to realize that it would be helpful for this lesson. "I like doing it by single step better. I think it's easier to deal with it that way." He soon observes the process, sometimes characterized as sifting, in which the new value added is swapped with its parent until the tree again becomes a heap. He describes this process as follows:

> Now if I enqueue let's say 99, what it should probably do is it stick it, um, stick it to the right of 57. Yeah, and then it'll switch there and it'll move itself all the way up.

Next, he tried the dequeue operation. Not fully understanding the dequeue operation, he wanted to pick the element to be dequeued. It was explained to him that the *I'll Pick* mode only applies to the enqueue operation, because there is only one value that can ever be dequeued—the largest. The help feature was demonstrated to him. When the help feature is on, and the mouse is moved over the *I'll Pick/Random* radio buttons, this explanation is provided. When the help feature was demonstrated to him, he made the following significant comment.

> Oh. That's neat. I didn't know how that worked. Ah, OK, cool. OK. That's very useful. I think that thing should be on whether or not you select it.

We have already noted that only about half the students ever turned on the help feature, but that information gave us no insight why. It may have been that they did not believe they needed it. From the preceding comment, it is more likely that, like this student, they did not know how it worked. Clearly, having it on initially, as the default, is best solution, as this student suggests. He quickly finished with the dequeue operation commenting: "OK. This is pretty simple. I get this pretty easily. That helped."

The second lesson that week was height-balanced trees. This lesson was one of the three that does not involve any animation, only visualization. He began with the tree height and was quick to characterize the height of subtrees by the following comment:

> OK, that's easy, all the leaves are one, then everything else is defined by their furthest distance away from the leaves. That's easy to understand.

It was never explained to the students that some students would have the *I'll Try* mode each week and others would not. Having had lessons with it the previous week, this student was looking for it as his comment illustrates: "I'll try, where's I'll try." He had little difficulty in understanding the concept of balance factors. He quickly summarized his understanding as follows:

> Ah, the factor difference between the height of its left and right subtree. Left minus right. Ah, OK. Yeah, OK, makes sense.

The final topic for this lesson was height-balancing. He made a noteworthy comment while working on this topic:

> Let me try some, let me try—. It would be nice if we could maybe create our own tree there, and see, that'd be cool, and then see whether it's balanced or not.

This student clearly enjoys the experimental side of learning and would have liked to test his knowledge by creating a tree of his own. The *I'll Pick* feature allows students to select values in

some lessons, but no capability was provided to allow students to create the entire tree or graph. This possibility requires further evaluation. At first he comments that he is confused about the height-balancing but then studies the code and comments:

> Oh, I see now, once looking at the Ada code I understand what the—the idea is behind this height-balanced trees business.

There is one final comment that he made in this lesson that we believe is important. Although, as we have noted, this lesson did not contain the *I'll Try* mode, he imitates this mode by predicting the result as the following comment reveals:

> This will not be height-balanced 2, true.

This behavior raises an important issue, which is whether the tendency to predict is a natural one or whether it was conditioned by previous lessons, in which students were required to predict. To avoid this potentially confounding factor, in the final experiment, students were given one version of the courseware.

The final lesson for that week was categorizing binary trees. The first two categories are almost complete and complete. He is able to summarize his understanding of almost complete trees rather quickly:

> Almost complete means that there's—oh, I understand that, that's means that you have one or less rows only missing, I mean less than one row missing.

The most difficult property for this student was determining whether the tree was a binary search tree. His confusion is summarized by the following comments:

> Yellow subtrees wrong. Yellow subtrees are wrong. OK, why is the yellow subtree wrong? That's a heap, isn't it? This is AVL. OK. Now, all I need to do is remember what a binary search tree is. Then I'll be set. OK. All right, now. What I need to do is determine what, remember what a binary search tree is. Can I look at the previous animations or previous text or whatever?

His comments highlight two important issues. First the visual display of why a tree fails the binary search tree property is hard to understand. The binary search tree property is more complicated than the heap property. Failure to meet the heap property can always be illustrated by identifying one or more branches that violate the heap requirement. A tree can fail the binary search tree requirements due to a faulty relationship between nodes many levels apart. The second important issue that his comment reveals is that although he completed a lesson on binary search trees the week before, he has forgotten its definition. He returned to the lesson of the previous week to try to better understand this concept, but was still confused. It was not until he was reminded that there was a definition available in the text that accompanied the lesson that he finally understood this concept. (Note that the researcher's comments are italicized.)

> *Look at the definition.* Where is the definition? *In the text.* Textbook? *No, no, if you look at the text, and you click on binary search tree.* Oh, cool. Maybe that should be the same thing in the visualizations.

His comments highlight his lack of awareness of the availability of the definitions. Recall that the log file data indicated that only slightly more than half the students ever accessed the definitions. We should note that these students were using a monitor with relatively low resolution. On a higher resolution monitor it would have been possible to have viewed the complete text and applet simultaneously. User interface issues of this kind are inherent in the design of Web-based courseware that can be viewed on many different platforms. Once the student read the definition he was able to complete the lesson without difficulty. He makes one final comment that is noteworthy:

> Do I still have to do sixteen? Do I have to all sixteen of these things? *What do you mean?* Like do I have to do sixteen more of everything?

His comment is similar to some of the comments in the evaluations. It indicates that, for certain lessons, twenty correct responses was excessive. In the final version, the number of required responses was tailored to the lesson.

The final week the same student was recorded using the final three lessons, which all involved sorting. These lessons contained the *I'll Try* mode. The first lesson was the heap sort. The student's articulated thoughts for this lesson are less revealing than for many of the previous ones. He seems to vacillate between confusion and understanding. He makes fewer attempts to characterize what is happening than in previous lessons. One of his few attempts is:

> All right, it starts from the right, I guess. Now, it goes back from here. All right, OK, it sweeps from the right, it looks like.

He concludes the lesson with the following comments:

> OK, I think I generally got the idea here. . . . A weird way of sorting a tree. OK. It's moving everything around everywhere. This is so weird. It's so weird. OK, I think it's done. Oh, oh I see. I understand this.

Like all the more efficient sorts, the heap sort is rather subtle one, so perhaps his comments are predictable.

The second lesson for that week was the quadratic sorting algorithms. These algorithms are all conceptually very easy, which was confirmed by his articulated thoughts. He was quickly able to understand the bubble sort and summarized his understanding as follows:

> OK, all right, I understand it. If it's lower, you swap, and if it's higher you leave it alone. So, this is very easy and intuitive. . . . Yeah, this is very simplistic. I like the bubble sort. It is an easy concept to understand.

The insertion sort presented no greater difficulty and he characterized its behavior as follows:

> Oh, I see. So what does is each time it finds something, it tries and put it in the place of some, of everything already been sorted. OK. Makes sense. Yeah. All right, that's easy. . . .OK, yeah, so selection just basically, um, picks each one that hasn't been done yet and swaps it with the lowest one that not there yet. OK, easy enough.

Finally, he described the behavior of the insertion sort as follows:

> Oh, it just kind of sticks it in. . . . Oh, I see, it just does that between—oh, OK, that's pretty simple, it just puts the each bar between two other bars, but the next bar is moved to wherever it's supposed to go. OK. That's easy enough.

He makes one final noteworthy comment about the fact that bars that are of almost the same height are hard to distinguish.

> These—I think maybe these numbers should be, um, like the graph should be easier to distinguish because even if you understand it sometimes it's confusing because they're almost the same size. That would make it much, much easier to deal with.

Because the actual numbers are provided below the bars, we decided against any changes in response to this comment.

The final lesson was the efficient sorting algorithms—the merge sort and the quick sort. He quickly observes an important aspect of the merge sort that he articulates as follows:

> Oh, I see what it seems to be doing in merge is it does the first 2, then 4, and then 8 and then 16. Ah, interesting, interesting, interesting. OK, now I understand it. Have I seen enough to go for it?

It is noteworthy that he asks himself whether he is ready to try it. If we return to the statistics from the log file we see that the average number of clicks on *I'll Try* is about double the average for *Show Me*. What this statistic means is that about half the time students returned to watch the animation after trying it. Once in the *I'll Try* mode, he seems to vacillate between understanding and not fully understanding.

> This, this, the way it works, it takes a few clicks to figure out how this works. But, I still have no clue how it works. Um, oh. It takes a while to figure how this thing works, but afterwards, it's working!

The quick sort was last. His first observation concerns the line that was added to the quicksort to make it easier to understand. Refer back to Figure 3.8 for an illustration of how this line is used in the quick sort.

> Oh, this idea with the line thing is pretty cool. The line on quick sort makes it much easier to understand.

The quick sort is a rather subtle algorithm, yet he is able to understand it with relative ease. He expresses his understanding in terms on this line:

> Oh, I see what it does. Everything greater than the first thing it pushes somewhere else or something. . . . I see. I understand how this works. It swaps the—it swaps anything that's above the line with the first one from the right that's below the line. So, everything from the left that's above the line, it swaps with something from the right that's below the line.

Given the complexity of the algorithm, his characterization of its behavior is amazingly clear—even clearer than his descriptions of some of the simpler algorithms. Most importantly, the line is a critical component of his description. We believe that this example demonstrates how important such *visual semantic cues* are.

Finally, let us summarize our most significant observations from the think-aloud protocol. Because we observed this student, we can characterize the learning strategy that he used. He always watched the animations first. After he felt he understood them, he tried them to confirm that he understood them. We believe, from our observation, that he benefited from his use of the *I'll Try* mode. From the think-aloud protocol we were also able to gain further insight into information that we gathered in the log file. The log file had shown that many students did not use the help feature. By observing this student, it was apparent that the lack of use was probably because they did not notice it. The final benefit of the think-aloud protocol was that it helped us understand whether the animations were understandable, and whether specific features of the animations were helpful or not. Our student always tried to express his understanding of what he observed. From his comments it was clear that the check marks in the binary tree traversal were not as effective as we had hoped. By contrast, the line in the quick sort appeared to greatly help him express his understanding of what he was observing.

# CHAPTER 6

# CONCLUSIONS AND FUTURE WORK

We have developed Web-based courseware containing algorithm animation and data structure visualizations that we have used for four semesters in an introductory data structures course. We have conducted two experiments with the courseware to determine whether the interactivity that we included in the form of the *I'll Try* capability has a significant benefit and whether student learning styles influence that benefit. Now, we summarize the conclusions that can be drawn from our work and suggest several areas for future work.

## 6.1 Conclusions

Our work has addressed a set of primary questions about the effectiveness of the interactive aspect of our courseware and the role of learning styles and set of secondary questions regarding how Web-based courseware can be used most effectively. Let us consider what conclusions can be drawn regarding each the two major sets of questions.

### 6.1.1 The benefits of interactivity

Our study has shown that although interactivity has the power to significantly increase the amount of time students spend using such courseware, this added time does not necessarily contribute to increased understanding of the material presented. Furthermore, the results of both of our experiments, each with a slightly different experimental design, indicated that the students who used the interactive version performed worse, although not significantly so. Our explanation of these results is that some students who use the interactive version, employ an ineffective learning strategy—using the *I'll Try* mode too soon. They find it entertaining and therefore spend more time with it, but do not achieve the desired result—learning the topics.

Although we have not verified this claim in our study, we believe that the interactivity might be an asset to learning, when properly used. Students use the courseware properly when they use the interactive mode to confirm their understanding of the material. Improper use results when students use the interactive mode before they have watched the animations sufficiently and simply begin guessing. We believe that we observed proper use by the student who had participated in the think-aloud protocol. He used the *I'll Try* mode only after he felt he understood the material. The interactive mode then either confirmed his understanding or caused him to realize that there was an aspect that he had not yet understood.

Our post hoc analysis provides some indication that the weaker students were hurt most by the interactivity, presumably because they were more likely to begin guessing rather than trying to

first understand the concepts. It also suggested that provided students were not guessing, the more they used the interactivity, the better they performed on the posttests. Consequently we believe one key to gaining any benefit from this kind of interactivity is its proper use.

Furthermore, the students who used the interactive version scored better than the other group on several individual questions in the second experiment. These questions pertained to some of the most difficult lessons, ones that involved animation, not just visualization. These results suggest that the benefit of interactivity may only become readily apparent on the most difficult lessons—the animation of complicated algorithms.

Although our primary experiment did not reveal a clear benefit to interactivity, the ancillary data did suggest its potential value. The feedback from the evaluations indicates that students believe that the interactive version is useful. In the evaluations, one student specifically noted that he paid closer attention to the sections that included the *I'll Try* mode. Students were asked to rate the importance of the *I'll Try* mode, the animation and the random generation of problems. The *I'll Try* mode received the highest rating both semesters these questions were asked. Evaluations from students who used both versions indicated they preferred the interactive version. Students who did not use the interactive version during the second experiment expressed a desire for the interactive features. The think-aloud transcriptions also provided evidence of the potential benefit of interactivity. It revealed that students may believe they understand an algorithm after watching it, only to find when they try to replicate it, they do not understand as well as they thought.

### 6.1.2 The influence of learning styles

We administered the Felder-Silverman learning style inventory to students in both of our experiments. This inventory measures four learning style dimensions. The two that we felt were the most pertinent to our study were the processing and input dimensions. The two poles of the processing dimension are active and reflective. In our second experiment, the active students outperformed the reflective students, but not significantly so. The two poles of the input dimension are visual and verbal. In the second experiment the visual students scored higher, but not significantly higher than the verbal students. In addition, there was no significant interaction between either of these dimensions and which version of the courseware was used. Finally, the differences for both of the remaining two dimensions were not significant.

Although our primary experiments revealed no statistically significant results with the learning style instrument that we used, the evaluations did provide some insights about the influence of learning styles. As an example, one student indicated that the courseware would help such students like her who learn better visually.

Our conclusion is that learning style differences as measured by the Felder-Silverman learning style inventory are probably not a significant factor affecting student performance when they use courseware such as ours. Nor is there any interaction between student learning styles and

the presence of absence of interactivity. Our results suggest that other student differences, such as student ability, are more likely to influence how students use such courseware. Additional studies would be required, however, to verify this claim.

### 6.1.3 The effective use of Web-based courseware

Next we consider our conclusions to the secondary questions of our study. The first of those questions was whether such courseware should be used and if so, how it can be used most effectively in an elementary data structures course. We believe that we have demonstrated that such courseware should indeed be used. The evaluations confirmed that students perceived the courseware as a helpful addition to the course. The high scores on the questions concerning its effectiveness as a learning tool together with many favorable written comments are evidence that courseware of this kind is well accepted and appreciated by students.

When asked about the future of learning tools of this kind, we received a wide range of responses. Some students have become so comfortable with such learning environments that they are prepared to forego traditional classroom environments. The response from one of our students suggests such a level of comfort.

> I think that class, as it stands now, should not be. We should all have live hook ups to a teacher that might be next door or in another country. I don't want to have to leave my computer even. I love my computer.

His response was somewhat atypical, however. The more common response was that although they felt that such courseware was a helpful complement to classroom instruction, that it should never become a replacement. The following responses were more typical.

> I think that these types of tools should play a secondary role in education second to a teacher.
>
> They should continue to play an increasing role in education, but it must also be accompanied by traditional education methods. A 50/50 split would be ideal.
>
> I think it is the future. But I hope it doesn't replace the human teacher.

Given that most students are ready to accept such courseware in a complementary role, we next consider our conclusions regarding how can it best complement the traditional classroom environment. Among our early observations was that to provide any benefit, courseware must be used. To ensure its use, students must be given a reason to use it. After the first semester, they suggested that it be changed to on-line homework problems, recognizing that they would spend more time on assignments that influence their grade, even ones whose grade contribution is small. We believe their suggestion was an appropriate one. The next decision was whether they should use it in the laboratory or wherever they chose. The difficulty of finding stable platforms on which it would run properly during the fall 1997 semester convinced us that the laboratory setting would be preferable. Its usage during the subsequent two semesters confirmed this choice. Students

objected, however, when we restricted its use to the laboratory. Our final assessment is that they should be expected to first use it in the laboratory, but be permitted to use it outside, recognizing that they needed to find a platform on which it would run properly.

If learning to use courseware requires more effort than learning the material that it is designed to teach, few students will regard it as a helpful addition. We recognized the importance of designing courseware that was easy to use early in our study. We have tried many of the algorithm animation systems that are currently on the Web. In our judgment, many, if not most of them, are not easy to use. In designing our courseware, ease of use was a far more important consideration than generality. The feedback that we received from each of the ancillary sources, the evaluations, the log file and the talk-aloud transcriptions, has helped us improve the interface each subsequent semester. Furthermore the three sources were complimentary. While the evaluations gave us an overall measure together with specific comments, it did not give us accurate measurements of the usage of the controls. The log file data provided us with this information. While the log file indicated that some controls were underused, it did not reveal why, which was revealed, in some cases, by the think-aloud transcriptions. The feedback we received resulted in numerous changes to the user interface, including the addition of more controls, the addition of a help feature, and the inclusion of a separate interaction panel. Although, some possible improvements remain, we believe that we have achieved the goal of creating courseware that is reasonably easy to use—primarily because we made such extensive use of these three sources of student feedback. Our experience has convinced us, however, that this goal can only be achieved by repeated evaluations together with direct student observations.

In summary, our experience has shown that Web-based courseware, that is easy to use and easy to understand, is well accepted by students as a component of the overall learning experience. On-line homeworks or laboratory exercises are one effective way to achieve this goal.

### 6.1.4 The Web as a platform for data structures courseware

The next of our secondary questions was to determine how well the World Wide Web served as a platform for courseware of the kind we developed. Clearly the compelling reason to use the Web is that it simplifies distribution and expands the availability of such courseware. Our conclusion regarding the Web as a platform must be, however, that it remains very problematic. When we began our study two years ago we encountered significant problems. Some of these problems, such as the availability of machines that had the necessary software or hardware capability to run the courseware, has improved. Other problems that we encountered the first semester, such as browser-dependency, do not seem to have improved significantly during this two year period. The very first semester we used the courseware it ran so poorly with version 2.0 of Netscape as to be unusable despite running fine with the version of Internet Explorer available then. During the last semester of the study, a comparable problem remained. A computer

laboratory that was used for one recitation section was equipped with computers running Windows NT. As had occurred the first semester, the courseware was unusably slow with Netscape, but worked fine with Internet Explorer. Platform independence also remained a problem. The courseware did not run properly in some Macintosh environments during the final semester.

These difficulties underscore the need to use such Web-based courseware in a known environment that contains hardware and software that is able to run it properly. It reinforces the importance of having students use the courseware in a closed laboratory setting—at least initially. Hopefully these problems will be solved in time. Their solution is clearly critical to the effective use of any significantly large Java applets.

### 6.1.5 Making animations understandable

Algorithm animation was a critical component of the courseware that we designed. As important as the overall ease of use, was the ease with which the animations were understood. The final of the secondary questions was to gain some insight into how animations can be made most understandable. We do not claim to have made any major strides in answering this question, but the process of our research has certainly emphasized its importance. Although algorithmanimation has been done for almost two decades now, this question has not been given adequate attention. The method for animating common data structures is well understood, but how to convey the subtle aspects of the behavior of particular algorithms is not. This difficulty was apparent in our study in several algorithms. Among them was the deletion in a binary search tree and many of the sorting algorithms.

The evaluations did provide some feedback on this question, however. Problems understanding the sorts were noted in some of the evaluations. The evaluations proved ineffective in measuring whether specific visual cues, like check marks, were understandable. The think-aloud transcriptions were far more helpful in that regard. We believe that we were able to improve the understandability of the sorting algorithms, but do not claim to have explored this aspect as fully as is possible. We find it significant that well-chosen visual semantic cues appear to greatly affect the understandability of algorithm animations. Although we believe we have achieved some success in enhancing the understandability of certain algorithms—the quick sort in particular, the broader question of how to achieve this in general is a more difficult one. One important conclusion we have reached is that designing clear easily understood animations requires the same effort required for designing courseware that is easy to use. Student feedback and observation are the key to success. This question is clearly one that is worthy of additional research. We address it further in the next section on future work.

## 6.2 Future work

Our experience developing and using our Web-based data structures courseware together with the experiments that we have conducted have led us to believe that there is a need for further work in several areas. A formalized understanding of how to design animations is one area that warrants more study. The second is exploring how audio, specifically narration, could be used to improve the understandability of animations. Next, finding ways to reduce the cost of building courseware containing animations is needed—specifically creating more high level tools that would make this process easier and less costly. Finally, the results of our study have shown that investigating methods for including interactivity in a more guided manner, needs to be studied.

### 6.2.1 Formalizing animation design

Let us consider the first area that we believe requires further study—investigating how to best design algorithm animations to visually convey the more subtle aspects of certain algorithms. Tufte [TUF97] has discussed the general problem of visual explanations, but his work offers little to the specific area of visually displaying algorithms. Most of the work on algorithm animation has focused on the more technical issues such as producing algorithm animators that can handle a wide range of algorithms or data sets. Much less attention has been given to studying the how to design the animations themselves—specifically what features of animations are important to their understandability. Most animation design has been ad hoc, relying primarily on how similar algorithms have been animated by others. In their discussion of graphical representation in general, Scaife and Rogers [SCA96] note that there has not been significant progress in developing a general framework to guide the design of graphical representations.

Work is needed on two fronts to improve our understanding of this process. First we believe that more empirical studies are needed. Specifically, studies that compare the understandability of the same algorithms animated in a variety of ways would be quite helpful. For example, several different animations of the quick sort, or perhaps the Shell sort, could be compared. The goal of such a study would be to try to identify the features, which we have called the visual semantic cues, that contribute most to their understandability. Although quantitative measures, such as scores on posttests, would be helpful in measuring this understanding, our experience has led us to believe that think-aloud transcriptions best reveal whether a specific feature contributes to the understandability of an animation. The ability of a student to clearly articulate what an algorithm does and frame that understanding using a specific feature, is substantial evidence of its value. Recall our addition of the line in the quick sort and our student's characterization of the behavior of the animation using the phrase, “It swaps anything that's above the line with the first one from the right that's below the line.” By contrast, in the same animation, we added two arrows that moved toward each other to represent the pointers to the values being compared. The same student never mentioned these arrows, which should cause us to question

whether they were effective. From posttests we can only infer that features are effective. When we use the think-aloud transcriptions the evidence is much more direct.

Although one study of a particular algorithm would help us discover how that algorithm is best animated, what is really needed are many such comparative studies of different algorithms. Only then would we be able to theorize why certain features are effective. We could then test our theory by predicting which features would be effective and then evaluating our predictions. The eventual goal would be to provide a set of guidelines to future designers of algorithm animations. These guidelines would likely provide guidance of two kinds. Perhaps, we could generalize on how specific algorithm characteristics, such as recursion, are best represented. Possibly, what might emerge would be a set of program plans together with recommended methods for animating each plan. In addition, we would expect that recommendations also might be provided in the reverse direction. For example, guidance might result on how and when to use specific features, such as lines, arrows, and check marks. In addition, a better understanding of when and when not to use color would be extremely helpful as a part of any comprehensive set of guidelines.

Besides beginning this investigation on the empirical front and moving toward the theoretical, we believe that it would be equally important to also work on the theoretical side and move toward empirical confirmation. Some work has been undertaken in this area. As already mentioned, Roman and Cox [ROM92a] have characterized animation as a mapping from a program to its graphical representation. They have defined three categories of visual expression (1) visual objects and their attributes, (2) visual relationship between these objects, and (3) visual events. Furthermore, they have attempted to categorize various levels of abstraction in animations. We believe these ideas are a useful beginning. What is still lacking is a more formal definition of animations and how the meaning of algorithms is best translated to their visual counterpart.

### 6.2.2 Exploring the use of audio

We chose not include audio in our courseware to avoid the possible confounding effects it might produce. It is important to recall, however, that in three of the four semesters that evaluations were given, several students in each of those semesters recommended that audio be added in some form—several suggested the addition of music. We have already noted that auditory icons could be used to emphasize correct or incorrect responses. In addition, they might be effective to convey the semantics of an animation—such as the difference between moving into and out of recursion. We believe, however, that the greatest benefit from audio could come from narration—specifically, dynamically generated narration.

In our observations of students using the courseware, several things became apparent. Textual explanations did not seem to be an effective way to communicate various aspects of animations, because either they were not read or not remembered. The think-aloud transcription of the binary tree traversals illustrated this point. Although the meaning of the check marks was

clearly explained in the text, that explanation appeared to have been missed. Second, when students were confused about a particular aspect of an animation, a simple verbal explanation, often no more than a sentence was sufficient to clarify their confusion. The need for such explanation became quite apparent from the evaluations the fourth semester. Once most other user interface problems had been corrected, the need for such concurrent explanation emerged as definite theme in the evaluations.

For these reasons we believe narrating the animations would substantially improve their understandability. Because the data sets are randomly generated, dynamically generated narrations would be the ideal. For example, as a value was being inserted into a binary search tree, the narration might be "because 85 is greater than 53, we move to the right child." We have already noted that some studies, such as those by Mayer and Anderson [MAY91, MAY92], have confirmed the benefit of concurrent narration. Mayer [MAY97] has proposed a theory he calls the generative theory of multimedia learning, which explains how learners connect the visual and verbal information.

We should note, however, that digitized audio, with the exception of music generated in the MIDI format, produces very large files. Although progress has been made in compressing such files, technical challenges remain if we wish to stage such narrated animations on the Web.

### 6.2.3 Developing algorithm animation authoring tools

Our study has convinced us that Web-based courseware of the kind we developed for our study will become an important part of future educational materials. The students' assessment of the value of such courseware was uniformly high each semester it was evaluated. One of the limiting factors to the development of such courseware is its high cost. Not only does the initial development require substantial effort, extensive evaluation is needed to ensure that the courseware is easy to use and easy to understand. The ability to stage such courseware on the Web does have the potential benefit that it can be widely used, reducing the cost per user.

Nonetheless, by reducing the cost of developing such courseware, its development would certainly be encouraged. When we considered what tool to use to develop our courseware, we considered both available general purpose algorithm animators and multimedia authoring tools. Unfortunately, what we needed was a tool that combined aspects of both. Currently, no such tool exists, which is why we were forced to use a very low level tool, a general purpose programming language, Java. Using low level tools is more costly than higher level tools. Consequently, we believe that developing high level tools that provided the essential elements of both would encourage the development of more courseware of the kind we developed, but at lower cost.

We would recommend that this process be begun by modifying an existing Web-based algorithm animation system to incorporate the *I'll Try* feature, together with the ability to keep track of a student's progress. In addition, the ability to modify the interface presented to the user to

include specific buttons that activate specific operations associated with a data structure, such as the insert and delete operations, would also be essential.

### 6.2.4 Guiding interactivity

Although our study did not find that interactivity as we implemented it improved student performance, our results suggest that had we implemented interactivity in a more guided way—a way that discouraged students from entering the interactive mode before they understood the material—they may have gained more benefit from it. For children learning the syntax of the programming language Logo, Heller [HEL86] has found a structured environment to be more beneficial than an exploratory environment. Possible ways that interactivity might be guided would be to require students to watch a particular algorithm for a prescribed number of times before they were permitted to try it. Another alternative would be to employ some metric—perhaps the ratio of incorrect to correct responses—that observed whether students appeared to be guessing. When students seemed to be guessing, we might offer further explanation or either encourage or require students to return to the observation mode. Further empirical studies would be needed to determine whether such changes would alter the value of interactivity.

## 6.3 Contributions to the field

We conclude with a brief discussion of the contributions that we believe our research has made to the field. Although a relatively recent development, the World Wide Web has already been recognized as an important component in the future of education. Because it is so new, the role it should play in education is not fully understood. It undoubtedly will be involved in a variety of different ways. Our research has demonstrated both the benefits and the remaining problems of implementing Web-based courseware. The feedback gathered from students during the two year period that it was used, has demonstrated that such courseware is both well accepted and perceived to be beneficial. Our experience has, however, indicated that some problems remain using courseware written in Java. Although these problems have lessened somewhat over this period, differences in browsers and machines remain a serious concern.

A key component of our research was the courseware that we developed. At the time we began our research, there were relatively few algorithm animation sites on the Web. In the past several years, the number of such sites has definitely increased. Our courseware remains one of the few sites that provides a comprehensive collection of lessons designed specifically for a CS 2 course that contains animations, textual explanations and code, and the only site we have found that provides the kind of interactivity offered by our *I'll Try* feature. In a recent article in the SIGCSE Bulletin, Renee McCauley [MCC98] discusses the proliferation of such Web sites. Our courseware is among the few that she highlights.

The design methodology that we used in our research differed from many of the previous studies on CAI and algorithm animation. Rather than comparing the use of our courseware to learning the subject matter in a classroom, we instead compared two versions of the courseware to isolate the benefits of a single characteristic—interactivity. Also, we conducted two experiments, each with a different design. Threats that were present in one design were eliminated in the other. Finally, most of the previous algorithm animation studies used a single animation, but we used a collection of animations over a several week period.

A final major contribution of our research is demonstrating both the importance and the potential limitations of merging some fundamental principles of CAI with algorithm animation. The *Show Me* and *I'll Try* modes are common features in many CAI applications as is the ability of a student to view a progress record. Although many algorithm animation systems have been created in the past decade, none have incorporated these features. Our research has demonstrated that students value this combination and it encourages students to spend more time using the courseware. What our research did not demonstrate, however, is that these interactive features improve student learning. Instead, we learned that if offered too freely it may engage students in nonproductive activity. Our research offers a cautionary note to future designers of interactive courseware to ensure the interactivity they incorporate enhances rather than distracts from the primarily goal—student learning.

# CHAPTER 7

# REFERENCES


[ALE98] Aleem, Abdul, "A Taxonomony of Multimedia Interactivity", Doctoral Dissertation, The Union Institute, 1998.

[ALL92] Allinson, Lesley, "Learning Styles and Computer-Based Learning Environments", Computer Assisted Learning, *ICCAL '92 Proceedings*, Berlin, Germany, Springer-Verlag, 1992, pp. 61-73.

[BAD91] Badre, A., Beranek, M., Morris, J., and Stasko, J., "Assessing Program Visualization Systems as Instructional Aids", Computer Assisted Learning, *ICCAL '92 Proceedings*, Springer-Verlag, Berlin, Germany, 1991, pp. 87-99.

[BAE81] Baecker, R., "Sorting Out Sorting", *SIGGRAPH '81*, Los Altos, CA, 1981.

[BAR93] Barron, A., and Kysilka, M., "The Effectiveness of Digital Audio in Computer-Based Training", *Journal of Research on Computing in Education*, vol. 25, no. 30, Spring 1993, pp. 277-289.

[BIL92] Billings, D., and Cobb, K., "Effects of Learning Style Preferences, Attitude and GPA on Learner Achievement Using Computer Assisted Interactive Videodisc Instruction", *Journal of Computer-Based Instruction*, vol. 19, no. 1, Winter 1992, pp. 12-16.

[BIS94] Bishop-Clark, C., and Wheeler, D., "The Myers-Briggs Personality Type and Its Relationship to Computer Programming", *Journal of Research on Computing in Education*, vol. 26, no. 3, Spring 1994, pp. 358-370.

[BLA89] Blattner, M., and Greenberg, R., "Communicating and Learning Through Non-speech Audio", in *Multimedia Interface Design in Education*, edited by A. Edwards and S. Holland, Springer-Verlag, 1989, pp. 133-143.

[BON88] Bonham, L. Addrianne, "Learning style instruments: Let the buyer beware", *Lifelong learning*, vol. 11, no. 6, 1998, pp. 12-16.

[BOS86] Bosco, James, "An Analysis of Evaluations of Interactive Video", *Educational Technology*, no. 5, May 1986, pp. 7-17.

[BOS90] Bostrom, R., Olfman, L., and Sein, M., "The Importance of Learning Style in End-User Training", *MIS Quarterly*, vol. 14, no. 1, March 1990, pp. 101-119.

[BOS97] Bostock, Stephen, "Designing Web-Based Instruction for Active Learning", in *Web-Based Instruction*, edited by Badrul Khan, Educational Technology Publications, Englewood Cliffs, N. J., 1997, pp. 225-229.

[BRO88a] Brown, Marc, *Algorithm Animation*, MIT Press, Cambridge, Massachusetts, 1988.

[BRO88b] Brown, Marc, "Exploring Algorithms Using Balsa-II", *Computer*, vol. 21, no. 5, May 1988, pp. 14-36.

[BRO89] Brown, M., Newsome, S. and Glinert, E. "An Experiment into the Use of Auditory Cues to Reduce Visual Workload", *CHI '89 Proceedings*, May 1989, pp. 339-346.

[BRO91] Brown, Marc, "Zeus: A System for Algorithm Animation and Multi-View Editing", *Proceedings of the 1991 IEEE Workshop on Visual Languages*, Kobe, Japan, 1991, pp. 4-9.

[BRO92] Brown M., and Hershberger, J., "Color and Sound in Algorithm Animation", *Computer*, vol. 25, no. 12, December 1992, pp. 52-63.

[BRO96] Brown, M., and Najork, M., "Collaborative Active Textbooks: A Web-Based Algorithm Animation System for an Electronic Classroom", *Proceedings of the 1996 Symposium on Visual Languages*, Boulder, Colorado, September 1996, pp. 266-275.

[BRO97a] Brown, M., Najork M., and Raisamo, R., "A Java-Based Implementation of Collaborative Active Textbooks", *Proceedings of the 1997 Symposium on Visual Languages*, Isle of Capri, Italy, September 1997, pp. 372-379.

[BRO97b] Brooks, David, Web-Teaching, *A Guide to Designing Interactive Teaching for the World Wide Web*, New York, New York, Plenum Press, 1997.

[BYR96a] Byrne, Christine, "Water on Tap: The Use of Virtual Reality as an Educational Tool", 1996 <www.hitl.washington.edu/publications/dissertations/Byrne>.

[BYR96b] Byrne, M., Catrambone, R. and Stasko, J., "Do Algorithms Aid Learning?", Georgia Institute of Technology Technical Report GIT-GVU-96-18, August 1996.

[CAR91] Carlson, Helen, "Learning Style and Program Design in Interactive Multimedia", *Educational Technology, Research and Development*, vol. 39, no. 3, 1991, pp. 41-48.

[CAR96] Carlson, D., Guzdial, M., Kehoe, C., Shah, V. and Stasko, J., "WWW Interactive Learning Environments for Computer Science Education", *SIGCSE Bulletin*, vol. 28, no. 1, March 1996, pp. 290-294.

[CAV89] Cavaiani, Thomas, "Cognitive Style and Diagnostic Skills of Student Programmers", *Journal of Research on Computing in Education*, vol. 21, no. 4, Summer 1989, pp. 411-420.

[CLA83a] Clark, Richard, "Media Will Never Influence Learning", *Educational Technology Research and Development*, vol. 42, no. 2, 1983, pp. 21-29.

[CLA83b] Clark, Richard, "Reconsidering Research on Learning from Media", *Review of Educational Research*, vol. 53, no. 4, Winter 1983, pp. 445-459.

[CLA85] Clark, Richard, "Evidence for Confounding in Computer-Based Instruction Studies: Analyzing the Meta-Analyses", *Educational Communication and Technology Journal*, vol. 33, no. 4, Winter 1985, pp. 249-262.

[CLA87] Claxton, C., and Murrell, P., *Learning Styles: Implications for Improving Educational Practices*, Association for the Study of Higher Education, Reston, Virginia, 1987.

[CLA91] Clark, R., and Craig, T., "Research and Theory on Multi-Media Learning Effects", in *Interactive Multimedia Learning Environments*, edited by Max Giardina. Springer-Verlag, Berlin, Germany, 1991, pp. 19-30.

[CLA94] Clark, Richard, "Media Will Never Influence Learning", *Educational Technology Research and Development*, vol. 42, no. 2, 1994, pp. 21-29.

[COR86] Corman, Larry, "Cognitive Style, Personality Type, and Learning Ability as Factors in Predicting the Success of the Beginning Programming Student", *SIGCSE Bulletin*, vol. 18, no. 4, December 1986, pp. 80-89.

[COR90] Cormen, T., Leiserson, C. and Riverst, R., *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[COR91] Cordell, Barbara, "A Study of Learning Styles and Computer-Assisted Instruction", *Computers and Education*, vol. 16, no. 2, 1991, pp. 175-83.

[COX92] Cox, K. and Roman, G., "Abstraction in algorithm animation", *1992 IEEE Workshop on Visual Languages,* Seattle, Washington, September 1992, pp. 18-24.

[CRO95] Crosby, M., and Stelovsky, J., "From Multimedia Instruction to Multimedia Evaluation", *Journal of Educational Multimedia and Hypermedia*, vol. 4, no. 2/3, 1995, pp. 147-162.

[CRO97] Cronin, M., and Myers, S., "The Effects of Visuals Versus No Visuals on Learning Outcomes from Interactive Multimedia Instruction", *Journal of Computing in Higher Education*, vol. 8, no. 2, Spring 1997, pp. 46-71.

[CUR83] Curry, Lynn, "An Organization of Learning Styles Theory and Constructs", *Annual Meeting of the American Educational Research Association*, Montreal, Quebec, April 1983, ERIC Document 235 185.

[DAV92] Davidson, G., Savenye, W., and Orr, K., "How Do Learning Styles Relate to Performance in a Computer Application Course?", *Journal of Research on Computing in Education*, vol. 24, no. 3, Spring 1992, pp. 348-358.

[DER98] Dershem, H., Brummund, P., "Tools for Web-Based Sorting Animation", *SIGCSE Bulletin*, vol. 30, no. 1, March 1998, pp. 222-226.

[DIE95] Dierker, Robert, "The Future of Electronic Education", in *The Electronic Classroom: A Handbook for Education in the Electronic Environment*, edited by Erwin Boschmann, Learned Information Inc., Medford, New Jersey, 1995, pp. 228-235.

[DUI90] Duisberg, Robert, "Visual Programming of Program Visualizations, A Gestural Interface for Animating Algorithms", in *Visual Languages and Applications*, edited by T. Ichikawa, E. Jungert and R. Korfhage, New York, New, York, Plenum Press, 1990, pp. 161-173.

[EAD89] Eades, P., and Xuemin, L., "How to Draw a Directed Graph", *1989 IEEE Workshop on Visual Languages*, Rome, Italy, October, 1989, pp. 13-17.

[ELL93] Ellis, A. and Fouts, J., *Research on Educational Innovations*, Eye on Education, Princeton Junction, New Jersey, 1993, pp. 59-71.

[ENT81] Entwistle, Noel, *Styles of Learning and Teaching, an Integrated Outline of Educational Psychology for Students, Teachers and Lecturers*, John Wiley and Sons, Chichester, England, 1981.

[EST94] Ester, Don, "CAI, Lecture, and Student Learning Style: The Differential Effects of Instructional Method", *Journal of Research on Computing in Education*, vol. 27, no. 2, Winter 1994-95, pp. 129-140.

[FAR97] Faraday, P., and Sutcliffe, A., "Designing Effective Multimedia Presentations", Proceedings of the 1997 *ACM SIGCHI Conference on Human Factors in Computing Systems*, Atlanta, Georgia, March 1997, pp. 272-278.

[FEL88] Felder, R., and Silverman, L., "Learning and Teaching Styles in Engineering Education", *Engineering Education*, vol. 78, no. 7, April 1988, pp. 674-681.

[FEL89] Feldman, M. and Moran, M., "Validating a Demonstration Tool for Graphics-Assisted Debugging of Ada Concurrent Programs", *IEEE Transactions on Software Engineering*, vol. 15, no. 3, 1989, pp. 305–313.

[FEL96] Felder, Richard, "Matters of Style", *ASEE Prism*, vol. 6, no. 4, December 1996, pp. 18-23.

[FEL97] Feldman, Michael, *Software Construction and Data Structures with Ada 95*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1997.

[FLE89] Fletcher, J., "The Effectiveness and Cost of Interactive Videodisc Instruction", *Machine-Mediated Learning*, vol. 3, 1989, pp. 361-385.

[FLE95] Fletcher-Flinn, C., and Gravatt, B., "The Efficacy of Computed-Assisted Instruction (CAI): A Meta-Analysis", *Journal of Educational Computing Research*, vol. 12, no. 3, 1995, pp. 219-241.

[GAV90] Gaver, W., and Smith, R., "Auditory Icons in Large-Scale Collaborative Environments", *Proceedings of Human-Computer Interaction—Interact '90*, Cambridge, United Kingdom, August 1990, pp. 735-740.

[GAV93] Gaver, William, "Synthesizing Auditory Icons," *Proceedings of INTERCHI '93*, Amsterdam, Netherlands, April 1993, pp. 228-235.

[GIL97] Gillani, B., and Relan, A., "Incorporating Interactivity and Multimedia into Web-Based Instruction", in *Web-Based Instruction*, edited by Badrul Khan, Educational Technology Publications, Englewood Cliffs, New Jersey, 1997, pp. 231-237

[GLA93] Glassman, Steven, "A Turbo Environment for Producing Algorithm Animations", *Proceedings of the 1993 Symposium on Visual Languages*, Bergen, Norway, August 1993, pp. 32-36.

[GLE96] Glennan, T., and Melmed, A., *Fostering the Use of Educational Technology: Elements of a National Strategy*, Santa Monica, California, Rand Corporation, 1996.

[GLO92] Gloor, P., "AACE: Algorithm animation for computer science education", *Proceedings of the 1992 Workshop on Visual Languages*, Seattle, Washington, September 1992, pp. 25-31.

[GOR90] Gordon, Harold, "Cognitive Asymmetry, Computer Science Students, and Professional Programmers", *Journal of Educational Computing Research*, vol. 6, no. 2, 1990, pp. 135-146.

[GUR96] Gurka, J., and Citrin, W., "Testing Effectiveness of Algorithm Animation", *Proceedings of the 1996 Symposium on Visual Languages*, Boulder, Colorado, September 1996, pp. 182-189.

[HAA97] Haajanen, J., Psesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., and Vanninen, P., "Animation of user algorithms on the Web", *Proceedings of the 1997 Symposium on Visual Languages*, Isle of Capri, Italy, September 1997, pp. 360-367.

[HAR96] Hartley, Stephen, "A Hypermedia lab manual for Operating Systems: using the network to teach", *Proceedings of the Conference on Integrating Technology into Computer Science Education* , Barcelona, Spain, June 1996, pp. 8-10.

[HEL86] Heller, Rachelle, "Different LOGO Teaching Styles, Do They Really Matter?", *Empirical Studies of Programmers*, June, 1986, pp 117-127.

[HEL90a] Heller, Rachelle, "The Role of Hypermedia in Education: A Look at the Research Issues", *Journal of Research on Computing in Education*, vol. 22, no. 4, Summer 1990, pp. 431-441.

[HEL90b] Helttula, E., Hyrskykari, A., and Raiha, K., "Principles of Aladdin and Other Algorithm Animation Systems", in *Visual Languages and Applications*, edited by T. Ichikawa, E. Jungert and R. Korfhage, New York, New York, Plenum Press, 1990, pp. 175-187.

[HEL91] Heller, Rachelle, "Evaluating Software: A Review of the Options", *Computers Education*, vol. 17, no. 4, 1991, pp. 285-291.

[HEL95] Heller, R., and Martin, C., "A Media Taxonomy", *IEEE Multimedia*, Winter 1995, pp. 36-45 .

[HIC95] Hickcox, Leslie, "Learning Styles: A Survey of Adult Learning Style Inventory Models", in *The Importance of Learning Styles: Understanding the Implications for Learning, Course Design, and Education*, edited by R. Sims and S. Sims, Westport, Connecticut, Greenwood Press, 1995, pp. 25-47.

[HOW96] Howard, R., Carver, C., and Lane, W., "Felders's Learning Styles, Bloom's Taxonomy, and the Kolb Learning Cycle: Tying It All Together in the CS 2 Course", *SIGCSE Bulletin*, vol. 28, no. 1, March 1996, pp. 227-231.

[HYR87] Hyrskykari, A., and Raiha, K.-J.,. "Animation of algorithms without programming", *1987 Workshop on Visual Languages*, Linkoping, Sweden, 1987, pp. 40-54.

[KAN97] Kann, C., Lindeman, R., and Heller, R., "Integrating Algorithm Animation into a Learning Environment", *Computers Education* , vol. 28, no. 4, 1997, pp. 233-8.

[KEE88] Keefe, James, *Profiling & Utilizing Learning Style*, Reston, Virginia, National Association of Secondary School Principals, 1988, pp. 1-13.

[KEH96] Kehoe, C., and Stasko, J., "Using Animations to Learn about Algorithms: An Ethnographic Case Study", Georgia Institute of Technology Technical Report GIT-GVU-96-20, September 1996.

[KEH99] Kehoe, C., Stasko, J., and Taylor, A. "Rethinking the Evaluation of Algorithm Animation as Learning Aids: An Observational Study", Georgia Institute of Technology Technical Report GIT-GVU-99-10, March 1999.

[KHA94] Khalili, A., and Shashaani, L., "The Effectiveness of Computer Applications", *Journal of Research on Computing in Education*, vol. 27, no. 1, Fall 1994, pp. 48-61.

[KOZ94a] Kozma, Robert, "Will Media Influence Learning? Reframing the Debate", *Educational Technology Research and Development*, vol. 42, no. 2, 1994, pp. 7-19.

[KOZ94b] Kozma, Robert, "A reply: Media and Methods", *Educational Technology Research and Development*, vol. 42, no. 3, 1994, pp. 11-14.

[KUL82] Kulik, C., Schwalb, B., and Kulik, J., "Programmed Instruction in Secondary Education: A Meta-Analysis of Evaluation Findings", *Journal of Educational Research*, vol. 75, no. 3, January/February 1982, pp. 133-138.

[KUL86] Kulik, C., and Kulik, J., "Effectiveness of Computer-Based Education in Colleges", *AEDS Journal*, Winter/Spring 1986, pp. 81-108.

[KUL91] Kulik, C. and Kulik, J., "Effectiveness of Computer-Based Instruction: An Updated Analysis", *Computers in Human Behavior*, vol. 7, no. 1-2, 1991, pp. 75-94.

[LAR92] Larsen, Ronald, "Relationship of Learning Styles to the Effectiveness and Acceptance of Interactive Video Instruction", *Journal of Computer-Based Instruction*, vol. 19, no. 1, Winter 1992, pp. 17-21.

[LAW94] Lawrence, A., Badre, A., and Stasko, J., "Empirically Evaluating the Use of Animations to Teach Algorithms", *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, Missouri, October 1994, pp. 48–54.

[LIA92] Liao, Yuen-Kuang, "Effects of Computer-Assisted Instruction on Cognitive Outcomes: A Meta-Analysis", *Journal of Research on Computing in Education*, vol. 24, no. 3, Spring 1992, pp. 367-380.

[MAR87] Marshall, Jon, "Examination of a Learning Style Topology", *Research in Higher Education*, vol. 26, no. 4, 1987, pp. 417-429.

[MAR96] Marshall, A., and Hurley, S., "Interactive multimedia courseware for the World Wide Web", *Proceedings of the Conference on Integrating Technology into Computer Science Education*, Barcelona, Spain, June 1996, pp. 1-5.

[MAY91] Mayer, R., and Anderson, R., "Animations Need Narrations: An Experimental Test of a Dual Coding Hypothesis", *Journal of Educational Psychology*, vol. 83, no. 4, 1991, pp. 484–490.

[MAY92] Mayer, R., and Anderson, R., "The Instructive Animation: Helping Students Build Connections Between Words and Pictures in Multimedia Learning", *Journal of Educational Psychology*, vol. 84, no. 4, 1992, pp. 444-452.

[MAY97] Mayer, Richard, "Multimedia Learning: Are We Asking the Right Questions?", *Educational Psychologist*, vol. 32, no. 1, May, 1997, pp. 1-19.

[MCC98] McCauley, Renée, "Warning: Instructional Animation Tools Abound on the Web", *SIGCSE Bulletin*, vol. 30, no. 4, December, 1998, pp. 19a-20a.

[MCN89] McNeil, Barbara, "A metanalysis of interactive video instruction: A ten-year review of achievement effects", *Dissertation Abstracts International*, vol. 50, no. 6, 1989, pp. 1636.

[MCW96] McWhirter, J., "AlgorithmExplorer: A student-centered algorithm animation system", *Proceedings of the 1996 Symposium on Visual Languages*, Boulder, Colorado, September 1996, pp. 174-181.

[MEY93] Meyers, C., and Jones, T., *Promoting Active Learning*, Jossey-Bass Inc., San Francisco, California, 1993.

[MIC96] Michail, A., "Teaching binary tree algorithms through visual programming", *Proceedings of the 1996 Symposium on Visual Languages*, Boulder, Colorado, September 1996, pp. 38-45.

[MIN95] Minghim, R. and Forrest, A., "An Illustrated Analysis of Sonification for Scientific Visualization", *Proceedings of '95 Visualization*, IEEE Computer Society Press, pp. 110-117.

[MON95] Montgomery, Susan, "Addressing Diverse Learning Styles Through Multimedia", *Proceedings of ASEE/IEEE Frontiers in Education '95 Conference*, <fre.www.ecn.purdue.edu/FrE/asee/fie95/3a2/3a22/3a22.html>.

[MOR94] Morrison, Gary, "The Media Effects Question: 'Unresolvable' or Asking the Right Question", *Educational Technology Research and Development*, vol. 42, no. 2, 1994, pp. 40-44.

[MOR] Morris, J., Bernek, P., Stasko, J., Lawrence, A., and Badre, A. "User Preferences and Performance for Traditional Representations of Algorithm Animation", unpublished report.

[NAJ96] Najjar, Lawrence, "Multimedia Information and Learning", *Journal of Educational Multimedia and Hypermedia*, vol. 5, no. 2, 1996, pp. 129-150.

[NAP96a] Naps, Thomas, Chair, Working Group on Visualization, "An overview of visualization: it use and design", *Proceedings of the Conference on Integrating Technology into Computer Science Education*, Barcelona, Spain, June 1996, pp. 192-200.

[NAP96b] Naps, Thomas, "Algorithm visualization served off the World Wide Web: why and how", *Proceedings of the Conference on Integrating Technology into Computer Science Education*, Barcelona, Spain, June 1996, pp. 66-71.

[NAP98] Naps, T., and Bressler, E., “A multi-windowed environment for simultaneous visualization of related algorithms on the World Wide Web”, *SIGCSE Bulletin*, vol. 30, no. 1, March 1998, pp. 277-281.

[NEB97] Nebesh, Bodan, “Using the World Wide Web for Experimentation in Reusable Component Comprehension”, Doctoral Dissertation, The George Washington University, May, 1997.

[PAN96] Pane, J., Corbett, A. and John, B., “Assessing Dynamics in Computer-Based Instruction”, *Proceedings of the 1996 ACM CHI Conference*, Vancouver, British Columbia, April 1996, <www.acm.org/sigchi/chi96/proceedings/papers/Pane/jfp_txt.htm>.

[PIE98] Pierson, W., and Rodger, S., “Web-based Animation of Data Structures Using JAWAA”, *SIGCSE Bulletin*, vol. 30, no. 1, March 1998, pp. 267-271.

[POL92] Pollard, Constance, “Effects of Interactive Videodisc Instruction on Learner Performance, Learner Attitude and Learning Time”, *Journal of Instructional Psychology*, vol. 19, no. 3, 1992, pp. 189-196.

[PRI94] Price, B., Baecker, R., and Small, I., “A Principled Taxonomy of Software Visualization”, *Journal of Visual Languages and Computing*, vol. 4, no. 3, 1994, pp. 211-266.

[PUR97] Purchase, Helen, “Which aesthetic has the greatest effect on human understanding?”, in *Proceedings of the Graph Drawing Symposium*, edited by G. DiBattista, Berlin, Germany, Springer-Verlag, 1997.

[REE97] Reeves, T., and Reeves, P., “Effective Dimensions of Interactive Learning on the World Wide Web”, in *Web-Based Instruction*, edited by Badrul Khan, Educational Technology Publications, Englewood Cliffs, New Jersey, 1997, pp. 59-66.

[REI94] Reiser, Robert, “Clark's Invitation to the Dance: An Instructional Designer's Response”, *Educational Technology Research and Development*, vol. 42, no. 2, 1994, pp. 45-48.

[RIE90a] Rieber, Lloyd, “Animation in Computer-Based Instruction”, *Educational Technology Research and Development*, vol. 38, no. 1, 1990, pp. 77-86.

[RIE90b] Rieber, L., Boyce, M., and Assad, C., “The Effects of Computer Animation on Adult Learning and Retrieval Tasks”, *Journal of Computer-Based Instruction*, vol. 17, no. 2, Spring 1990, pp. 46-52.

[ROM92a] Roman, G.-C., and Cox, K., “Program Visualization: The Art of Mapping Programs to Pictures”, *International Conference on Software Engineering*, Melbourne, Australia, May 1992, pp. 412-420.

[ROM92b] Roman, G.-C., Cox, K., Wilcox, C., and Plun, J., “Pavane: a System for Declarative Visualization of Concurrent Computation”, *Journal of Visual Languages and Computing*, vol. 3, no. 2, 1992, pp. 161-193.

[ROM93] Roman, G.-C., and Cox, K., “A Taxonomy of Program Visualization Systems”, Computer, vol. 26, no. 12, December 1993, pp. 11-23.

[SOL98] Soloway, Elliot, "No One Is Making Money from Educational Software", *Communications of the ACM*, vol. 41, no. 2, February 1998, pp. 11-15.

[SCA96] Scaife, M., and Rogers, Y., "External cognition: how do graphical representations work?", *International Journal of Human-Computer Studies*, vol. 45, no. 2, August 1996, pp. 185-213.

[STA90a] Stasko, John, "Tango: A Framework and System for Algorithm Animation", *Computer*, vol. 23, no. 9, September 1990, pp. 27-39.

[STA90b] Stasko, John, "Simplifying Algorithm Animation with TANGO", *1990 Workshop on Visual Languages*, 1990, pp 1-6.

[STA91] Stasko, John, "Using Direct Manipulation to Build Algorithm Animations by Demonstration", *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, New Orleans, Louisiana, May 1991, pp. 307-314.

[STA93] Stasko, J., Badre, A., and Lewis, C., "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis", *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems*, Amsterdam, Netherlands, April 1993, pp. 61–66.

[STA96] Stasko, J., and Muthukumarasamy, J., "Visualizing Program Executions on Large Data Sets", *Proceedings of the 1996 Symposium on Visual Languages*, Boulder, Colorado, September 1996, pp. 166-173.

[STA97] Stasko, John, "Using Student-Built Algorithm Animations as Learning Aids", *SIGCSE Bulletin*, vol. 29, no. 1, March 1997, pp. 25-29.

[TAK94] Takahashi, S., Misyashita, K., Matsuoka, S., and Yonezawa, A., "A Framework for Constructing Animations via Declarative Mapping Rules", *Proceedings of the 1994 IEEE Symposium on Visual Languages*, St. Louis, Missouri, October 1994, pp. 314–322.

[TJA98] Tjaden, Bunny, "Do Lab Modules in CS Actually Help Students? An Empirical Study", *SIGCSE Bulletin*, vol. 30, no. 1, March 1998, pp. 282-286.

[TUF97] Tufte, Edward, *Visual Explanations, Images and Quantities, Evidence and Narrative*, Graphics Press, Cheshire, Connecticut, 1997.

[TYM98] Tyma, Paul, "Why are we using Java Again?", *Communications of the ACM*, vol. 41, no. 6, June 1998, pp. 34-37.

[WES92] West, Thomas, "Visual Thinkers, Mental Models and Computer Visualization", in *Interactive Learning through Visualization*, edited by S. Cunningham and R. Hubbold, Springer-Verlag, Berlin, Germany, 1992, pp. 93-102.

[WU98] Wu, C., Dale, N., and Bethel, L., "Conceptual Models and Cognitive Learning Styles in Teaching Recursion", *SIGCSE Bulletin*, vol. 30, no. 1, March 1998, pp. 292-296.

[YIL93] Yildez, R., and Atkins, M., "Evaluating Multimedia Applications", *Computers Education*, vol. 21, no. 1-2, 1993, pp. 133-139.

# APPENDIX A SOURCE CODE

## A.1 Java applet code

```
import java.awt.*;
import java.applet.*;
import java.lang.*;
import java.util.*;
import java.net.*;
import java.io.*;

// Classes that define the applet and the applet window

class Screen extends Frame
    {
    public static final int CANVAS_WIDTH = 546;
    public static final int CANVAS_HEIGHT = 270;
    public static final int BUTTON_WIDTH = 72;
    public static final int BUTTON_HEIGHT = 24;
    public static final int TIME_OUT = 50;
    public static final int DELAY = 500;

    private static final int INSET = 3;
    private static final int FRAME_WIDTH = 650;
    private static final int FRAME_HEIGHT = 396;

    private static final int TRAVERSAL = 1;
    private static final int HEAP_PRIORITY_QUEUE = 2;
    private static final int HEAP_SORT = 3;
    private static final int BINARY_SEARCH_TREE = 4;
    private static final int HEIGHT_BALANCING = 5;
    private static final int TREE_CATEGORIES = 6;
    private static final int ADJACENCY_MATRIX = 7;
    private static final int SEARCH = 8;
    private static final int GRAPH_CATEGORIES = 9;
    private static final int QUAD_SORTS = 10;
    private static final int EFF_SORTS = 11;

    private static final Color TREES_BACKGROUND_COLOR = new Color(0, 0, 128);
    private static final Color TREES_CANVAS_COLOR = new Color(192, 192, 255);
    private static final Color GRAPH_BACKGROUND_COLOR = new Color(128, 0, 0);
    private static final Color GRAPH_CANVAS_COLOR = Color.lightGray;
    private static final Color SORT_BACKGROUND_COLOR = new Color(0, 64, 0);
    private static final Color SORT_CANVAS_COLOR = new Color(255, 255, 192);

    public static Screen screen;
    public static AnimatorThread thread;
    public static PromptPanel prompt;
    public static AnswerPanel answer;
    public static Interaction interaction;
    public static DrawingCanvas canvas;
    public static ScreenPanel panel;
    public static Controls controls;
    public static ProgressPanel progress;
    public static int panelNo;

    private long startTime;
    private int helpCount = 0, progressCount = 0;
    private String name, startDate;
```

```
    private String helpHiddenHelp = "The help feature is on. Moving the mouse over any of the
controls will display a help window such as this one. Clicking this button again will turn
off this help.";
    private Button helpButton = new Button("Help");
    private SizedComponent sizedHelpButton = new SizedComponent
        (helpButton, BUTTON_WIDTH, BUTTON_HEIGHT, helpHiddenHelp,
         "Northeast");
    private boolean helpOn = true;

    private String progressHiddenHelp = "Displays the progress window which shows the number
of times you have watched an operation and the number of correct and incorrect responses.
Completed assignments are submitted from the progress window.";
    private String progressShowingHelp = "Closes the progress window.";
    private String noProgressHelp = "The progress feature is not available in this version.";
    private Button progressButton = new Button("Progress");
    private SizedComponent sizedProgressButton;
    private ProgressFrame frame;
    private boolean progressShowing = false;

    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    public Screen(String title, int panelNo, String name, boolean allowTry,
        boolean showProgress)
        {
        super(title);
        Color backgroundColor;
        this.screen = this;
        this.panelNo = panelNo;
        this.name = name;
        Date date = new Date();
        startDate = date.toLocaleString();
        startTime = date.getTime();
        setFont(new Font("Dialog", Font.PLAIN, 12));
        setLayout(gridBag);
        resize(FRAME_WIDTH, FRAME_HEIGHT);
        thread = new AnimatorThread();
        thread.start();
        String progressHelp;
        if (showProgress)
            progressHelp = progressHiddenHelp;
        else
            progressHelp = noProgressHelp;
        sizedProgressButton = new SizedComponent(progressButton,
            BUTTON_WIDTH, BUTTON_HEIGHT, progressHelp, "Southeast");

        if (panelNo <= TREE_CATEGORIES)
            {
            backgroundColor = TREES_BACKGROUND_COLOR;
            canvas = new DrawingCanvas(TREES_CANVAS_COLOR);
            }
        else if (panelNo <= GRAPH_CATEGORIES)
            {
            backgroundColor = GRAPH_BACKGROUND_COLOR;
            canvas = new DrawingCanvas(GRAPH_CANVAS_COLOR);
            }
        else
            {
            backgroundColor = SORT_BACKGROUND_COLOR;
            canvas = new DrawingCanvas(SORT_CANVAS_COLOR);
            }
        setBackground(backgroundColor);
        controls = new Controls();
        switch (panelNo)
            {
            case TRAVERSAL:
```

```
    String traversalButtons[] =  {"Make Tree", "Preorder",
        "Inorder", "Postorder", "Level Order"};
    progress = new ProgressPanel(traversalButtons, 1, name, 25);
    panel = new TraversalPanel(traversalButtons, 1,
        allowTry, backgroundColor);
    break;
case HEAP_PRIORITY_QUEUE:
    String pqHeapButtons[] =  {"Enqueue", "Dequeue"};
    progress = new ProgressPanel(pqHeapButtons, 0, name, 25);
    panel = new PQHeapPanel(pqHeapButtons, 0,
        allowTry, backgroundColor);
    break;
case HEAP_SORT:
    String heapSortButtons[] =  {"Make Tree", "Sort"};
    progress = new ProgressPanel(heapSortButtons, 1, name, 25);
    panel = new HeapSortPanel(heapSortButtons, 1,
        allowTry, backgroundColor);
    break;
case BINARY_SEARCH_TREE:
    String bsTreeButtons[] = {"Insert", "Delete", "Find"};
    progress = new ProgressPanel(bsTreeButtons, 0, name, 15);
    panel = new BSTreePanel(bsTreeButtons, 0,
        allowTry, backgroundColor);
    break;
case HEIGHT_BALANCING:
    String hbTreeButtons[] =  {"Make Tree", "Heights",
        "Bal Factors", "Balanced?", "Is HB-n?"};
    progress = new ProgressPanel(hbTreeButtons, 1, name, 10);
    panel = new HBTreePanel(hbTreeButtons, 1,
        allowTry, backgroundColor);
    break;
case TREE_CATEGORIES:
    String treesButtons[] =  {"Make Tree", "Is Almost?",
        "Is Complete", "Is BS Tree?", "Is Heap?", "Is AVL?"};
    progress = new ProgressPanel(treesButtons, 1, name, 10);
    panel = new TreesPanel(treesButtons, 1,
        allowTry, backgroundColor);
    break;
case ADJACENCY_MATRIX:
    String matrixButtons[] =  {"Make One", "Create Other"};
    progress = new ProgressPanel(matrixButtons, 1, name, 25);
    panel = new MatrixPanel(matrixButtons, 1,
        allowTry, backgroundColor);
    break;
case SEARCH:
    String searchButtons[] =  {"Make Graph", "Depth First",
        "Breadth First"};
    progress = new ProgressPanel(searchButtons, 1, name, 25);
    panel = new SearchPanel(searchButtons, 1,
        allowTry, backgroundColor);
    break;
case GRAPH_CATEGORIES:
    String graphsButtons[] =  {"Make Graph", "Is Connected?",
        "Has Cycles?", "Is Tree?"};
    progress = new ProgressPanel(graphsButtons, 1, name, 10);
    panel = new GraphsPanel(graphsButtons, 1,
        allowTry, backgroundColor);
    break;
case QUAD_SORTS:
    String quadSortButtons[] =  {"Make Array", "Bubble",
        "Insertion", "Selection"};
    progress = new ProgressPanel(quadSortButtons, 1, name, 25);
    panel = new QuadSortPanel(quadSortButtons, 1,
        allowTry, backgroundColor);
    break;
case EFF_SORTS:
```

```
            String effSortButtons[] =  {"Make Array", "Merge",
                "Quick"};
            progress = new ProgressPanel(effSortButtons, 1, name, 25);
            panel = new EffSortPanel(effSortButtons, 1,
                allowTry, backgroundColor);
            break;
        default:
            return;
        }

    constraints.gridx = 0;
    constraints.gridy = 0;
    constraints.insets = new Insets(INSET, INSET, INSET, INSET);
    prompt = new PromptPanel();
    answer = new AnswerPanel();
    interaction = new Interaction(prompt, answer);
    gridBag.setConstraints(interaction, constraints);
    add(interaction);

    constraints.gridx = 1;
    gridBag.setConstraints(sizedHelpButton, constraints);
    add(sizedHelpButton);

    constraints.gridx = 0;
    constraints.gridy = 1;
    gridBag.setConstraints(canvas, constraints);
    add(canvas);
    canvas.show();

    constraints.gridx = 1;
    gridBag.setConstraints(panel, constraints);
    add(panel);
    panel.show();

    constraints.gridx = 0;
    constraints.gridy = 2;
    gridBag.setConstraints(controls, constraints);
    add(controls);
    controls.show();

    constraints.gridx = 1;
    gridBag.setConstraints(sizedProgressButton, constraints);
    add(sizedProgressButton);
    if (!showProgress)
        progressButton.disable();
    frame = new ProgressFrame(progress);
    }
public Dimension minimumSize()
    {return new Dimension(FRAME_WIDTH, FRAME_HEIGHT);}
public boolean handleEvent(Event event)
    {
    if (event.target == helpButton && event.id == Event.ACTION_EVENT)
        {
        helpOn = !helpOn;
        if (helpOn)
            {
            sizedHelpButton.showHelp();
            Screen.interaction.displayMessage("");
            }
        else
            Screen.interaction.displayMessage("Help Is Now Turned Off");
        helpCount++;
        return true;
        }
    else if (event.target == progressButton && event.id == Event.ACTION_EVENT)
        {
```

```
            clickProgress();
            return true;
            }
        else if (event.id == Event.WINDOW_DESTROY)
            {
            Screen.interaction.displayMessage("Use Stop Visualization Button");
            return true;
            }
        return false;
        }
    public boolean isHelpOn() {return helpOn;}
    public void clickProgress()
        {
        if (progressShowing)
            {
            frame.hide();
            progressShowing = false;
            sizedProgressButton.changeMessage(progressHiddenHelp);
            }
        else
            {
            frame.show();
            progressShowing = true;
            sizedProgressButton.changeMessage(progressShowingHelp);
            progressCount++;
            }
        }
    public void closeProgress()
        {
        if (progressShowing)
            frame.hide();
        }
    public void sendLog()
        {
        Socket socket;
        DataOutputStream out;
        String eventLog = name.replace(' ','_') + "_";
        long elapsedTime = ((new Date()).getTime() - startTime) / 1000;
        eventLog += startDate.replace(' ','_') + "_";
        eventLog += elapsedTime + "_";
        eventLog += "L:" + panelNo + "_";
        eventLog += progress.getLog();
        eventLog += panel.getLog();
        eventLog += controls.getLog();
        eventLog += "H:" + helpCount + "_P:" + progressCount;
        try
            {
            socket = new Socket("www.seas.gwu.edu", 80);
            out = new DataOutputStream(socket.getOutputStream());
            out.writeBytes("GET /htbin/jarc/log?" + eventLog + "\n\n");
            out.close();
            socket.close();
            }
        catch(Exception e) {}
        }
    }

public class Animator extends Applet
    {
    public static final int CANVAS_WIDTH = 546;
    public static final int CANVAS_INSET = 8;

    private Screen screen;

    public void start()
        {
```

```
        addNotify();
        super.init();
        String name = getParameter("studentName");
        int panelNo = Integer.parseInt(getParameter("panelNo"));
        boolean allowTry = (Boolean.valueOf(getParameter("allowTry"))).booleanValue();
        boolean showProgress = (Boolean.valueOf(getParameter("showProgress")))
            .booleanValue();
        String titles[] = {"Binary Tree Traversals", "Priority Queue with Heap",
            "Heap Sort", "Binary Search Trees", "Height Balanced Trees",
            "Categorizing Binary Trees", "Graph Representation",
            "Graph Searches", "Categorizing Graphs", "Quadratic Sorts",
            "Efficent Sorts"};
        screen = new Screen(titles[panelNo - 1], panelNo, name, allowTry,
            showProgress);
        screen.show();
        }
    public void stop()
        {
        super.stop();
        screen.sendLog();
        screen.hide();
        screen.closeProgress();
        }
    }

//Classes that define the help panel

class HelpPanel extends Panel
    {
    private static final int INSET = 4;
    private static final int OFFSET = 4;
    private static final int WIDTH = 150;

    private String alignment, helpMessage;
    private FontMetrics fontMetrics;
    private DrawableTextBox helpTextBox;

    public HelpPanel(String helpMessage, String alignment)
        {
        this.alignment = alignment;
        this.helpMessage = helpMessage;
        }
    public void changeMessage(String helpMessage)
        {
        Screen.canvas.removeHelp(helpTextBox);
        this.helpMessage = helpMessage;
        helpTextBox = null;
        }
    public boolean mouseEnter(Event event, int x, int y)
        {
        showHelp();
        return true;
        }
    public boolean mouseExit(Event event, int x, int y)
        {
        Screen.canvas.removeHelp(helpTextBox);
        return true;
        }
    public void showHelp()
        {
        if (helpTextBox == null)
            helpTextBox = makeTextBox(helpMessage);
        Screen.canvas.addHelp(helpTextBox);
        }
    private DrawableTextBox makeTextBox(String helpMessage)
        {
```

```
    int height, xPos, yPos;
    String lines[];
    Font font = Screen.screen.getFont();
    fontMetrics = Screen.screen.getToolkit().getFontMetrics(font);
    int noLines = splitLines(helpMessage, null);
    lines = new String[noLines];
    splitLines(helpMessage, lines);
    height = fontMetrics.getHeight() * noLines;
    helpTextBox = new DrawableTextBox(height, WIDTH);
    helpTextBox.setText(lines);
    int boxWidth = WIDTH + INSET * 2, boxHeight = height + INSET * 2;
    int xMax = Screen.CANVAS_WIDTH - boxWidth - OFFSET;
    int yMax = Screen.CANVAS_HEIGHT - boxHeight - OFFSET;
    xPos = xMax; yPos = yMax;
    if (alignment.equals("North") || alignment.equals("Northeast"))
        yPos = OFFSET;
    if (alignment.equals("East"))
        {
        yPos = location().y + size().height / 2 - boxHeight / 2;
        yPos = Math.max(OFFSET, yPos);
        yPos = Math.min(yMax, yPos);
        }
    else if (alignment.equals("North") || alignment.equals("South"))
        {
        xPos = location().x + size().width / 2 - boxWidth / 2;
        xPos = Math.max(OFFSET, xPos);
        xPos = Math.min(xMax, xPos);
        }
    helpTextBox.reposition(new Position(xPos, yPos));
    return helpTextBox;
    }
private int splitLines(String text, String lines[])
    {
    int toIndex = 0, fromIndex = 0, lastIndex = 0, lineNo = 0;
    String line, lastLine;

    while (fromIndex != -1)
        {
        while (fromIndex < text.length() && text.charAt(fromIndex) == ' ')
            fromIndex++;
        toIndex = fromIndex;
        line = null;
        do
            {
            lastLine = line;
            lastIndex = toIndex;
            toIndex = text.indexOf(' ', toIndex);
            if (toIndex == -1)
                line = text.substring(fromIndex);
            else
                line = text.substring(fromIndex, toIndex++);
            }
         while (fontMetrics.stringWidth(line) < WIDTH && toIndex != -1);
         if (fontMetrics.stringWidth(line) > WIDTH && lastLine != null)
             {
             line = lastLine;
             fromIndex = lastIndex;
             }
         else
             fromIndex = toIndex;
         if (lines != null)
             lines[lineNo] = line;
         lineNo++;
         }
    return lineNo;
    }
```

```
    }

class SizedComponent extends HelpPanel
    {
    private int width, height;

    public SizedComponent(Component component, int width, int height,
        String helpMessage, String alignment)
        {
        super(helpMessage, alignment);
        setLayout(new BorderLayout());
        add("Center", component);
        setBackground(Color.lightGray);
        this.width = width;
        this.height = height;
        }
    public Dimension minimumSize()
        {return new Dimension(width, height);}
    }

class SizedComponentPair extends HelpPanel
    {
    private static final int COMPONENT_1_Y = 0;
    private static final int COMPONENT_2_Y = 20;
    private static final int INSET = 2;

    private int width, height;

    public SizedComponentPair(Component component1, Component component2,
        int width, int height, String helpMessage, String alignment)
        {
        super(helpMessage, alignment);
        this.width = width;
        this.height = height;
        if (alignment.equals("East"))
            {
            setLayout(null);
            component1.reshape(INSET, COMPONENT_1_Y, width, height / 2);
            component2.reshape(INSET, COMPONENT_2_Y, width, height / 2);
            add(component1);
            add(component2);
            }
        else
            {
            setLayout(new BorderLayout());
            add("West", component1);
            add("East", component2);
            }
        setBackground(Color.lightGray);
        }
    public Dimension minimumSize()
        {return new Dimension(width, height);}
    }

// Classes that define the button panel

class AnimatorThread extends Thread
    {
    private boolean clicked = false;
    private int buttonNumber;

    public int button() {return buttonNumber;}
    synchronized public void buttonClicked(int buttonNumber)
        {
        if (clicked)
            try {wait();} catch (InterruptedException e) {}
```

```
        this.buttonNumber = buttonNumber;
        clicked = true;
        notifyAll();
        }
    synchronized public void run()
        {
        while (true)
            {
            if (!clicked)
                try {wait();} catch (InterruptedException e) {}
            Screen.answer.clearAnswer();
            Screen.panel.disableButtons();
            Screen.controls.start();
            Screen.panel.buttonClicked(buttonNumber);
            if (Screen.controls.isAborted())
                Screen.interaction.displayMessage("");
            Screen.controls.stop();
            Screen.panel.enableButtons();
            Screen.canvas.repaint();
            clicked = false;
            notifyAll();
            }
        }
    }

abstract class ScreenPanel extends Panel
    {
    private protected static final int RADIO_HEIGHT = 40;
    private static final int COMPUTER = 1;
    private static final int USER = 2;

    private protected Random random = new Random();
    private protected boolean shownOrTried[], running = false;
    private protected SizedComponent buttonPanels[];
    private protected String showMessages[], tryMessages[];
    private protected String runningMessage = "This button is inactive while an animation is
running.";

    private Button buttons[];
    private int who = COMPUTER, where = COMPUTER;
    private int first, numberOfButtons;
    private int showButtonNo, tryButtonNo, randomButtonNo, pickButtonNo,
        totalButtons;
    private int buttonClicks[];
    private CheckboxGroup Group1 = new CheckboxGroup();
    private Checkbox showMe = new Checkbox("Show Me", Group1, true);
    private Checkbox illTry = new Checkbox("I'll Try", Group1, false);
    private String showTryHelp = "In the Show Me mode, you will be shown the correct
solution.  In the I'll Try mode you will be asked to provide the correct responses.";
    private SizedComponentPair showTryGroup = new SizedComponentPair(showMe,
        illTry, Screen.BUTTON_WIDTH, RADIO_HEIGHT, showTryHelp, "East");
    private CheckboxGroup Group2 = new CheckboxGroup();
    private Checkbox randomPick = new Checkbox("Random", Group2, true);
    private Checkbox illPick = new Checkbox("I'll Pick", Group2, false);
    private String randomPickHelp;
    private SizedComponentPair pickGroup;

    private int gridy = 0;
    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    abstract public void buttonClicked(int number);
    public Dimension minimumSize()
        {return new Dimension(Screen.BUTTON_WIDTH, Screen.CANVAS_HEIGHT);}
    public void init(String buttonNames[], String showMessages[],
        String tryMessages[], int first, boolean allowTry,
```

```
    String randomOrPick, boolean pickOnly, Color backgroundColor)
    {
    Screen.panel = this;
    setLayout(gridBag);
    setBackground(backgroundColor);
    this.first = first;
    this.showMessages = showMessages;
    this.tryMessages = tryMessages;
    randomPickHelp = randomOrPick;
    numberOfButtons = buttonNames.length;
    totalButtons = numberOfButtons;
    buttons = new Button[numberOfButtons];
    buttonPanels = new SizedComponent[numberOfButtons];
    shownOrTried = new boolean[numberOfButtons];
    for (int button = 0; button < numberOfButtons; button++)
        shownOrTried[button] = false;
    constraints.weighty = 1;
    if (randomOrPick != null || pickOnly)
        constraints.gridy = gridy++;
    for (int number = 0; number < numberOfButtons; number++)
        {
        buttons[number] = new Button(buttonNames[number]);
        buttonPanels[number] = addButton(buttons[number],
            showMessages[number]);
        }
    if (allowTry)
        {
        addPanel(showTryGroup);
        showButtonNo = totalButtons++;
        tryButtonNo = totalButtons++;
        }
    if (randomOrPick != null)
        {
        pickGroup = new SizedComponentPair(randomPick, illPick,
            Screen.BUTTON_WIDTH, RADIO_HEIGHT, randomOrPick, "East");
        addPanel(pickGroup);
        randomButtonNo = totalButtons++;
        pickButtonNo = totalButtons++;
        }
    buttonClicks = new int[totalButtons];
    for (int button = 0; button < totalButtons; button++)
        buttonClicks[button] = 0;
    }
synchronized public boolean handleEvent(Event event)
    {
    for (int number = 0; number < numberOfButtons; number++)
        if (event.target == buttons[number] && event.id == Event.ACTION_EVENT)
            {
            if (number == 0 && first == 1)
                for (int button = 0; button < numberOfButtons; button++)
                    shownOrTried[button] = false;
            else
                shownOrTried[number] = true;
            Screen.thread.buttonClicked(number);
            buttonClicks[number]++;
            return true;
            }
    if (event.target == showMe && event.id == Event.ACTION_EVENT)
        {
        who = COMPUTER;
        Screen.canvas.showMe();
        changeMessages();
        Screen.controls.changeMessages();
        enableButtons();
        buttonClicks[showButtonNo]++;
        return true;
```

```
        }
    if (event.target == illTry && event.id == Event.ACTION_EVENT)
        {
        who = USER;
        changeMessages();
        Screen.controls.changeMessages();
        enableButtons();
        buttonClicks[tryButtonNo]++;
        return true;
        }
    if (event.target == randomPick && event.id == Event.ACTION_EVENT)
        {
        where = COMPUTER;
        buttonClicks[randomButtonNo]++;
        return true;
        }
    if (event.target == illPick && event.id == Event.ACTION_EVENT)
        {
        where = USER;
        buttonClicks[pickButtonNo]++;
        return true;
        }
    return false;
    }
public String getLog()
    {
    String log = "B:";
    for (int button = 0; button < totalButtons; button++)
        log += buttonClicks[button] + "_";
    return log;
    }
public boolean isInteractive()
    {return who == USER;}
public boolean userPicks()
    {return where == USER;}
public void enableButtons()
    {
    running = false;
    changeMessages();
    for (int number = 0; number < numberOfButtons; number++)
        if (who == USER && shownOrTried[number] && number != 0 && first != 0)
            buttons[number].disable();
        else
            buttons[number].enable();
    illTry.enable();
    showMe.enable();
    illPick.enable();
    randomPick.enable();
    enableSpecials();
    }
public void enableSpecials() {}
public void disableButtons()
    {
    running = true;
    changeMessages();
    for (int number = 0; number < numberOfButtons; number++)
        buttons[number].disable();
    illTry.disable();
    showMe.disable();
    illPick.disable();
    randomPick.disable();
    disableSpecials();
    }
public void disableSpecials() {}
public void changeMessagesSpecials() {}
public int getRandom(int lower, int upper)
```

```
        {
        return Math.abs(random.nextInt() % (upper - lower))
            + lower;
        }

    private protected void addPanel(Panel panel)
        {
        constraints.gridy = gridy++;
        gridBag.setConstraints(panel, constraints);
        add(panel);
        }
    private protected void changeMessages()
        {
        for (int button = 0; button < numberOfButtons; button++)
            if (running)
                buttonPanels[button].changeMessage(runningMessage);
            else
                if (isInteractive())
                    if (shownOrTried[button])
                        buttonPanels[button].changeMessage("This button is inactive because
the I'll Try mode is selected and this operation has already been shown or tried.  To
reactivate it, switch to the Show Me mode or make a new data set.");
                    else
                        buttonPanels[button].changeMessage(tryMessages[button]);
                else
                    buttonPanels[button].changeMessage(showMessages[button]);
        if (running)
            showTryGroup.changeMessage(runningMessage);
        else
            showTryGroup.changeMessage(showTryHelp);
        if (pickGroup != null)
            if (running)
                pickGroup.changeMessage(runningMessage);
            else
                pickGroup.changeMessage(randomPickHelp);
        changeMessagesSpecials();
        }
    private SizedComponent addButton(Button button, String helpMessage)
        {
        SizedComponent sizedButton = new SizedComponent(button, Screen.BUTTON_WIDTH,
            Screen.BUTTON_HEIGHT, helpMessage, "East");
        constraints.gridy = gridy++;
        gridBag.setConstraints(sizedButton, constraints);
        add(sizedButton);
        return sizedButton;
        }
    }

// Classes that define the animation canvas

interface Clickable
    {
    abstract boolean inside(Position position);
    abstract void show();
    }

interface Selectable
    {
    abstract int select(Position position);
    }

class Position
    {
    private int positionX, positionY;

    public Position(int x, int y)
```

```
        {positionX = x;  positionY = y;}
    public int x() {return positionX;}
    public int y() {return positionY;}
    }

class Move
    {
    private int leg, legs;
    private Position from, to;
    private double xIncrement, yIncrement;

    public Move(Position from, Position to)
        {
        this.from = from;
        this.to = to;
        double deltaX = to.x() - from.x();
        double deltaY = to.y() - from.y();
        double distance = Math.sqrt(deltaX * deltaX + deltaY * deltaY);
        distance = Math.max(1.0, distance);
        if (Screen.controls.isAborted())
            legs = 0;
        else
            legs = (int) Math.round(Math.sqrt(distance)) /
                Screen.controls.getSpeed();
        if (legs > 0)
            {
            xIncrement = deltaX / legs;
            yIncrement = deltaY / legs;
            }
        leg = 1;
        }
    public boolean moreLegs() {return legs > 0;}
    public void nextMove(Drawable object)
        {
        int x = (int)Math.round((double) from.x() + xIncrement * leg);
        int y = (int)Math.round((double) from.y() + yIncrement * leg);
        leg++;
        if (--legs <= 0)
            {
            x = to.x();
            y = to.y();
            }
        object.reposition(new Position(x, y));
        }
    }

abstract class Drawable implements Clickable
    {
    private protected int x, y;
    private protected Color color;

    private boolean visible = false;
    private boolean moving = false;
    private int numberOfMoves, move;
    private Move moves[];

    abstract public void draw(Graphics g);
    public void setColor(Color color) {this.color = color;}
    synchronized public boolean drawObject(Graphics g)
        {
        Position position;
        Screen.canvas.repaint();
        if (moving && visible)
            {
            if (moves[move].moreLegs())
                moves[move].nextMove(this);
```

```
            else if (++move < numberOfMoves)
                moves[move].nextMove(this);
            else
                moving = false;
            draw(g);
            }
        else if (moving)
            moving = false;
        else if (visible)
            draw(g);
        return moving;
        }
    public void show() {visible = true;}
    public void hide() {visible = false;}
    synchronized public void reposition(Position position)
        {x = position.x(); y = position.y();}
    synchronized public void moveTo(Position[] moveList)
        {
        numberOfMoves = moveList.length;
        moves = new Move[numberOfMoves];
        Position from = new Position(x, y);
        for (move = 0; move < numberOfMoves; move++)
            {
            Position to = moveList[move];
            moves[move] = new Move(from, to);
            from = to;
            }
        move = 0;
        moving = true;
        }
    public boolean inside(Position position) {return false;}
    }

class DrawingCanvas extends Canvas
    {
    private static final int MAX_TRIES = 5;
    private static final int HELP_TIME_OUT = 200;

    private DrawableTextBox helpBox;
    private int helpCount = 0;
    private Vector drawableObjects = new Vector();
    private Vector clickableObjects;
    private Vector selectableObjects;
    private Vector highlighted = new Vector();
    private Enumeration objectList;
    private Image canvasImage;
    private Graphics canvasGraphics;
    private int tries = 1;
    private int selection;
    private boolean clickAll;
    private boolean awaitButton = false;
    private boolean pressed = false;
    private boolean moving = false;
    private boolean answerShown = false;

    public DrawingCanvas(Color canvasColor)
        {
        resize(Screen.CANVAS_WIDTH, Screen.CANVAS_HEIGHT);
        setFont(new Font("Dialog", Font.PLAIN, 12));
        setBackground(canvasColor);
        }
    public void awaitMovingCompletion()
        {
        if (Screen.controls.isAborted()) return;
        synchronized (this) {moving = true;}
        while (moving)
```

```
        {
        repaint();
        try {Thread.sleep(Screen.DELAY);}
        catch (InterruptedException e) {}
        }
    }
synchronized public void acceptClick(Clickable object)
    {
    if (clickableObjects == null)
        clickableObjects = new Vector();
    clickableObjects.addElement(object);
    }
synchronized public void buttonClicked(boolean correct)
    {
    if (correct)
        awaitButton = false;
    else
        invalidClick();
    }
public boolean answerWasShown() {return answerShown;}
public boolean awaitClick(String awaitMessage, boolean clickAll,
    boolean buttonCorrect)
    {
    this.clickAll = clickAll;
    awaitButton = buttonCorrect;
    answerShown = false;
    if (!Screen.controls.isAborted())
        Screen.prompt.displayPrompt(awaitMessage);
    Screen.interaction.displayMessage("");
    synchronized(this) {tries = 1;}
    for (int timer = 0; !Screen.controls.isAborted(); timer++)
        {
        try {Thread.sleep(Screen.DELAY);} catch (InterruptedException e) {}
        synchronized (this)
            {
            if (clickableObjects == null && !awaitButton)
                {
                Screen.progress.oneMoreRight(Screen.thread.button());
                Screen.interaction.displayMessage("Very Good, That's Correct");
                Screen.prompt.displayPrompt("");
                try {Thread.sleep(Screen.DELAY);}
                catch (InterruptedException e) {}
                return true;
                }
            if (quitTrying(clickableObjects, timer))
                {
                clickableObjects = null;
                return false;
                }
            }
        }
    clickableObjects = null;
    return false;
    }
synchronized public void acceptSelection(Selectable object)
    {
    if (selectableObjects == null)
        selectableObjects = new Vector();
    selectableObjects.addElement(object);
    }
public int awaitSelection(String awaitMessage)
    {
    if (!Screen.controls.isAborted())
        Screen.prompt.displayPrompt(awaitMessage);
    Screen.interaction.displayMessage("");
    synchronized(this) {tries = 1;}
```

```
    for (int timer = 0; !Screen.controls.isAborted(); timer++)
        {
        try {Thread.sleep(Screen.DELAY);} catch (InterruptedException e) {}
        synchronized (this)
            {
            if (selectableObjects == null)
                return selection;
            if (quitTrying(selectableObjects, timer))
                {
                selectableObjects = null;
                return -1;
                }
            }
        }
    selectableObjects = null;
    return -1;
    }

public void add(Drawable object) {drawableObjects.addElement(object);}
public void remove(Drawable object) {drawableObjects.removeElement(object);}
public void addHelp(DrawableTextBox helpBox)
    {
    this.helpBox = helpBox;
    helpCount = HELP_TIME_OUT;
    }
public void removeHelp(DrawableTextBox helpBox)
    {
    if (this.helpBox == helpBox)
        helpCount = 0;
    }
synchronized public void paint(Graphics g)
    {
    if (Screen.controls.isAborted()) return;
    if (canvasImage == null)
        {
        canvasImage = createImage(Screen.CANVAS_WIDTH, Screen.CANVAS_HEIGHT);
        canvasGraphics = canvasImage.getGraphics();
        }
    canvasGraphics.setColor(this.getBackground());
    canvasGraphics.fillRect(0, 0, Screen.CANVAS_WIDTH, Screen.CANVAS_HEIGHT);
    canvasGraphics.setColor(this.getForeground());
    boolean movingObjects = false;
    objectList = drawableObjects.elements();
    while (objectList.hasMoreElements())
        {
        Drawable object = (Drawable) objectList.nextElement();
        movingObjects |= object.drawObject(canvasGraphics);
        }
    if (Screen.screen.isHelpOn() && helpCount > 0)
        {
        helpBox.draw(canvasGraphics);
        helpCount--;
        }
    g.drawImage(canvasImage, 0 , 0, null);
    moving = movingObjects;
    }
public void update(Graphics g)
    {
    paint(g);
    }

synchronized public void showMe() {tries = 5;}
public boolean mouseDown(Event e, int x, int y)
    {
    pressed = true;
    return true;
```

```
    }
synchronized public boolean mouseUp(Event e, int x, int y)
    {
    if (pressed)
        {
        pressed = false;
        checkClickables(x, y);
        checkSelectables(x, y);
        }
    return true;
    }
public void clear()
    {
    for (int index = 0; index < highlighted.size(); index++)
        {
        Drawable object = (Drawable) highlighted.elementAt(index);
        remove(object);
        }
    highlighted = new Vector();
    }
public void highlight(Drawable object, Color color, boolean checkWho)
    {
    object.setColor(color);
    if (checkWho && Screen.panel.isInteractive())
        acceptClick(object);
    else
        object.show();
    highlighted.addElement(object);
    add(object);
    }
private void checkClickables(int x, int y)
    {
    if (Screen.panel.isInteractive() && clickableObjects != null
        && !clickableObjects.isEmpty())
        {
        objectList = clickableObjects.elements();
        while (objectList.hasMoreElements())
            {
            Clickable object = (Clickable) objectList.nextElement();
            if (object.inside(new Position(x, y)))
                {
                object.show();
                clickableObjects.removeElement(object);
                if (clickAll && !clickableObjects.isEmpty())
                    Screen.prompt.displayPrompt("OK, Click The Other One");
                else
                    clickableObjects = null;
                return;
                }
            }
        invalidClick();
        }
    else if (Screen.panel.isInteractive() && awaitButton)
        invalidClick();
    }
private void checkSelectables(int x, int y)
    {
    if ((Screen.panel.userPicks() || Screen.panel.isInteractive()) &&
        selectableObjects != null && !selectableObjects.isEmpty())
        {
        objectList = selectableObjects.elements();
        while (objectList.hasMoreElements())
            {
            Selectable object = (Selectable) objectList.nextElement();
            if ((selection = object.select(new Position(x, y))) >= 0)
                {
```

```
                    selectableObjects = null;
                    return;
                    }
                }
            invalidClick();
            }
        }
    private void invalidClick()
        {
        Screen.progress.oneMoreWrong(Screen.thread.button());
        if (tries == 1)
            Screen.interaction.displayMessage("Not Correct, Try Again");
        else if (tries == 2)
            Screen.interaction.displayMessage("Still Wrong, Try Again");
        else if (tries == 3)
            Screen.interaction.displayMessage("OK, Try One More Time");
        tries++;
        }
    private boolean quitTrying(Vector objects, int timer)
        {
        if (tries >= MAX_TRIES)
            Screen.interaction.displayMessage("OK, Let Me Show You");
        else if (timer < Screen.TIME_OUT)
            return false;
        else
            Screen.interaction.displayMessage("Time's Up, I'll Show You");
        Screen.prompt.displayPrompt("");
        try {Thread.sleep(Screen.DELAY);}
        catch (InterruptedException e) {}
        answerShown = true;
        return true;
        }
    }

// Classes that define the control panel

class SpeedControl extends Scrollbar
    {
    private static final int MAXIMUM = 4;
    private static final int MINIMUM = 1;
    private static final int INITIAL = (MAXIMUM + MINIMUM) / 2;

    private int current = INITIAL;

    public SpeedControl()
        {super(Scrollbar.HORIZONTAL, INITIAL, 0, MINIMUM, MAXIMUM);}
    public boolean handleEvent(Event event)
        {
        current = ((Scrollbar) event.target).getValue();
        return false;
        }
    public int getSpeed()
        {
        return current;
        }
    }

class Controls extends Panel
    {
    private static final int NO_OF_CONTROLS = 6;
    private static final int RADIO_WIDTH = 165;
    private static final int SPEED_WIDTH = 80;
    private static final int BUTTON_WIDTH = 65;
    private static final int CONTROLS_HEIGHT = 20;

    private int gridx = 0;
```

```
    private int controlClicks[] = new int[NO_OF_CONTROLS];
    private boolean aborted = false, singleSteps = false, paused = false,
        resumeClicked = false, stepClicked = false, abortClicked = false,
        running = false, showingHelp = false;
    private Button pause, resume, step, abort;
    private CheckboxGroup group = new CheckboxGroup();
    private Checkbox singleStep = new Checkbox("Single Step", group, false);
    private Checkbox continuous = new Checkbox("Continuous", group, true);
    private String stepRunHelp = "These buttons determine whether the animation runs
continously or in single steps.  In the single step mode, the Step button to the left must be
clicked to move to the next step.";
    private String stepOffHelp = "This button is inactive because the Continuous mode is
selected.";
    private String pauseResumeOffHelp = "This button is inactive because the Single Step mode
is selected.";
    private String notRunningHelp = "This button is inactive because no animation is
running.";
    private SizedComponent stepPanel, pausePanel, resumePanel, abortPanel;
    private SizedComponentPair stepRunPanel = new SizedComponentPair
        (singleStep, continuous, RADIO_WIDTH, CONTROLS_HEIGHT, stepRunHelp,
        "South");
    private SpeedControl speedControl = new SpeedControl();
    private String speedHelp = "This scrollbar controls the speed of the animation, move it
left for slower, right for faster.";
    private SizedComponent speedPanel = new SizedComponent(speedControl, SPEED_WIDTH,
        CONTROLS_HEIGHT, speedHelp, "South");
    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    public Controls()
        {
        setBackground(Color.lightGray);
        setLayout(gridBag);
        constraints.weightx = 1;
        constraints.gridy = 0;
        step = new Button("Step");
        stepPanel = addButton(step, stepOffHelp);
        constraints.gridx = gridx++;
        gridBag.setConstraints(stepRunPanel, constraints);
        add(stepRunPanel);
        pause = new Button("Pause");
        pausePanel = addButton(pause, notRunningHelp);
        resume = new Button("Resume");
        resumePanel = addButton(resume, notRunningHelp);
        speedControl.setBackground(Color.gray);
        constraints.gridx = gridx++;
        gridBag.setConstraints(speedPanel, constraints);
        add(speedPanel);
        abort = new Button("Abort");
        abortPanel = addButton(abort, notRunningHelp);
        step.disable();
        pause.disable();
        resume.disable();
        abort.disable();
        for (int control = 0; control < NO_OF_CONTROLS; control++)
            controlClicks[control] = 0;
        }
    public Dimension minimumSize()
        {return new Dimension(Screen.CANVAS_WIDTH, Screen.BUTTON_HEIGHT);}
    public int getSpeed() {return speedControl.getSpeed();}
    public boolean isAborted() {return aborted;}
    public boolean abortInitiated() {return aborted || abortClicked;}
    public void start()
        {
        running = true;
        changeMessages();
```

```
    if (!Screen.panel.isInteractive())
        {
        if (!singleSteps)
            pause.enable();
        }
    singleStep.disable();
    continuous.disable();
    abort.enable();
    }
public void stop()
    {
    running = false;
    changeMessages();
    aborted = false;
    abortClicked = false;
    step.disable();
    pause.disable();
    resume.disable();
    abort.disable();
    singleStep.enable();
    continuous.enable();
    }
synchronized public void nextStep(boolean counts, boolean doStep)
    {
    if (aborted)
        return;
    if (counts && !Screen.panel.isInteractive())
        Screen.progress.oneMoreShown(Screen.thread.button());
    if (!doStep || Screen.panel.isInteractive())
        return;
    if (paused)
        {
        while (!resumeClicked)
            try {wait();} catch (InterruptedException e) {}
        resumeClicked = false;
        notify();
        }
    if (singleSteps)
        {
        step.enable();
        Screen.prompt.displayPrompt("Click Step For Next Step");
        while (!stepClicked)
            try {wait();} catch (InterruptedException e) {}
        stepClicked = false;
        notify();
        step.disable();
        Screen.prompt.displayPrompt("");
        }
    if (abortClicked)
        aborted = true;
    }
public boolean handleEvent(Event event)
    {
    if (event.target == step && event.id == Event.ACTION_EVENT)
        {
        clickStep();
        controlClicks[0]++;;
        return true;
        }
    else if (event.target == singleStep && event.id == Event.ACTION_EVENT)
        {
        singleSteps = true;
        changeMessages();
        controlClicks[1]++;
        return true;
        }
```

```
        else if (event.target == continuous && event.id == Event.ACTION_EVENT)
            {
            singleSteps = false;
            changeMessages();
            controlClicks[2]++;
            return true;
            }
        if (event.target == pause && event.id == Event.ACTION_EVENT)
            {
            pause.disable();
            resume.enable();
            paused = true;
            changeMessages();
            controlClicks[3]++;
            return true;
            }
        else if (event.target == resume && event.id == Event.ACTION_EVENT)
            {
            resume.disable();
            clickResume();
            changeMessages();
            controlClicks[4]++;
            return true;
            }
        else if (event.target == abort && event.id == Event.ACTION_EVENT)
            {
            abort.disable();
            Screen.prompt.displayPrompt("");
            if (Screen.panel.isInteractive())
                aborted = true;
            else
                {
                abortClicked = true;
                if (singleSteps)
                    clickStep();
                else if (paused)
                    {
                    resume.disable();
                    clickResume();
                    }
                }
            controlClicks[5]++;
            return true;
            }
        return false;
        }
    public String getLog()
        {
        String log = "C:";
        for (int control = 0; control < NO_OF_CONTROLS; control++)
            log += controlClicks[control] + "_";
        return log;
        }
    public void changeMessages()
        {
        if (singleSteps)
            {
            if (running)
                if (Screen.panel.isInteractive())
                    stepPanel.changeMessage("This button is inactive because the I'll Try
mode is selected.");
                else
                    stepPanel.changeMessage("Takes you the the next step of the animation.");
            else
                stepPanel.changeMessage(notRunningHelp);
            pausePanel.changeMessage(pauseResumeOffHelp);
```

```
            resumePanel.changeMessage(pauseResumeOffHelp);
            }
        else
            {
            stepPanel.changeMessage(stepOffHelp);
            if (running)
                if (paused)
                    {
                    pausePanel.changeMessage("This button is inactive because the animation
is already paused.  Click the Resume button to the right, to resume the animation.");
                    resumePanel.changeMessage("Resumes the animation.");
                    }
                else
                    {
                    pausePanel.changeMessage("Pauses the animation. The Resume button to the
right resumes it.");
                    resumePanel.changeMessage("This button is inactive because the animation
is not paused.  Click the Pause button to the left, to pause the animation.");
                    }
            else
                {
                pausePanel.changeMessage(notRunningHelp);
                resumePanel.changeMessage(notRunningHelp);
                }
            }
        if (running)
            {
            abortPanel.changeMessage("Terminates the animation.");
            stepRunPanel.changeMessage("These buttons are inactive while an animation is
running.");
            }
        else
            {
            abortPanel.changeMessage(notRunningHelp);
            stepRunPanel.changeMessage(stepRunHelp);
            }
        }
    private SizedComponent addButton(Button button, String helpMessage)
        {
        SizedComponent sizedButton = new SizedComponent(button, BUTTON_WIDTH,
            CONTROLS_HEIGHT, helpMessage, "South");
        constraints.gridx = gridx++;
        gridBag.setConstraints(sizedButton, constraints);
        add(sizedButton);
        return sizedButton;
        }
    synchronized private void clickResume()
        {
        while (resumeClicked)
            try {wait();} catch (InterruptedException e) {}
        paused = false;
        resumeClicked = true;
        notify();
        pause.enable();
        }
    synchronized private void clickStep()
        {
        while (stepClicked)
            try {wait();} catch (InterruptedException e) {}
        stepClicked = true;
        notify();
        }
    }

// Classes that define the progress window
```

```
class ProgressFrame extends Frame
    {
    private static final int INSET = 20;
    private static final int PADDING_X = 8;
    private static final int PADDING_Y = 5;

    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    public ProgressFrame(ProgressPanel panel)
        {
        super("Progress Window");
        setFont(Screen.screen.getFont());
        setLayout(gridBag);
        setBackground(Color.darkGray);
        constraints.insets = new Insets(INSET, INSET, INSET, INSET);
        constraints.ipadx = PADDING_X;
        constraints.ipady = PADDING_Y;
        gridBag.setConstraints(panel, constraints);
        add(panel);
        panel.show();
        resize(400, 300);
        }
    public boolean handleEvent(Event event)
        {
        if (event.id == Event.WINDOW_DESTROY)
            {
            Screen.screen.clickProgress();
            return true;
            }
        return false;
        }
    }

class ProgressPanel extends Panel
    {
    private static final int TEXT_WIDTH = 3;
    private static final int NAME_WIDTH = 20;
    private static final int MSG_WIDTH = 50;
    private static final int LABEL_X = 0;
    private static final int SHOWN_X = 1;
    private static final int RIGHT_X = 2;
    private static final int WRONG_X = 3;
    private static final int NEED_X = 4;
    private static final int INSET = 5;
    private static final int INSET_Y = 8;

    private int first, buttonCount, amountNeeded;
    private int shown[], right[], wrong[], need[];
    private TextField shownField[], rightField[], wrongField[], needField[];
    private TextField name, message;
    private Button submit;
    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    public ProgressPanel(String buttonNames[], int first, String studentName,
        int amountNeeded)
        {
        this.first = first;
        this.amountNeeded = amountNeeded;
        buttonCount = buttonNames.length - first;
        setLayout(gridBag);
        setBackground(Color.white);
        shownField = new TextField[buttonCount];
        rightField = new TextField[buttonCount];
        wrongField = new TextField[buttonCount];
```

```
    needField = new TextField[buttonCount];
    shown = new int[buttonCount];
    right = new int[buttonCount];
    wrong = new int[buttonCount];
    need = new int[buttonCount];
    constraints.insets = new Insets(INSET, INSET, INSET, INSET);
    addLabel("Shown", SHOWN_X, 0);
    addLabel("Right", RIGHT_X, 0);
    addLabel("Wrong", WRONG_X, 0);
    addLabel("Need", NEED_X, 0);
    for (int button = 0; button < buttonCount; button++)
        {
        shown[button] = 0;
        right[button] = 0;
        wrong[button] = 0;
        need[button] = amountNeeded;
        addLabel(buttonNames[button + first], LABEL_X, button + 1);
        shownField[button] = addTextField(TEXT_WIDTH, false, "" + shown[button],
            SHOWN_X, button + 1, 1);
        rightField[button] = addTextField(TEXT_WIDTH, false, "" + right[button],
            RIGHT_X, button + 1, 1);
        wrongField[button] = addTextField(TEXT_WIDTH, false, "" + wrong[button],
            WRONG_X, button + 1, 1);
        needField[button] = addTextField(TEXT_WIDTH, false, "" + need[button],
            NEED_X, button + 1, 1);
        }
    constraints.insets = new Insets(INSET_Y, INSET, INSET_Y, INSET);
    addLabel("Your Name:", LABEL_X, buttonCount + 1);
    name = addTextField(NAME_WIDTH, false, "", SHOWN_X, buttonCount + 1, 3);
    name.setText(studentName);
    submit = new Button("Submit");
    constraints.anchor = GridBagConstraints.WEST;
    constraints.gridx = NEED_X;
    constraints.gridy = buttonCount + 1;
    constraints.gridwidth = 2;
    gridBag.setConstraints(submit, constraints);
    add(submit);
    submit.disable();
    message = addTextField(MSG_WIDTH, false, "", LABEL_X, buttonCount + 2, 6);
    message.setText("When \"Need\" column is all zero you can submit your assignment.");
    }
public void oneMoreShown(int button)
    {
    button -= first;
    shown[button]++;
    shownField[button].setText("" + shown[button]);
    }
public void oneMoreRight(int button)
    {
    button -= first;
    right[button]++;
    rightField[button].setText("" + right[button]);
    if (need[button] > 0)
        {
        need[button]--;
        needField[button].setText("" + need[button]);
        }
    for (int buttonNo = 0; buttonNo < buttonCount; buttonNo++)
        if (need[buttonNo] > 0)
            return;
    submit.enable();
    }
public void oneMoreWrong(int button)
    {
    button -= first;
    wrong[button]++;
```

```
    need[button] += 2;
    wrongField[button].setText("" + wrong[button]);
    needField[button].setText("" + need[button]);
    }
public boolean handleEvent(Event event)
    {
    if (event.target == submit && event.id == Event.ACTION_EVENT)
        {
        Socket socket;
        DataOutputStream out;
        String studentName = name.getText();
        try
            {
            socket = new Socket("www.seas.gwu.edu", 80);
            out = new DataOutputStream(socket.getOutputStream());
            String data = Screen.panelNo + ":" +
                studentName.replace(' ', '_');
            out.writeBytes("GET /htbin/jarc/hws?" + data + "\n\n");
            }
        catch(UnknownHostException e)
            {
            message.setText("Unknown Server");
            return true;
            }
        catch(IOException e)
            {
            message.setText("IO Error");
            return true;
            }
        message.setText("Your assignment has been accepted");
        try
            {
            out.close();
            socket.close();
            }
        catch (Exception e) {}
        for (int button = 0; button < buttonCount; button++)
            {
            need[button] = amountNeeded;
            needField[button].setText("" + need[button]);
            }
        submit.disable();
        return true;
        }
    return false;
    }
public String getLog()
    {
    String log = "A:";
    for (int button = 0; button < buttonCount; button++)
        {
        log += shown[button] + "_";
        log += right[button] + "_";
        log += wrong[button] + "_";
        }
    return log;
    }
private TextField addTextField(int fieldWidth, boolean edit,
    String value, int x, int y, int width)
    {
    TextField field = new TextField(fieldWidth);
    field.setEditable(edit);
    field.setText(value);
    constraints.anchor = GridBagConstraints.EAST;
    constraints.gridwidth = width;
    constraints.gridx = x;
```

```
        constraints.gridy = y;
        gridBag.setConstraints(field, constraints);
        add(field);
        return field;
        }
    private void addLabel(String name, int x, int y)
        {
        Label label = new Label(name);
        constraints.anchor = GridBagConstraints.WEST;
        constraints.gridx = x;
        constraints.gridy = y;
        gridBag.setConstraints(label, constraints);
        add(label);
        }
    }

// Classes that define the interaction panel

class PromptPanel extends HelpPanel
    {
    private static final int MAX_TRIES = 3;

    private boolean entryComplete = false;
    private Label prompt = new Label("", Label.CENTER);
    private TextField pickText = new TextField("", 2);

    public PromptPanel()
        {
        super("Prompts are displayed in this area.", "North");
        setLayout(new BorderLayout());
        setBackground(Color.lightGray);
        add("Center",prompt);
        }
    public boolean handleEvent(Event event)
        {
        if (event.target == pickText)
            {
            char ch = (char)event.key;
            if (ch == '\n')
                {
                entryComplete = true;
                return true;
                }
            }
        return super.handleEvent(event);
        }
    public void displayPrompt(String string)
        {
        prompt.setText(string);
        }
    public int getData(int digits, int lower, int upper, boolean alwaysGet)
        {
        int pickedValue, tries = 1;
        boolean validPick = false;
        int value = (int) Math.floor((upper - lower + 1) * Math.random()
            + lower);
        if (!alwaysGet && !Screen.panel.userPicks())
            return value;
        else
            {
            entryComplete = false;
            pickText.setText("");
            add("East",pickText);
            layout();
            pickText.requestFocus();
```

```
            for (int timer = 0; timer < Screen.TIME_OUT && !Screen.controls.abortInitiated();
                timer++)
                {
                try {Thread.sleep(Screen.DELAY);} catch (InterruptedException e) {}
                String text = pickText.getText();
                if (text.length() >= digits || entryComplete)
                    {
                    try
                        {
                        pickedValue = Integer.parseInt(text);
                        if (pickedValue < lower)
                            Screen.interaction.displayMessage("Value Must >= " + lower);
                        else if (pickedValue > upper)
                            Screen.interaction.displayMessage("Value Must Be <= " + upper);
                        else
                            {
                            validPick = true;
                            value = pickedValue;
                            break;
                            }
                        }
                    catch (NumberFormatException e)
                        {
                        Screen.interaction.displayMessage("Value Must Be Numeric");
                        }
                    pickText.setText("");
                    if (tries++ == MAX_TRIES)
                        break;
                    }
                }
            remove(pickText);
            layout();
            displayPrompt("");
            if (!validPick)
                {
                Screen.interaction.displayMessage("OK, I'll Pick " + value);
                try {Thread.sleep(Screen.DELAY);}
                catch (InterruptedException e) {}
                }
            return value;
            }
        }
    public void awaitInput(String prompt, int correctAnswer)
        {
        int value, timer, tries = 1;
        if (Screen.controls.isAborted())
            return;
        displayPrompt(prompt);
        Screen.interaction.displayMessage("");
        pickText.setText("");
        add("East",pickText);
        layout();
        pickText.requestFocus();
        for (timer = 0; timer < Screen.TIME_OUT && !Screen.controls.isAborted(); timer++)
            {
            if (Screen.panel.userPicks())
                {Screen.interaction.displayMessage("OK, I'll Show You"); break;}
            try {Thread.sleep(Screen.DELAY);} catch (InterruptedException e) {}
            String input = pickText.getText();
            if (input.length() > 0 && !input.equals("-"))
                {
                pickText.setText("");
                try
                    {
                    value = Integer.parseInt(input);
                    if (value == correctAnswer)
```

```
                        {
                        Screen.interaction.displayMessage("Very Good, That's Correct");
                        Screen.progress.oneMoreRight(Screen.thread.button());
                        break;
                        }
                    }
                catch (NumberFormatException e) {}
                Screen.progress.oneMoreWrong(Screen.thread.button());
                if (tries == 1)
                    Screen.interaction.displayMessage("Not Correct, Try Again");
                else if (tries == 2)
                    Screen.interaction.displayMessage("Still Wrong, Try Again");
                else if (tries == 3)
                    Screen.interaction.displayMessage("OK, Try One More Time");
                else
                    {
                    Screen.interaction.displayMessage("OK, Let Me Show You");
                    break;
                    }
                tries++;
                pickText.setText("");
                }
            }
        if (timer >= Screen.TIME_OUT)
            Screen.interaction.displayMessage("Time's Up, I'll Show You");
        displayPrompt("");
        try {Thread.sleep(Screen.DELAY);} catch (InterruptedException e) {}
        remove(pickText);
        layout();
        }
    public Dimension minimumSize()
        {return new Dimension(Interaction.PANEL_WIDTH, Interaction.COMPONENT_HEIGHT);}
    }

class AnswerPanel extends HelpPanel
    {
    private int correctChoice = 0;
    private Label label = new Label("", Label.CENTER);
    private Button choice1 = new Button("");
    private Button choice2 = new Button("");

    public AnswerPanel()
        {
        super("Answers are displayed in this area.", "North");
        setLayout(new GridLayout(1, 2, 5, 0));
        setBackground(Color.lightGray);
        add(label);
        }
    public boolean handleEvent(Event event)
        {
        if (event.target == choice1 && event.id == Event.ACTION_EVENT)
            {
            Screen.canvas.buttonClicked(correctChoice == 1);
            return true;
            }
        else if (event.target == choice2 && event.id == Event.ACTION_EVENT)
            {
            Screen.canvas.buttonClicked(correctChoice == 2);
            return true;
            }
        return super.handleEvent(event);
        }
    public void clearAnswer() {label.setText("");}
    public boolean oneButton(boolean buttonCorrect, String prompt, String valid,
        String invalidShow, String invalidTry)
        {
```

```
    if (Screen.panel.isInteractive())
        {
        remove(label);
        Screen.prompt.displayPrompt(prompt);
        choice1.setLabel(valid);
        add(choice1);
        layout();
        if (buttonCorrect)
            correctChoice = 1;
        else
            correctChoice = 0;
        boolean success = Screen.canvas.awaitClick(prompt, false,
            buttonCorrect);
        remove(choice1);
        if (buttonCorrect)
            label.setText(valid);
        else
            if (Screen.canvas.answerWasShown() ||
                Screen.controls.isAborted())
                label.setText(invalidShow);
            else
                label.setText(invalidTry);
        add(label);
        Screen.prompt.displayPrompt("");
        return success;
        }
    else
        {
        Screen.controls.nextStep(true, false);
        if (buttonCorrect)
            label.setText(valid);
        else
            label.setText(invalidShow);
        return true;
        }
    }
public void twoButtons(String prompt, String label1, String label2,
    String message1, String message2, boolean choice1Correct,
    boolean choice2Correct)
    {
    if (Screen.panel.isInteractive())
        {
        remove(label);
        choice1.setLabel(label1);
        choice2.setLabel(label2);
        add(choice1);
        add(choice2);
        layout();
        if (choice1Correct)
            correctChoice = 1;
        else if (choice2Correct)
            correctChoice = 2;
        else
            correctChoice = 0;
        boolean success = Screen.canvas.awaitClick(prompt, false,
            correctChoice > 0);
        remove(choice1);
        remove(choice2);
        add(label);
        Screen.prompt.displayPrompt("");
        }
    if (choice1Correct)
        label.setText(message1);
    else if (choice2Correct)
        label.setText(message2);
    else
```

```
            label.setText("");
        Screen.controls.nextStep(true, false);
        }
    public Dimension minimumSize()
        {return new Dimension(Interaction.PANEL_WIDTH, Interaction.COMPONENT_HEIGHT);}
    }

class Interaction extends Panel
    {
    public static final int PANEL_WIDTH = 177;
    public static final int COMPONENT_HEIGHT = 20;

    private Label message = new Label("", Label.CENTER);
    private SizedComponent messagePanel;
    private GridBagLayout gridBag = new GridBagLayout();
    private GridBagConstraints constraints = new GridBagConstraints();

    public Interaction(PromptPanel prompt, AnswerPanel answer)
        {
        setBackground(Color.gray);
        setLayout(gridBag);
        constraints.weightx = 1;
        constraints.gridy = 0;
        constraints.gridx = 0;
        gridBag.setConstraints(prompt, constraints);
        add(prompt);
        constraints.gridx = 1;
        gridBag.setConstraints(answer, constraints);
        add(answer);
        constraints.gridx = 2;
        SizedComponent sizedMessage = new SizedComponent(message,
            PANEL_WIDTH,COMPONENT_HEIGHT, "Messages are displayed in this area.", "North");
        gridBag.setConstraints(sizedMessage, constraints);
        add(sizedMessage);
        }
    public Dimension minimumSize()
        {return new Dimension(Screen.CANVAS_WIDTH, Screen.BUTTON_HEIGHT);}

    public void displayMessage(String string)
        {
        message.setText(string);
        }
    }

// Classes that define drawable objects

class HiddenNode extends Drawable
    {
    private protected static final int RADIUS = 15;

    public HiddenNode() {}
    public HiddenNode(Position position, int level)
        {
        this.reposition(position);
        this.level = level;
        }
    public boolean inside(Position position)
        {
        int width = RADIUS, height = RADIUS;
        if (isHidden())
            {
            for (int i = 5; i > level; i--)
                width *= 2;
            height *= 2;
            }
        return (position.x() > x - width && position.x() < x + width &&
```

```
            position.y() > y - height && position.y() < y + height);
        }
    public void draw(Graphics g)
        {
        g.setColor(Color.yellow);
        g.fillOval(x - RADIUS, y - RADIUS, RADIUS * 2, RADIUS * 2);
        }
    public boolean isHidden() {return true;}

    private protected int level;
    }

abstract class DrawableNode extends HiddenNode
    {
    public static final int maxNodes = 32;
    private static final int treeFullness = 2;
    private static final Color treeColor1 = new Color(0, 0, 128);
    private static final Color treeColor2 = new Color(128, 0, 64);
    private static final int pause = 500;
    private static final int xCheck = 15;
    private static final int yCheck = 20;
    private static final int checkWidth = 10;
    private protected static final int xDiff = 275;
    private protected static final int yDiff = 50;
    private protected static final int xOffset = 7;
    private protected static final int yOffset = 5;
    private protected static final int xString = 18;
    private protected static final int yString = 25;
    private protected static final int maxLevels = 6;
    private protected static final int maxValue = 100;
    private protected static Random random = new Random();

    abstract public DrawableNode left();
    abstract public DrawableNode right();
    public boolean isHidden() {return nullNode;}
    public void draw(Graphics g)
        {
        if (nullNode) return;
        if (left() != null && !left().nullNode)
            {
            g.setColor(treeColor1);
            g.drawLine(x, y, left().x, left().y);
            g.setColor(nodeColor);
            if (!movingTreeNode)
                left().draw(g);
            }
        if (right() != null && !right().nullNode)
            {
            g.setColor(treeColor1);
            g.drawLine(x, y, right().x, right().y);
            g.setColor(nodeColor);
            if (!movingTreeNode)
                right().draw(g);
            }
        g.setColor(nodeColor);
        g.fillOval(x - RADIUS, y - RADIUS, RADIUS * 2, RADIUS * 2);
        g.setFont(Screen.screen.getFont());
        if (stringValueNode)
            {
            g.setColor(Color.black);
            g.drawString(getString(), x - xString, y + yString);
            }
        else if (value >= 0)
            {
            g.setColor(Color.white);
            g.drawString("" + value, x - xOffset, y + yOffset);
```

```
        }
    if (!Screen.panel.isInteractive())
        for (int check = 0; check < numberChecked; check++)
            {
            int x1 = x - xCheck + check * checkWidth, y1 = y + yCheck;
            int x2 = x1 + 2, y2 = y1 + 4;
            int x3 = x2 + 4, y3 = y2 - 8;
            g.setColor(Color.black);
            g.drawLine(x1, y1, x2, y2);
            g.drawLine(x2, y2, x3, y3);
            }
    }
public void changeColor()
    {
    if (nodeColor == treeColor1)
        nodeColor = treeColor2;
    else
        nodeColor = treeColor1;
    }
public void resetColor()
    {
    nodeColor = treeColor1;
    if (left() != null && !left().nullNode)
        left().resetColor();
    if (right() != null && !right().nullNode)
        right().resetColor();
    }
public void move(String prompt, boolean visitNode, DrawableCircle tracer)
    {
    if (!Screen.panel.isInteractive())
        {
        Position positions[] = new Position[1];
        positions[0] = new Position(x, y);
        tracer.moveTo(positions);
        Screen.canvas.awaitMovingCompletion();
        }
    tracer.reposition(new Position(x, y));
    if (visitNode)
        {
        if (Screen.panel.isInteractive())
            {
            Screen.canvas.acceptClick(this);
            Screen.canvas.awaitClick(prompt, true, false);
            }
        }
    }
public void moveValue(DrawableNode to, DrawableString nodeValue)
    {
    value = -1;
    Position positions[] = new Position[1];
    positions[0] = NodeStringPosition(to.x, to.y);
    nodeValue.moveTo(positions);
    nodeValue.show();
    }
public String getString() {return "";}
private protected Position NodeStringPosition(int x, int y)
    {return new Position(x - xOffset, y + yOffset);}
private protected Position leftPosition()
    {return new Position(x - xDiff / (int) Math.pow(2, level), y + yDiff);}
private protected Position rightPosition()
    {return new Position(x + xDiff / (int) Math.pow(2, level), y + yDiff);}
private protected boolean isNullNode()
    {
    return level == maxLevels || Math.abs(random.nextInt() % maxLevels)
        + treeFullness < level;
    }
```

```
    private protected int value;
    private protected int numberChecked = 0;
    private protected Color nodeColor = treeColor1;
    private protected boolean nullNode = false;
    private protected boolean movingTreeNode = false;
    private protected boolean stringValueNode = false;
    }

abstract class DrawableGraph extends Drawable implements Selectable
    {
    private static final int RADIUS = 3;
    private static final int SELECTION_RADIUS = 10;
    private protected static final Color GRAPH_COLOR = new Color(128, 0, 64);
    private protected static final Color TEXT_COLOR = new Color(0, 0, 64);
    private protected static final int X_NAME_OFFSET = 5;
    private protected static final int Y_NAME_OFFSET = 10;

    private protected int spacing;
    private protected int rows, cols, noOfNodes;
    private protected boolean named, directed = false;
    private protected boolean nodes[];
    private protected boolean edges[][];
    private protected String names[];
    private protected Random random = new Random();

    abstract public boolean isSelectable(int node);
    abstract public int xCoor(int node);
    abstract public int yCoor(int node);
    public DrawableGraph(Position position, int rows, int cols, boolean named,
        int spacing)
        {
        reposition(position);
        this.rows = rows;
        this.cols = cols;
        this.named = named;
        this.spacing = spacing;
        noOfNodes = rows * cols;
        nodes = new boolean[noOfNodes];
        edges = new boolean[noOfNodes][noOfNodes];
        names = new String[noOfNodes];
        }
    public void draw(Graphics g)
        {
        g.setColor(GRAPH_COLOR);
        for (int node = 0; node < noOfNodes; node++)
            if (nodes[node])
                g.fillOval(xCoor(node) - RADIUS, yCoor(node) - RADIUS,
                    RADIUS * 2, RADIUS * 2);
        for (int from = 0; from < noOfNodes; from++)
            for (int to = 0; to < noOfNodes; to++)
                if (directed || to < from)
                    if (edges[from][to] && showing(from, to))
                        {
                        g.drawLine(xCoor(from), yCoor(from),xCoor(to), yCoor(to));
                        if (directed)
                            drawArrow(g, from, to);
                        }
        g.setColor(TEXT_COLOR);
        g.setFont(Screen.screen.getFont());
        if (named)
            for (int node = 0; node < noOfNodes; node++)
                if (nodes[node])
                    g.drawString(names[node], xCoor(node) - X_NAME_OFFSET,
                        yCoor(node) - Y_NAME_OFFSET);
        }
```

```
    public int select(Position position)
        {
        for (int node = 0; node < noOfNodes; node++)
            if (isSelectable(node) &&
                position.x() > xCoor(node) - SELECTION_RADIUS &&
                position.x() < xCoor(node) + SELECTION_RADIUS &&
                position.y() > yCoor(node) - SELECTION_RADIUS &&
                position.y() < yCoor(node) + SELECTION_RADIUS)
                return node;
        return -1;
        }
    public boolean showing(int from, int to) {return true;};

    private protected int randomNo(int limit)
        {
        if (limit == 0) return 0;
        return Math.abs(random.nextInt()) % limit;
        }

    private void drawArrow(Graphics g, int from, int to) {}
    private int signOf(int value)
        {return Math.abs(value) / value;}
    }

class DrawableString extends Drawable
    {
    private static final int HEIGHT = 15;
    private static final int CHARACTER_WIDTH = 7;
    private static final int DELTA = 10;

    private String string;

    public DrawableString(Position position)
        {this.reposition(position); color = Color.black;}
    public boolean inside(Position position)
        {
        if (position.x() > x - DELTA && position.x() < x + string.length() *
            CHARACTER_WIDTH + DELTA && position.y() > y - HEIGHT - DELTA &&
            position.y() < y + DELTA)
            return true;
        return false;
        }
    public void setString(String string) {this.string = string;}
    public void draw(Graphics g)
        {
        g.setColor(color);
        g.setFont(Screen.screen.getFont());
        if (string != null)
            g.drawString(string, x, y);
        }
    }

class DrawableTextBox extends Drawable
    {
    private static final int INSET = 4;

    private int height, width;
    private String lines[];

    public DrawableTextBox(int height, int width)
        {
        this.height = height + INSET * 2;
        this.width = width + INSET * 2;
        }
    public void setText(String lines[]) {this.lines = lines;}
    public void draw(Graphics g)
```

```
        {
        g.setColor(Color.white);
        g.fillRect(x, y, width, height);
        g.setColor(Color.black);
        g.drawRect(x, y, width, height);
        int drawx = x + INSET, drawy = y + INSET;
        Font font = Screen.screen.getFont();
        g.setFont(font);
        FontMetrics fontMetrics = Screen.screen.getToolkit().getFontMetrics(font);
        g.setFont(font);
        for (int line = 0; line < lines.length; line++)
            {
            g.drawString(lines[line], drawx, drawy + fontMetrics.getMaxAscent());
            drawy += fontMetrics.getHeight();
            }
        }
    }

class DrawableCircle extends Drawable
    {
    private static final int THICKNESS = 3;

    private int diameter;
    private int clickingDiameter;
    private boolean filled;

    public DrawableCircle(Position position, int diameter, boolean filled,
        boolean widerClicking)
        {
        this.reposition(position);
        this.diameter = diameter;
        if (widerClicking)
            clickingDiameter = diameter * 3;
        else
            clickingDiameter = diameter;
        this.filled = filled;
        }
    public boolean inside(Position position)
        {
        int radius = clickingDiameter / 2;
        return (position.x() > x - radius && position.x() < x + radius &&
            position.y() > y - radius && position.y() < y + radius);
        }
    public void draw(Graphics g)
        {
        int radius = diameter / 2;
        g.setColor(color);
        if (filled)
            g.fillOval(x - radius, y - radius, diameter, diameter);
        else
            for (int i = 0, j = 0; i < THICKNESS; i++, j += 2)
                g.drawOval(x - radius - i, y - radius - i, diameter + j,
                    diameter + j);

        }
    }

class DrawableCross extends Drawable
    {
    private static final int SIZE = 4;

    public DrawableCross(Position position)
        {this.reposition(position);}
    public void draw(Graphics g)
        {
        g.setColor(color);
```

```
        for (int i = -1; i <= 1; i++)
            {
            g.drawLine(x - SIZE + i, y - SIZE, x + SIZE + i, y + SIZE);
            g.drawLine(x + SIZE + i, y - SIZE, x - SIZE + i, y + SIZE);
            }
        }
    }

class DrawableLine extends Drawable
    {
    private static final int DELTA = 10;

    private Position other;
    private int thickness;
    private float slope;
    private boolean steepSlope;
    private int leftX, rightX, upperY, lowerY;

    public DrawableLine(Position first, Position second, int thickness)
        {
        this.reposition(first);
        this.other = second;
        this.thickness = thickness;
        if (first.y() < second.y())
            {lowerY = second.y(); upperY = first.y();}
        else
            {lowerY = first.y(); upperY = second.y();}
        if (first.x() < second.x())
            {leftX = first.x(); rightX = second.x();}
        else
            {leftX = second.x(); rightX = first.x();}
        int deltaX = second.x() - first.x();
        int deltaY = second.y() - first.y();
        if (Math.abs(deltaX) < Math.abs(deltaY))
            {steepSlope = true; slope = (float) deltaX / deltaY; }
        else
            {steepSlope = false; slope = (float) deltaY / deltaX;}
        }
    public boolean inside(Position position)
        {
        if (steepSlope)
            {
            if (position.y() > upperY && position.y() < lowerY)
                {
                int xIntercept = (int) (x - (y - position.y()) * slope);
                if (position.x() > xIntercept - DELTA &&
                    position.x() < xIntercept + DELTA)
                    return true;
                }
            }
        else
            if (position.x() > leftX && position.x() < rightX)
                {
                int yIntercept = (int) (y + (position.x() - x) * slope);
                if (position.y() < yIntercept + DELTA &&
                    position.y() > yIntercept - DELTA)
                    return true;
                }
        return false;
        }
    public void draw(Graphics g)
        {
        g.setColor(color);
        for (int i = 0; i < thickness; i++)
            g.drawLine(x + i, y, other.x()+ i, other.y());
        }
```

```
    }

class HorizontalLine extends Drawable
    {
    private int left, right, y;

    public HorizontalLine(int left, int right, int y)
        {
        this.left = left;
        this.right = right;
        this.y = y;
        }
    public void draw(Graphics g)
        {
        g.setColor(Color.black);
        g.drawLine(left, y, right, y);
        }
    }

class DrawableArrow extends Drawable
    {
    private static int ARROW = 12;

    private double theta;
    private boolean flip;

    public void setDirection(Position to, boolean forward)
        {
        double deltaX = to.x() - x;
        double deltaY = to.y() - y;
        flip = x > to.x();
        if (!forward)
            {
            deltaX = -deltaX;
            deltaY = -deltaY;
            flip = to.x() > x;
            }
        if (deltaX != 0)
            theta = Math.atan(deltaY / deltaX);
        else if (deltaY > 0)
            theta = Math.PI / 2;
        else
            theta = -Math.PI / 2;
        theta += Math.PI / 2;
        }
    public void draw(Graphics g)
        {
        g.setColor(color);
        drawArrow(g, theta + Math.PI / 6);
        drawArrow(g, theta - Math.PI / 6);
        }
    private void drawArrow(Graphics g, double phi)
        {
        int deltaX = (int)Math.round(Math.sin(phi) * ARROW);
        int deltaY = (int)Math.round(Math.cos(phi) * ARROW);
        for (int xI = x - 1; xI <= x + 1; xI++)
            if (flip)
                g.drawLine(xI, y, xI + deltaX, y - deltaY);
            else
                g.drawLine(xI, y, xI - deltaX, y + deltaY);
        }
    }

class DrawableBar extends Drawable
    {
    private static final int WIDTH = 10;
```

```
    private static final int CLICK = 5;

    private int height;

    public DrawableBar(Position position)
        {x = position.x(); y = position.y();}
    public boolean inside(Position position)
        {
        return (position.x() > x - CLICK && position.x() < x + WIDTH + CLICK &&
            position.y() > y  && position.y() < y + height);
        }
    public void draw(Graphics g)
        {
        g.setColor(Color.yellow);
        g.fillRect(x, y, WIDTH, height);
        }
    public void setHeight(int height) {this.height = height;}
    }

class QuicksortBar extends Drawable
    {
    private static final int WIDTH = 10;

    private int height, first, last;

    public QuicksortBar(Position position)
        {x = position.x(); y = position.y();}
    public void draw(Graphics g)
        {
        g.setColor(Color.white);
        g.fillRect(x, y, WIDTH, height);
        g.setColor(Color.black);
        g.drawRect(x, y, WIDTH, height);
        g.drawLine(first, y, last + WIDTH - 1, y);
        }
    public void setDimensions(int height, int first, int last)
        {
        this.height = height;
        this.first = first;
        this.last = last;
        }
    }

class DrawableList extends Drawable
    {
    private static final int HEIGHT = 22;
    private protected static final int DELTA_X = 17;
    private protected static final int OFFSET_Y = 15;
    private protected static final int INSET_X = 2;

    private protected int count = 0, valueX;
    private int width;

    public DrawableList(Position position, int size)
        {
        this.reposition(position);
        valueX = x + INSET_X;
        width = size * DELTA_X + INSET_X * 2;
        }
    public void draw(Graphics g)
        {
        g.setColor(color);
        g.fillRect(x, y, width, HEIGHT);
        }
    }
```

```
class TraversalList extends DrawableList
    {
    private static final Color BOX_COLOR = new Color(128, 0, 64);

    private String list[];

    public TraversalList(Position position, int size)
        {
        super(position, size);
        list = new String[size];
        color = BOX_COLOR;
        }
    public void init()
        {
        count = 0;
        valueX = x + INSET_X;
        }
    public void draw(Graphics g)
        {
        super.draw(g);
        int numberX = x + INSET_X;
        g.setFont(Screen.screen.getFont());
        g.setColor(Color.white);
        for (int i = 0; i < count; i++)
            {
            g.drawString(list[i], numberX, y + OFFSET_Y);
            numberX += DELTA_X;
            }
        }
    public void append(String value, DrawableString number)
        {
        Position positions[] = new Position[1];
        number.setString(value);
        number.show();
        positions[0] = new Position(valueX, y + OFFSET_Y);
        number.moveTo(positions);
        Screen.canvas.awaitMovingCompletion();
        number.hide();
        list[count++] = value;
        valueX += DELTA_X;
        }
    }

// Classes that define the binary tree traversal lesson

class TraversalNode extends DrawableNode
    {
    private static final int PREORDER = 1;
    private static final int INORDER = 2;
    private static final int POSTORDER = 3;

    private TraversalNode leftChild;
    private TraversalNode rightChild;

    public DrawableNode left() {return leftChild;}
    public DrawableNode right() {return rightChild;}
    public TraversalNode(int level, Position position)
        {
        this.level = level;
        this.reposition(position);
        if (level < maxLevels)
            {
            leftChild = new TraversalNode(level + 1, leftPosition());
            rightChild = new TraversalNode(level + 1, rightPosition());
            }
        else
```

```
            nullNode = true;
        }
    public void randomize(int level)
        {
        numberChecked = 0;
        if (!(nullNode = isNullNode()))
            {
            value = Math.abs(random.nextInt() % 100);
            leftChild.randomize(level + 1);
            rightChild.randomize(level + 1);
            }
        }
    public void clearChecks()
        {
        if (!nullNode)
            {
            numberChecked = 0;
            leftChild.clearChecks();
            rightChild.clearChecks();
            }
        }
    public void check() {numberChecked++;}
    public void traversal(int order, TraversalPanel panel)
        {
        if (!nullNode)
            {
            panel.traversalMove(this, order == PREORDER, newPosition(), value, true);
            leftChild.traversal(order, panel);
            panel.traversalMove(this, order == INORDER, newPosition(), value, true);
            rightChild.traversal(order, panel);
            panel.traversalMove(this, order == POSTORDER, newPosition(), value, true);
            }
        }
    public void levelTraversal(Vector queue, TraversalPanel panel)
        {
        if (!leftChild.nullNode)
            queue.addElement(leftChild);
        if (!rightChild.nullNode)
            queue.addElement(rightChild);
        panel.traversalMove(this, true, newPosition(), value, false);
        if (queue.size() > 0)
            {
            TraversalNode next = (TraversalNode) queue.firstElement();
            queue.removeElement(next);
            next.levelTraversal(queue, panel);
            }
        panel.repositionTracer(new Position(x, y));
        }
    private Position newPosition()
        {return new Position(x - xOffset, y + yOffset);}
    }

class TraversalPanel extends ScreenPanel
    {
    private static final int MAKE_TREE_BUTTON = 0;
    private static final int LEVEL_ORDER_BUTTON = 4;
    private static final Position ROOT_POSITION = new Position(273, 20);
    private static final Position LIST_POSITION = new Position(0, 250);
    private static final int TRACER_DIAMETER = 30;

    private TraversalNode binaryTree = new TraversalNode(1, ROOT_POSITION);
    private Vector queue = new Vector();
    private TraversalList traversalList = new TraversalList(LIST_POSITION,
        DrawableNode.maxNodes);
    private DrawableCircle tracer = new DrawableCircle(ROOT_POSITION,
        TRACER_DIAMETER, false, false);
```

```
private DrawableString nodeLabel = new DrawableString(ROOT_POSITION);
private String showHelp[] = {
    "Generates a new binary tree with a random number of nodes and random values.",
    "Demonstrates a preorder traversal of the binary tree on the screen.",
    "Demonstrates an inorder traversal of the binary tree on the screen.",
    "Demonstrates a postorder traversal of the binary tree on the screen.",
    "Demonstrates a level order traversal of the binary tree on the screen."};
private String tryHelp[] = {
    "Generates a new binary tree with a random number of nodes and random values.",
    "You will be asked to click the nodes of this binary tree in preorder.",
    "You will be asked to click the nodes of this binary tree in inorder.",
    "You will be asked to click the nodes of this binary tree in postorder.",
    "You will be asked to click the nodes of this binary tree in level order."};

public TraversalPanel(String names[], int first, boolean allowTry,
    Color backgroundColor)
    {
    super.init(names, showHelp, tryHelp, first, allowTry, null, false,
        backgroundColor);
    binaryTree.randomize(1);
    Screen.canvas.add(binaryTree);
    Screen.canvas.add(traversalList);
    Screen.canvas.add(tracer);
    tracer.setColor(Color.white);
    Screen.canvas.add(nodeLabel);
    binaryTree.show();
    Screen.canvas.repaint();
    }
public void buttonClicked(int buttonNumber)
    {
    Screen.interaction.displayMessage("");
    if (buttonNumber == MAKE_TREE_BUTTON)
        {
        traversalList.hide();
        binaryTree.randomize(1);
        }
    else
        {
        if (isInteractive())
            tracer.hide();
        else
            tracer.show();
        tracer.reposition(ROOT_POSITION);
        traversalList.init();
        traversalList.show();
        Screen.canvas.repaint();
        if (buttonNumber == LEVEL_ORDER_BUTTON)
            binaryTree.levelTraversal(queue, this);
        else
            {
            binaryTree.traversal(buttonNumber, this);
            binaryTree.clearChecks();
            }
        tracer.hide();
        }
    binaryTree.resetColor();
    Screen.canvas.repaint();
    }
public void traversalMove(TraversalNode node, boolean visitNode, Position position,
    int value, boolean checkNode)
    {
    node.move("Click the Next Node", visitNode, tracer);
    if (checkNode)
        node.check();
    if (visitNode)
        {
```

```
            nodeLabel.reposition(position);
            node.changeColor();
            traversalList.append("" + value, nodeLabel);
            }
        Screen.controls.nextStep(visitNode, true);
        }
    public void repositionTracer(Position position)
        {tracer.reposition(position);}
    }

// Classes that define heaps

class HeapNode extends DrawableNode
    {
    private int nodeNumber;
    private HeapNode parent;
    private HeapNode leftChild;
    private HeapNode rightChild;
    private Heap heap;

    public DrawableNode left() {return leftChild;}
    public DrawableNode right() {return rightChild;}
    public HeapNode(int level, int nodeNumber, Position position, HeapNode parent,
        HeapNode heapList[], Heap heap)
        {
        this.level = level;
        this.nodeNumber = nodeNumber;
        x = position.x();
        y = position.y();
        this.parent = parent;
        this.heap = heap;
        nullNode = true;
        heapList[nodeNumber] = this;
        if (level < maxLevels)
            {
            leftChild = new HeapNode(level + 1, nodeNumber * 2,
                leftPosition(), this, heapList, heap);
            rightChild = new HeapNode(level + 1, nodeNumber * 2 + 1,
                rightPosition(), this, heapList, heap);
            }
        }
    public void addRightLeaf(int value, DrawableString nodeLabel1,
        DrawableString nodeLabel2)
        {
        this.value = value;
        nullNode = false;
        reheapUp(nodeLabel1, nodeLabel2);
        }
    public void setNode(int value, boolean nullNode)
        {
        this.value = value;
        this.nullNode = nullNode;
        }
    public void removeRightLeaf(HeapNode root, DrawableString nodeLabel1,
        DrawableString nodeLabel2)
        {
        int newRootValue = value;
        nullNode = true;
        if (this != root)
            {
            root.value = -1;
            nodeLabel1.setString("" + value);
            nodeLabel1.reposition(NodeStringPosition(x, y));
            moveValue(root, nodeLabel1);
            heap.startSwap(root.nodeNumber, nodeNumber);
            Screen.canvas.awaitMovingCompletion();
```

```
        heap.finishSwap(root.nodeNumber, nodeNumber);
        root.value = newRootValue;
        nodeLabel1.hide();
        root.reheapDown(this.nodeNumber, nodeLabel1, nodeLabel2);
        }
    Screen.canvas.repaint();
    }
public void reheapUp(DrawableString nodeLabel1, DrawableString nodeLabel2)
    {
    HeapNode current = this;
    HeapNode parent = this.parent;
    while (parent != null)
        {
        int currentValue = current.value;
        int parentValue = parent.value;
        if (parentValue >= currentValue)
            break;
        current.swapNodes(parent, nodeLabel1, nodeLabel2);
        current = parent;
        parent = current.parent;
        }
    }
public void reheapDown(int numberOfNodes, DrawableString nodeLabel1,
    DrawableString nodeLabel2)
    {
    HeapNode next;
    HeapNode current = this;
    int currentValue = current.value;
    while (current.leftChild.nodeNumber <= numberOfNodes)
        {
        if (current.rightChild.nodeNumber > numberOfNodes)
            next = current.leftChild;
        else
            {
            int leftValue = current.leftChild.value;
            int rightValue = current.rightChild.value;
            if (leftValue > rightValue)
                next = current.leftChild;
            else
                next = current.rightChild;
            }
        int nextValue = next.value;
        if (nextValue > currentValue)
            current.swapNodes(next, nodeLabel1, nodeLabel2);
        current = next;
        }
    }
public void swapNodes(HeapNode second, DrawableString nodeLabel1,
    DrawableString nodeLabel2)
    {
    HeapNode first = this;
    int firstValue = first.value;
    int secondValue = second.value;
    nodeLabel1.setString("" + first.value);
    nodeLabel1.reposition(NodeStringPosition(first.x, first.y));
    nodeLabel2.setString("" + second.value);
    nodeLabel2.reposition(NodeStringPosition(second.x, second.y));
    if (Screen.panel.isInteractive())
        {
        Screen.canvas.acceptClick(nodeLabel1);
        Screen.canvas.acceptClick(nodeLabel2);
        Screen.canvas.awaitClick("Click Nodes To Swap", true, false);
        }
    else
        Screen.controls.nextStep(true, true);
    first.moveValue(second, nodeLabel1);
```

```
        second.moveValue(first, nodeLabel2);
        heap.startSwap(first.nodeNumber, second.nodeNumber);
        Screen.canvas.awaitMovingCompletion();
        heap.finishSwap(first.nodeNumber, second.nodeNumber);
        nodeLabel1.hide();
        nodeLabel2.hide();
        first.value = secondValue;
        second.value = firstValue;
        Screen.canvas.repaint();
        }
    }

class HeapList extends SwapableList
    {
    private static final Color SORTED_COLOR = new Color(180, 180, 255);

    public HeapList(Position position, int size)
        {super(position, size, SORTED_COLOR);}
    public void push(int value) {list[++count] = value;}
    public void pop() {count--;}
    }

class Heap extends Drawable
    {
    private protected int numberOfNodes = 0;
    private protected HeapNode root;
    private protected HeapNode heap[] = new HeapNode[DrawableNode.maxNodes * 2];
    private protected HeapList list;

    public Heap(Position rootPosition, Position listPosition)
        {
        root = new HeapNode(1, 1, rootPosition, null, heap, this);
        list = new HeapList(listPosition, DrawableNode.maxNodes);
        }
    public void draw(Graphics g)
        {
        root.draw(g);
        list.draw(g);
        }
    public boolean inside(Position position) {return root.inside(position);}
    public void startSwap(int left, int right) {list.startSwap(left, right);}
    public void finishSwap(int left, int right) {list.finishSwap(left, right);}
    public void markSorted(int index) {list.markSorted(index);}
    }

// Classes that define the priority queue lesson

class PQHeap extends Heap
    {
    private static final int DIGITS = 2;
    private static final int LOWER = 0;
    private static final int UPPER = 99;

    public PQHeap(Position rootPosition, Position listPosition)
        {super(rootPosition, listPosition);}
    public void enqueue(DrawableString nodeLabel1, DrawableString nodeLabel2)
        {
        Screen.interaction.displayMessage(" ");
        if (numberOfNodes < DrawableNode.maxNodes - 1)
            {
            if (Screen.panel.userPicks())
                Screen.prompt.displayPrompt("Enter Value To Enqueue: ");
            int nodeValue = Screen.prompt.getData(DIGITS, LOWER, UPPER, false);
            if (nodeValue >= 0)
                {
                list.push(nodeValue);
```

```
                heap[++numberOfNodes].addRightLeaf(nodeValue,nodeLabel1, nodeLabel2);
                }
            }
        else
            Screen.interaction.displayMessage("Queue Is Full");
        }
    public void dequeue(DrawableString nodeLabel1, DrawableString nodeLabel2)
        {
        Screen.interaction.displayMessage(" ");
        if (numberOfNodes > 0)
            {
            heap[numberOfNodes--].removeRightLeaf(root, nodeLabel1, nodeLabel2);
            list.pop();
            }
        else
            Screen.interaction.displayMessage("Queue Is Empty");
        }
    }

class PQHeapPanel extends ScreenPanel
    {
    private static final int ENQUEUE_BUTTON = 0;
    private static final Position ROOT_POSITION = new Position(273, 20);
    private static final Position LIST_POSITION = new Position(0, 250);

    private PQHeap heap = new PQHeap(ROOT_POSITION, LIST_POSITION);
    private DrawableString nodeLabel1 = new DrawableString(ROOT_POSITION);
    private DrawableString nodeLabel2 = new DrawableString(ROOT_POSITION);
    private String randomPickHelp = "Selecting the Random button causes values to be randomly
generated for the enqueue operation.  Selecting I'll Pick allows you to enter the value in a
box the will appear in the upper left corner of the screen when the enqueue operation is
selected.";
    private String showHelp[] = {
        "Enqueues a new value onto the priority queue, which adds a node at the bottom of the
tree.  The tree is then transformed back into a heap.",
        "Dequeues the highest value from the queue, which removes the value in the root.  The
bottom value is moved to the root and the tree is then transformed back into a heap."};
    private String tryHelp[] = {
        "Enqueues a new value onto the priority queue, which adds a node at the bottom of the
tree.  If the resulting tree is not a heap, you will be asked to click the node pairs to swap
to transform it into a heap.",
        "Dequeues the highest value from the queue, which removes the value in the root.  The
bottom value is moved to the root.  If the resulting tree is not a heap, you will be asked to
click the node pairs to swap to transform it into a heap."};

    public PQHeapPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, randomPickHelp,
            false, backgroundColor);
        Screen.canvas.add(heap);
        Screen.canvas.add(nodeLabel1);
        Screen.canvas.add(nodeLabel2);
        heap.show();
        Screen.canvas.repaint();
        }
    public void buttonClicked(int buttonNumber)
        {
        if (buttonNumber == ENQUEUE_BUTTON)
            heap.enqueue(nodeLabel1, nodeLabel2);
        else
            heap.dequeue(nodeLabel1, nodeLabel2);
        Screen.canvas.repaint();
        }
    }
```

```
// Classes that define the heap sort lesson

class HeapSort extends Heap
    {
    private static final int MIN_NODES = 10;

    private static Random random = new Random();

    public HeapSort(Position rootPosition, Position listPosition)
        {super(rootPosition, listPosition);}
    public void makeTree()
        {
        int values[] = new int[DrawableNode.maxNodes];
        numberOfNodes = Math.abs(random.nextInt() % (DrawableNode.maxNodes / 2)) + MIN_NODES;
        for (int node = 1; node <= DrawableNode.maxNodes; node++)
            {
            if (node <= numberOfNodes)
                {
                int nodeValue = Math.abs(random.nextInt() % 100);
                heap[node].setNode(nodeValue, false);
                values[node] = nodeValue;
                }
            else
                heap[node].setNode(-1, true);
            }
        list.setValues(values, numberOfNodes);
        }
    public void sort(DrawableString nodeLabel1, DrawableString nodeLabel2)
        {
        Screen.interaction.displayMessage("Changing into a Heap");
        int totalNodes = numberOfNodes;
        for (int node = 1; node <= totalNodes; node++)
            {
            numberOfNodes = totalNodes;
            heap[node].reheapUp(nodeLabel1, nodeLabel2);
            heap[node].changeColor();
            }
        Screen.interaction.displayMessage("Performing Sort");
        while (numberOfNodes > 1)
            {
            root.swapNodes(heap[numberOfNodes], nodeLabel1, nodeLabel2);
            heap[numberOfNodes].changeColor();
            markSorted(numberOfNodes);
            root.reheapDown(--numberOfNodes, nodeLabel1, nodeLabel2);
            }
        root.changeColor();
        markSorted(numberOfNodes);
        Screen.interaction.displayMessage("");
        }
    }

class HeapSortPanel extends ScreenPanel
    {
    private static final int MAKE_TREE_BUTTON = 0;
    private static final Position ROOT_POSITION = new Position(273, 20);
    private static final Position LIST_POSITION = new Position(0, 250);

    private HeapSort heap = new HeapSort(ROOT_POSITION, LIST_POSITION);
    private DrawableString nodeLabel1 = new DrawableString(ROOT_POSITION);
    private DrawableString nodeLabel2 = new DrawableString(ROOT_POSITION);
    private String showHelp[] = {
        "Generates an almost complete binary tree with a random number of nodes containing
random values.",
        "Demonstrates the heap sort algorithm by first transforming the tree into a heap and
then performing the sort."};
    private String tryHelp[] = {
```

```
        "Generates an almost complete binary tree with a random number of nodes containing
random values.",
        "You will be asked to perform the heap sort algorithm by indicating which node pairs
are to be swapped at each step of the algorithm."};

    public HeapSortPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, null, false,
            backgroundColor);
        Screen.canvas.add(heap);
        Screen.canvas.add(nodeLabel1);
        Screen.canvas.add(nodeLabel2);
        heap.show();
        buttonClicked(MAKE_TREE_BUTTON);
        Screen.canvas.repaint();
        }
    public void buttonClicked(int buttonNumber)
        {
        if (buttonNumber == MAKE_TREE_BUTTON)
            heap.makeTree();
        else
            heap.sort(nodeLabel1, nodeLabel2);
        Screen.canvas.repaint();
        }
    }

// Classes that define the binary search tree lesson

class BSNode extends DrawableNode
    {
    private BSNode leftChild, rightChild, parent;
    private Vector values = new Vector();

    public DrawableNode left() {return leftChild;}
    public DrawableNode right() {return rightChild;}
    public BSNode(int level, Position position, BSNode parent)
        {
        this.level = level;
        this.reposition(position);
        this.parent = parent;
        if (level < maxLevels)
            {
            leftChild = new BSNode(level + 1, leftPosition(), this);
            rightChild = new BSNode(level + 1, rightPosition(), this);
            }
        nullNode = true;
        }
    public void insert(int value, DrawableCircle tracer)
        {
        BSNode node = retrieve(value, tracer, false, false);
        if (node.nullNode && node.level < maxLevels && Screen.panel.isInteractive())
            Screen.canvas.acceptClick(node);
        Screen.answer.twoButtons("Click Location For " + value + " Or",
            "Branch Full", "In Tree", "That Branch Is Full",
            "Value Already In Tree", node.level >= maxLevels,
             !node.nullNode);
        if (node.nullNode && node.level < maxLevels)
            {
            node.value = value;
            node.nullNode = false;
            values.addElement(new Integer(value));
            }
        }
    public void delete(int value, DrawableString nodeLabel, DrawableCircle tracer)
        {
```

```
    BSNode oneChild;
    BSNode node = retrieve(value, tracer, false, false);
    if (!node.nullNode && Screen.panel.isInteractive())
        Screen.canvas.acceptClick(node);
    Screen.answer.oneButton(node.nullNode, "Click Node With " + value + " Or",
        "Value Is Not In Tree", "Value Deleted", "That Is The Node To Delete");
    if (!node.nullNode)
        {
        values.removeElement(new Integer(value));
        if (node.leftChild.nullNode && node.rightChild.nullNode)
            node.nullNode = true;
        else if (node.leftChild.nullNode || node.rightChild.nullNode)
            {
            if (node.leftChild.nullNode)
                oneChild = node.rightChild;
            else
                oneChild = node.leftChild;
            BSNode subtree = new BSNode(oneChild, node);
            Screen.canvas.awaitMovingCompletion();
            subtree.removeAndCopy(node);
            }
        else
            {
            BSNode successor = node.rightChild;
            if (!Screen.panel.isInteractive())
                Screen.canvas.highlight(new DrawableLine(new Position(node.x, node.y),
                    new Position(successor.x, successor.y), 3), Color.yellow, false);
            while (!successor.leftChild.nullNode)
                {
                BSNode previous = successor;
                successor = successor.leftChild;
                if (!Screen.panel.isInteractive())
                    Screen.canvas.highlight(new DrawableLine(
                        new Position(previous.x, previous.y),
                        new Position(successor.x, successor.y), 3), Color.yellow, false);
                }
            int successorValue = successor.value;
            nodeLabel.setString("" + successor.value);
            nodeLabel.reposition(NodeStringPosition(successor.x, successor.y));
            if (Screen.panel.isInteractive())
                {
                Screen.canvas.acceptClick(nodeLabel);
                Screen.canvas.awaitClick("Click Value To Move", true, false);
                Screen.answer.clearAnswer();
                }
            else
                Screen.controls.nextStep(true, false);
            successor.moveValue(node, nodeLabel);
            Screen.canvas.awaitMovingCompletion();
            nodeLabel.hide();
            node.value = successorValue;
            if (successor.rightChild.nullNode)
                successor.nullNode = true;
            else
                {
                BSNode subtree = new BSNode(successor.rightChild, successor);
                Screen.canvas.awaitMovingCompletion();
                subtree.removeAndCopy(successor);
                }
            Screen.canvas.clear();
            }
        }
    }
public void find(int value, DrawableCircle tracer)
    {
    BSNode node = retrieve(value, tracer, true, false);
```

```
    Screen.answer.twoButtons("Click One Of Following",
        "Value Found", "Not In Tree", "The Value Was Found",
        "Value Is Not In Tree", !node.nullNode, true);
    node = retrieve(value, tracer, false, true);
    }
private BSNode retrieve(int value, DrawableCircle tracer, boolean visit,
    boolean recolorOnly)
    {
    if (nullNode)
        return this;
    if (!recolorOnly)
        {
        move("Click Next Node To Find " + value, visit, tracer);
        Screen.controls.nextStep(false, true);
        }
    if (visit || recolorOnly)
        changeColor();
    if (value < this.value)
        return leftChild.retrieve(value, tracer, visit, recolorOnly);
    else if (value > this.value)
        return rightChild.retrieve(value, tracer, visit, recolorOnly);
    else
        return this;
    }
private BSNode(BSNode copy, BSNode to)
    {
    x = copy.x;
    y = copy.y;
    level = copy.level;
    value = copy.value;
    nullNode = false;
    copy.nullNode = true;
    movingTreeNode = true;
    nodeColor = Color.gray;
    Screen.canvas.add(this);
    show();
    Position positions[] = new Position[1];
    positions[0] = new Position(to.x, to.y);
    moveTo(positions);
    if (copy.leftChild.nullNode)
        leftChild = null;
    else
        leftChild = new BSNode(copy.leftChild, to.leftChild);
    if (copy.rightChild.nullNode)
        rightChild = null;
    else
        rightChild = new BSNode(copy.rightChild, to.rightChild);
    }
private void removeAndCopy(BSNode copy)
    {
    copy.value = this.value;
    copy.nullNode = false;
    Screen.canvas.remove(this);
    if (leftChild != null)
        leftChild.removeAndCopy(copy.leftChild);
    else
        copy.leftChild.nullNode = true;
    if (rightChild != null)
        rightChild.removeAndCopy(copy.rightChild);
    else
        copy.rightChild.nullNode = true;
    }
public int getValue()
    {
    int one_in_ten = Math.abs(random.nextInt()) % 10;
    if (one_in_ten == 0 || values.size() == 0)
```

```
            return Math.abs(random.nextInt()) % 100;
        int choice = Math.abs(random.nextInt()) % values.size();
        int value = ((Integer)values.elementAt(choice)).intValue();
        return value;
        }
    }

class BSTreePanel extends ScreenPanel
    {
    private static final int INSERT_BUTTON = 0;
    private static final int DELETE_BUTTON = 1;
    private static final int FIND_BUTTON = 2;
    private static final int DIGITS = 2;
    private static final int LOWER = 0;
    private static final int UPPER = 99;
    private static final Position ROOT_POSITION = new Position(275, 20);
    private static final Position LIST_POSITION = new Position(2, 257);
    private static final int TRACER_DIAMETER = 30;

    private BSNode binaryTree = new BSNode(1, ROOT_POSITION, null);
    private DrawableCircle tracer = new DrawableCircle(ROOT_POSITION,
        TRACER_DIAMETER, false, false);
    private DrawableString nodeLabel = new DrawableString(ROOT_POSITION);
    private String randomPickHelp = "Selecting the Random button causes values to be randomly
generated.  Selecting I'll Pick allows you to enter the value in a box the will appear at the
top of the screen when a value is required.";
    private String showHelp[] = {
        "Demonstrates the insertion of a new value into the binary search tree.  A search is
performed to find the location and a new node is added at the bottom of the tree.  If the
value is already in the tree or the branch is full, a message is displayed.",
        "Demonstrates the deletion of a node in the binary search tree.  The steps required
to complete the deletion depend upon the number of children of the node containing the value
to be deleted.",
        "Demonstrates the find operation of a binary search tree.  The nodes of the path
followed to search for the value turn red as it is traced.  A message indicates whether it
was found."};
    private String tryHelp[] = {
        "You will be asked to click the location on the screen where the node containing the
new value should be inserted.  If the value is already in the tree or the branch where it
would be added is full, you are to click on the appropriate button at the top of the
screen.",
        "You will be asked to click the node containing the value to delete, or the button
indicating it is not in the tree.  If the node has two children, you will also be asked to
click the node containing the value to be moved to the node containing the value to be
deleted.",
        "You will be asked to click the nodes beginning at the root that trace the path that
is followed to search for the specified value.  Finally, you be asked to click the
appropriate button at the top of the screen, depending upon whether the value was found or
not."};

    public BSTreePanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry,
            randomPickHelp, false, backgroundColor);
        Screen.canvas.add(binaryTree);
        Screen.canvas.add(tracer);
        Screen.canvas.add(nodeLabel);
        binaryTree.show();
        Screen.canvas.repaint();
        }
    public void buttonClicked(int buttonNumber)
        {
        int value;
        Screen.interaction.displayMessage("");
        if (isInteractive())
```

```
            tracer.hide();
        else
            tracer.show();
        tracer.reposition(ROOT_POSITION);
        Screen.canvas.repaint();
        if (buttonNumber == INSERT_BUTTON)
            {
            if (Screen.panel.userPicks())
                Screen.prompt.displayPrompt("Enter Value To Insert: ");
            value = Screen.prompt.getData(DIGITS, LOWER, UPPER, false);
            if (value >= 0)
                {
                if (!Screen.panel.isInteractive())
                    Screen.prompt.displayPrompt("Number To Insert: " + value);
                binaryTree.insert(value, tracer);
                }
            }
        else if (buttonNumber == DELETE_BUTTON)
            {
            if (Screen.panel.userPicks())
                {
                Screen.prompt.displayPrompt("Enter Value To Delete: ");
                value = Screen.prompt.getData(DIGITS, LOWER, UPPER, false);
                }
            else
                value = binaryTree.getValue();
            if (value >= 0)
                {
                if (!Screen.panel.isInteractive())
                    Screen.prompt.displayPrompt("Number To Delete: " + value);
                binaryTree.delete(value, nodeLabel, tracer);
                }
            }
        else
            {
            if (!isInteractive())
                tracer.show();
            if (Screen.panel.userPicks())
                {
                Screen.prompt.displayPrompt("Enter Value To Find: ");
                value = Screen.prompt.getData(DIGITS, LOWER, UPPER, false);
                }
            else
                value = binaryTree.getValue();
            if (value >= 0)
                {
                if (!Screen.panel.isInteractive())
                    Screen.prompt.displayPrompt("Number To Find: " + value);
                binaryTree.find(value, tracer);
                }
            }
        tracer.hide();
        Screen.canvas.repaint();
        }
    }

// Classes that define the height balanced tree lesson

class HBNode extends DrawableNode
    {
    private static final int HIGHLIGHT_DIAMETER = 30;

    private String stringValue;
    private HBNode leftChild, rightChild, parent;
    private int height;
```

```
public DrawableNode left() {return leftChild;}
public DrawableNode right() {return rightChild;}
public HBNode(int level, Position position, HBNode parent)
    {
    this.level = level;
    this.reposition(position);
    this.parent = parent;
    if (level < maxLevels)
        {
        leftChild = new HBNode(level + 1, leftPosition(), this);
        rightChild = new HBNode(level + 1, rightPosition(), this);
        }
    nullNode = true;
    }
public void randomize()
    {
    stringValueNode = true;
    if (!(nullNode = isNullNode()))
        {
        stringValue = "";
        leftChild.randomize();
        rightChild.randomize();
        height = Math.max(leftChild.height, rightChild.height) + 1;
        }
    else
        height = 0;
    }
public void clear()
    {
    if (!nullNode)
        {
        stringValue = "";
        leftChild.clear();
        rightChild.clear();
        }
    }
public void heights(boolean set)
    {
    if (!nullNode)
        {
        leftChild.heights(set);
        rightChild.heights(set);
        if (!Screen.panel.isInteractive())
            Screen.controls.nextStep(true, true);
        else if (!set)
            {
            Screen.canvas.highlight(new DrawableCircle(new Position(x, y),
                HIGHLIGHT_DIAMETER, true, false), Color.yellow, false);
            Screen.prompt.awaitInput("Enter Height Of Subtree: ", height);
            Screen.canvas.clear();
            }
    stringValue = "   " + height;
        }
    }
public void balanceFactors()
    {
    if (!nullNode)
        {
        int factor = leftChild.height - rightChild.height;
        if (Screen.panel.isInteractive())
            {
            Screen.canvas.highlight(new DrawableCircle(new Position(x, y),
                HIGHLIGHT_DIAMETER, true, false), Color.yellow, false);
            Screen.prompt.awaitInput("Enter Balance Factor: ", factor);
            Screen.canvas.clear();
            }
```

```
            else
                Screen.controls.nextStep(true, true);
            stringValue = "" + leftChild.height + "-" + rightChild.height +
                "=" + factor;
            leftChild.balanceFactors();
            rightChild.balanceFactors();
            }
        }
    public int isHBn(int n, boolean checkWho)
        {
        if (!nullNode)
            {
            int unBalanced = 0;
            int factor = leftChild.height - rightChild.height;
            if (Math.abs(factor) > n)
                {
                Screen.canvas.highlight(new DrawableCircle(new Position(x, y),
                    HIGHLIGHT_DIAMETER, false, false), Color.yellow, checkWho);
                if (!(checkWho && Screen.panel.isInteractive()))
                    stringValue = "" + leftChild.height + "-" + rightChild.height +
                        "=" + factor;
                unBalanced = 1;
                }
            else
                stringValue = "";
            return leftChild.isHBn(n, checkWho) + rightChild.isHBn(n, checkWho)
                + unBalanced;
            }
        return 0;
        }
    public String getString() {return stringValue;}
    }

class HBTreePanel extends ScreenPanel
    {
    private static final int MAKE_TREE_BUTTON = 0;
    private static final int HEIGHT_BUTTON = 1;
    private static final int BALANCE_FACTORS_BUTTON = 2;
    private static final int IS_BALANCED_BUTTON = 3;
    private static final int IS_HBn_BUTTON = 4;
    private static final Position ROOT_POSITION = new Position(273, 20);

    private HBNode binaryTree = new HBNode(1, ROOT_POSITION, null);
    private String showHelp[] = {
        "Generates a binary tree with a random number of nodes.",
        "Each node is labeled with the height of the subtree of which it is the root.",
        "Each node is labeled with the computation of its balance factor as the difference
between the height of its left and right subtrees.",
        "If the tree on the screen is an balanced tree, a message is displayed at the top of
the screen, otherwise the nodes of the tree with improper balance factors are labeled.",
        "You will be prompted first to enter the value of n in a box in the upper left
corner. If the tree on the screen is a height-balanced n, a message will be displayed at the
top of the screen, otherwise the nodes of the tree with improper balance factors are
labeled."};
    private String tryHelp[] = {
        "Generates a binary tree with a random number of nodes.",
        "You will be asked to input in a box in the upper left corner the height of the
subtree whose root will be highlighted in yellow.",
        "You will be asked to input in a box in the upper left corner the balance factor of
the node that will be highlighted in yellow.",
        "You will be asked to click on a button at the top of the screen if the tree is an
balanced tree, otherwise you will be asked to click on one of the nodes with an improper
balance factor.",
        "You will be prompted first to enter the value of n in a box in the upper left
corner. You will then be asked to click on a button at the top of the screen if the tree is
```

```
height-balanced n, otherwise you will be asked to click on one of the nodes with an improper
balance factor."};

    public HBTreePanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, null, true,
            backgroundColor);
        Screen.canvas.add(binaryTree);
        binaryTree.randomize();
        binaryTree.show();
        Screen.canvas.repaint();
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        binaryTree.clear();
        Screen.canvas.clear();
        Screen.canvas.repaint();
        if (buttonNumber == MAKE_TREE_BUTTON)
            binaryTree.randomize();
        else if (buttonNumber == HEIGHT_BUTTON)
            binaryTree.heights(false);
        else if (buttonNumber == BALANCE_FACTORS_BUTTON)
            {
            if (isInteractive())
                binaryTree.heights(true);
            binaryTree.balanceFactors();
            }
        else if (buttonNumber == IS_BALANCED_BUTTON || buttonNumber == IS_HBn_BUTTON)
            {
            int n = 1;
            String correctResponse;
            if (buttonNumber == IS_HBn_BUTTON)
                {
                Screen.prompt.displayPrompt("Enter Value for n (2-4)");
                n = Screen.prompt.getData(1, 2, 4, true);
                correctResponse = "It Is Height Balanced - " + n;
                }
            else
                correctResponse = "It Is An Balanced Tree";
            int unBalanced = binaryTree.isHBn(n, true);
            if (!Screen.answer.oneButton(unBalanced == 0, "Click Unbalanced Node or",
                correctResponse, "No, " + unBalanced + " Unbalanced Nodes",
                "That Node Is Unbalanced"))
                binaryTree.isHBn(n, false);
            }
        Screen.canvas.repaint();
        }
    }

// Classes that define the tree categorization lesson

class TreeNode extends DrawableNode
    {
    public static final int HEAP = 0;
    public static final int BINARY_SEARCH_TREE = 1;
    public static final int RANDOM_VALUE = 2;

    private static final int MAX_VALUE = 80;
    private static final int HIGHLIGHT_DIAMETER = 30;

    private static Vector highlighted = new Vector();
    private static int nextMin, increment;
    private TreeNode leftChild, rightChild, parent;
    private int height, nodeNumber;
```

```
public DrawableNode left() {return leftChild;}
public DrawableNode right() {return rightChild;}
public TreeNode(int level, int nodeNumber, Position position, TreeNode parent)
    {
    this.level = level;
    this.nodeNumber = nodeNumber;
    this.reposition(position);
    this.parent = parent;
    if (level < maxLevels)
        {
        leftChild = new TreeNode(level + 1, nodeNumber * 2, leftPosition(), this);
        rightChild = new TreeNode(level + 1, nodeNumber * 2 + 1, rightPosition(), this);
        }
    nullNode = true;
    }
public void nullify()
    {
    nullNode = true;
    if (leftChild != null) leftChild.nullify();
    if (rightChild != null) rightChild.nullify();
    }
public int randomizeShape(int totalNodes)
    {
    if (totalNodes == 0)
        nullNode = isNullNode();
    else
        nullNode = nodeNumber > totalNodes;
    if (!nullNode)
        {
        int biggest = Math.max(
            leftChild.randomizeShape(totalNodes),
            rightChild.randomizeShape(totalNodes));
        height = Math.max(leftChild.height, rightChild.height) + 1;
        return Math.max(biggest, nodeNumber);
        }
    else
        {
        height = 0;
        return 0;
        }
    }
public boolean randomizeValues(int constraint)
    {
    boolean valid;
    if (!nullNode)
        {
        if (constraint == BINARY_SEARCH_TREE)
            {
            leftChild.randomizeValues(constraint);
            value = Screen.panel.getRandom(nextMin, nextMin + increment);
            nextMin = value + 1;
            rightChild.randomizeValues(constraint);
            return true;
            }
        else if (constraint == HEAP)
            do
                {
                value = Math.abs(random.nextInt() % MAX_VALUE);
                if (parent == null)
                    valid = true;
                else
                    {
                    if (parent.value == 0) return false;
                    valid = value < parent.value;
                    }
```

```
                }
            while (!valid);
        else
            value = Math.abs(random.nextInt() % MAX_VALUE);
        return leftChild.randomizeValues(constraint) &&
            rightChild.randomizeValues(constraint);
        }
    return true;
    }
public boolean isAlmost(int totalNodes, boolean checkWho)
    {
    boolean almost;
    if (nodeNumber <= totalNodes)
        if (nullNode)
            {
            Screen.canvas.highlight(new HiddenNode(new Position(x, y), level),
                Color.yellow, checkWho);
            almost = false;
            }
        else
            almost = true;
    else
        return true;
    boolean leftAlmost = leftChild.isAlmost(totalNodes, checkWho);
    boolean rightAlmost = rightChild.isAlmost(totalNodes, checkWho);
    return leftAlmost && rightAlmost && almost;
    }
public boolean isBSTree(boolean highlight, boolean checkWho)
    {
    boolean bstree = true;
    boolean leftBSTree = true;
    boolean rightBSTree = true;
    if (!leftChild.nullNode)
        {
        bstree &= value > leftChild.maxValue();
        leftBSTree = leftChild.isBSTree(highlight, checkWho);
        }
    if (!rightChild.nullNode)
        {
        bstree &= value < rightChild.minValue();
        rightBSTree = rightChild.isBSTree(highlight, checkWho);
        }
    boolean subtrees =  leftBSTree && rightBSTree;
    if (!subtrees)
       return false;
    if (!bstree)
        if (highlight)
            Screen.canvas.highlight(new DrawableCircle(new Position(x, y),
                HIGHLIGHT_DIAMETER, false, false), Color.yellow, checkWho);
    return bstree;
    }
public boolean isHeap(boolean checkWho)
    {
    boolean heap;
    if (parent == null)
        heap = true;
    else if (!nullNode)
        {
        heap = value < parent.value;
        if (!heap)
            Screen.canvas.highlight(new DrawableLine(new Position(x, y),
                new Position(parent.x, parent.y), 3), Color.yellow, checkWho);
        }
    else
        return true;
    boolean leftHeap = leftChild.isHeap(checkWho);
```

```
        boolean rightHeap = rightChild.isHeap(checkWho);
        return leftHeap && rightHeap && heap;
        }
    public int isBalanced()
        {
        if (!nullNode)
            {
            int unBalanced = 0;
            int factor = leftChild.height - rightChild.height;
            if (Math.abs(factor) > 1)
                unBalanced = 1;
            return leftChild.isBalanced() + rightChild.isBalanced() +
                unBalanced;
            }
        return 0;
        }
    public void setIncrement(int totalNodes)
        {
        nextMin = 0;
        increment = (MAX_VALUE * 2) / totalNodes;
        }

    private int minValue()
        {
        int minimum = value;
        if (!leftChild.nullNode)
            minimum = Math.min(minimum, leftChild.minValue());
        if (!rightChild.nullNode)
            minimum = Math.min(minimum, rightChild.minValue());
        return minimum;
        }
    private int maxValue()
        {
        int maximum = value;
        if (!leftChild.nullNode)
            maximum = Math.max(maximum, leftChild.maxValue());
        if (!rightChild.nullNode)
            maximum = Math.max(maximum, rightChild.maxValue());
        return maximum;
        }
    }

class TreesPanel extends ScreenPanel
    {
    private static final int MAKE_TREE_BUTTON = 0;
    private static final int IS_ALMOST_BUTTON = 1;
    private static final int IS_COMPLETE_BUTTON = 2;
    private static final int IS_BS_TREE_BUTTON = 3;
    private static final int IS_HEAP_BUTTON = 4;
    private static final int IS_AVL_BUTTON = 5;
    private static final int MIN_NODES = 4;
    private static final Position ROOT_POSITION = new Position(273, 20);

    private TreeNode binaryTree = new TreeNode(1, 1, ROOT_POSITION, null);
    private int totalNodes;
    private String showHelp[] = {
        "Generates a new binary tree with a random number of nodes and random values.",
        "If the tree on the screen is almost complete, a message is displayed at the top of
the screen, otherwise all the missing nodes are highlighted in yellow.",
        "If the tree on the screen is complete, a message is displayed at the top of the
screen, otherwise all the missing nodes are highlighted in yellow.",
        "If the tree on the screen is a binary search tree, a message is displayed at the top
of the screen, otherwise all of the roots of the lowest subtrees that are not binary search
trees are highlighted in yellow.",
```

```
        "If the tree on the screen is a heap, a message is displayed at the top of the
screen, otherwise all of the branches that violate the heap property are highlighted in
yellow.",
        "If the tree on the screen is an AVL tree, a message is displayed at the top of the
screen, otherwise the nodes of the tree with improper balance factors are highlighted in
yellow."};
    private String tryHelp[] = {
        "Generates a new binary tree with a random number of nodes and random values.",
        "You will be asked to click on a button at the top of the screen if the tree is
almost complete, otherwise you are to click on the location of any missing node.",
        "You will be asked to click on a button at the top of the screen if the tree is
complete, otherwise you are to click on the location of any missing node.",
        "You will be asked to click on a button at the top of the screen if the tree on the
screen is a binary search tree, otherwise you are to click on the root of any of the lowest
subtrees that are not binary search trees.",
        "You will be asked to click on a button at the top of the screen if the tree on the
screen is a heap, otherwise you are to click on any branch of the tree that violates the heap
property.",
        "You will be asked to click on a button at the top of the screen if the tree is an
AVL tree, otherwise you will be asked to click on one of the nodes with an improper balance
factor."};

    public TreesPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, null, false,
            backgroundColor);
        Screen.canvas.add(binaryTree);
        binaryTree.show();
        makeTree();
        Screen.canvas.repaint();
        }
    public void makeTree()
        {
        binaryTree.nullify();
        int type = Math.abs(random.nextInt() % 5);
        switch (type)
           {
           case 0: // Almost Complete Tree
                totalNodes = Math.abs(random.nextInt() %
                    DrawableNode.maxNodes);
                totalNodes = Math.max(totalNodes, MIN_NODES);
                totalNodes = binaryTree.randomizeShape(totalNodes);
                binaryTree.randomizeValues(TreeNode.RANDOM_VALUE);
                break;
           case 1: // Complete Tree
                int levels = Math.abs(random.nextInt() % 5 + 1);
                totalNodes = (int) Math.pow(2, levels) - 1;
                totalNodes = binaryTree.randomizeShape(totalNodes);
                binaryTree.randomizeValues(TreeNode.RANDOM_VALUE);
                break;
           case 2: // Binary Search Tree
                totalNodes = binaryTree.randomizeShape(0);
                binaryTree.setIncrement(totalNodes);
                binaryTree.randomizeValues(TreeNode.BINARY_SEARCH_TREE);
                break;
            case 3: // Heap
                totalNodes = Math.abs(random.nextInt() %
                    DrawableNode.maxNodes);
                totalNodes = Math.max(totalNodes, MIN_NODES);
                totalNodes = binaryTree.randomizeShape(totalNodes);
                while (!binaryTree.randomizeValues(TreeNode.HEAP));
                break;
            default: // Totally Random
                totalNodes = binaryTree.randomizeShape(0);
                binaryTree.randomizeValues(TreeNode.RANDOM_VALUE);
```

```
            }
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        Screen.canvas.clear();
        Screen.canvas.repaint();
        if (buttonNumber == MAKE_TREE_BUTTON)
            makeTree();
        else if (buttonNumber == IS_ALMOST_BUTTON)
            {
            boolean isAlmost = binaryTree.isAlmost(totalNodes, true);
            if (!Screen.answer.oneButton(isAlmost, "Click A Missing Node or",
                "It Is Almost Complete", "Yellow Nodes Missing",
                "That Node Is Missing"))
                binaryTree.isAlmost(totalNodes, false);
            }
        else if (buttonNumber == IS_COMPLETE_BUTTON)
            {
            int complete;
            for (complete = 1; complete - 1 < totalNodes; complete *= 2);
            boolean isComplete = binaryTree.isAlmost(--complete, true);
            if (!Screen.answer.oneButton(isComplete, "Click A Missing Node or",
                "This Tree Is Complete", "Yellow Nodes Missing",
                "That Node Is Missing"))
                binaryTree.isAlmost(complete, false);
            }
        else if (buttonNumber == IS_BS_TREE_BUTTON)
            {
            boolean isBSTree = binaryTree.isBSTree(true, true);
            if (!Screen.answer.oneButton(isBSTree, "Click A Wrong Subtree or",
                "Is Binary Search Tree", "Yellow Subtrees Wrong",
                "That Subtree Is Wrong"))
                binaryTree.isBSTree(true, false);
            }
        else if (buttonNumber == IS_HEAP_BUTTON)
            {
            boolean isHeap = binaryTree.isHeap(true);
            if (!Screen.answer.oneButton(isHeap, "Click A Wrong Branch or",
                "This Tree Is A Heap", "Yellow Branches Wrong",
                "That Branch Is Wrong"))
                binaryTree.isHeap(false);
            }
        else if (buttonNumber == IS_AVL_BUTTON)
            {
            boolean isAVLTree = binaryTree.isBalanced() == 0 &&
                binaryTree.isBSTree(false, false);
            Screen.answer.twoButtons("Click One Of The Choices",
                "Is AVL Tree", "Not AVL Tree", "It Is An AVL Tree",
                "It Is Not An AVL Tree", isAVLTree, true);
            }
        Screen.canvas.repaint();
        }
    }

// Classes that define graph representation lesson

class MatrixGraph extends DrawableGraph
    {
    private static final int BOX_SIZE = 18;
    private static final int TOP_LABEL_OFFSET = 6;
    private static final int SIDE_LABEL_OFFSET = 12;
    private static final int X_OFFSET = 270;
    private static final int Y_OFFSET = 20;
    private static final int RADIUS = 5;
```

```
private int xVal = x + X_OFFSET, yVal = y - Y_OFFSET;
private boolean makeMatrix = true;
private boolean matrixShowing[][];
private boolean graphShowing[][];
private DrawableCircle tracer1 = new DrawableCircle(new Position(0, 0),
    RADIUS * 2, false, false);
private DrawableCircle tracer2 = new DrawableCircle(new Position(0, 0),
    RADIUS * 2, false, false);
private DrawableCircle correct1 = new DrawableCircle(new Position(0, 0),
    RADIUS * 2, false, true);
private DrawableCircle correct2 = new DrawableCircle(new Position(0, 0),
    RADIUS * 2, false, true);

public MatrixGraph(Position position, int rows, int cols)
    {
    super(position, rows, cols, true, 65);
    matrixShowing = new boolean[noOfNodes][noOfNodes];
    graphShowing = new boolean[noOfNodes][noOfNodes];
    Screen.canvas.add(tracer1);
    Screen.canvas.add(tracer2);
    Screen.canvas.add(correct1);
    Screen.canvas.add(correct2);
    }
public boolean isSelectable(int node) {return true;}
public int xCoor(int node)
    {
    int col = node % cols, row = node / cols;
    return col * spacing + (row % 2) * (spacing / 2) + x;
    }
public int yCoor(int node)
    {
    int col = node % cols, row = node / cols;
    return row * spacing + (col % 2) * (spacing / 3) + y;
    }
public void draw(Graphics g)
    {
    super.draw(g);
    int xLeft = xVal, xRight = xVal + noOfNodes * BOX_SIZE;
    int yTop = yVal, yBottom = yVal + noOfNodes * BOX_SIZE;

    g.setColor(GRAPH_COLOR);
    for (int row = 0; row <= noOfNodes; row++)
        g.drawLine(xLeft, yRow(row), xRight, yRow(row));
    for (int col = 0; col <= noOfNodes; col++)
        g.drawLine(xCol(col), yTop, xCol(col), yBottom);

    g.setColor(TEXT_COLOR);
    char label = 'A';
    for (int node = 0; node < noOfNodes; node++)
        {
        g.drawString("" + label, xLeft - SIDE_LABEL_OFFSET, yVal + node *
            BOX_SIZE + SIDE_LABEL_OFFSET);
        g.drawString("" + label, xVal + node * BOX_SIZE + TOP_LABEL_OFFSET,
            yTop - TOP_LABEL_OFFSET);
        label++;
        }
    for (int from = 0; from < noOfNodes; from++)
        for (int to = 0; to < noOfNodes; to++)
            if (edges[from][to] && matrixShowing[from][to])
                g.drawString("x", xVal + to * BOX_SIZE + TOP_LABEL_OFFSET,
                    yVal + from * BOX_SIZE + SIDE_LABEL_OFFSET);
    }
public void make()
    {
    char upperLetter = 'A';
    for (int node = 0; node < noOfNodes; node++)
```

```
        {
        nodes[node] = true;
        names[node] = "" + upperLetter++;
        }
    for (int from = 0; from < noOfNodes; from++)
        for (int to = 0; to < noOfNodes; to++)
            {
            edges[from][to] = false;
            matrixShowing[from][to] = false;
            graphShowing[from][to] = false;
            }
    int noOfEdges = randomNo(noOfNodes);
    for (int edge = 0; edge < noOfEdges; edge++)
        {
        int from = randomNo(noOfNodes);
        int to = randomNo(noOfNodes);
        edges[from][to] = edges[to][from] = true;
        if (makeMatrix)
            matrixShowing[from][to] = matrixShowing[to][from] = true;
        else
            graphShowing[from][to] = graphShowing[to][from] = true;
        }
    }
public void create()
    {
    reset(makeMatrix);
    for (int from = 0; from < noOfNodes; from++)
        for (int to = 0; to < from; to++)
            if (edges[from][to])
                {
                if (makeMatrix)
                    {
                    action("Click Edge Nodes", matrixPosition(from, to),
                        nodePosition(from), matrixPosition(to, from),
                        nodePosition(to));
                    graphShowing[from][to] = graphShowing[to][from] = true;
                    }
                else
                    {
                    action("Click Matrix Cells", nodePosition(from),
                        matrixPosition(from, to), nodePosition(to),
                        matrixPosition(to, from));
                    matrixShowing[from][to] = matrixShowing[to][from] = true;
                    }
                }
    }
public void reset(boolean makeMatrix)
    {
    this.makeMatrix = makeMatrix;
    for (int from = 0; from < noOfNodes; from++)
        for (int to = 0; to < from; to++)
            if (edges[from][to])
                if (makeMatrix)
                    {
                    matrixShowing[from][to] = matrixShowing[to][from] = true;
                    graphShowing[from][to] = graphShowing[to][from] = false;
                    }
                else
                    {
                    matrixShowing[from][to] = matrixShowing[to][from] = false;
                    graphShowing[from][to] = graphShowing[to][from] = true;
                    }
    }
public boolean showing(int from, int to)
    {return graphShowing[from][to];}
```

```
    private void action(String prompt, Position start1, Position finish1,
        Position start2, Position finish2)
        {
        click(tracer1, correct1, start1, finish1);
        click(tracer2, correct2, start2, finish2);
        if (Screen.panel.isInteractive())
            Screen.canvas.awaitClick(prompt, true, false);
        Screen.controls.nextStep(false, true);
        move(tracer1, finish1);
        move(tracer2, finish2);
        Screen.canvas.awaitMovingCompletion();
        tracer1.hide(); tracer2.hide();
        correct1.hide(); correct2.hide();
        }
    private void move(DrawableCircle tracer, Position finish)
        {
        Position positions[] = new Position[1];
        positions[0] = finish;
        tracer.moveTo(positions);
        }
    private void click(DrawableCircle tracer, DrawableCircle correct,
        Position start, Position finish)
        {
        tracer.reposition(start);
        tracer.show();
        tracer.setColor(Color.white);
        if (Screen.panel.isInteractive())
             {
             correct.reposition(finish);
             Screen.canvas.acceptClick(correct);
             }
        }
    private Position nodePosition(int node)
        {return new Position(xCoor(node), yCoor(node));}
    private Position matrixPosition(int from, int to)
        {return new Position(xCol(from) + BOX_SIZE / 2, yRow(to) + BOX_SIZE / 2);}
    private int yRow(int row)  {return yVal + row * BOX_SIZE;}
    private int xCol(int col) {return xVal + col * BOX_SIZE;}
    }

class MatrixPanel extends ScreenPanel
    {
    private static final int MAKE_BUTTON = 0;
    private static final int CREATE_BUTTON = 1;
    private static final Position GRAPH_POSITION = new Position(45, 65);

    private CheckboxGroup Group3 = new CheckboxGroup();
       private Checkbox matrix = new Checkbox("Matrix", Group3, true);
       private Checkbox graph = new Checkbox("Graph", Group3, false);
       private String matrixGraphHelp = "Selecting Matrix causes the Make One button to make
a matrix, and the Create Other button to make a graph.  Selecting Graph has the opposite
effect.";
    private SizedComponentPair matrixGraphGroup = new SizedComponentPair(matrix,
        graph, Screen.BUTTON_WIDTH, RADIO_HEIGHT, matrixGraphHelp, "East");
    private MatrixGraph matrixgraph = new MatrixGraph(GRAPH_POSITION, 3, 4);
    private String graphShowHelp[]= {
        "Generates a undirected graph with a random number of edges.",
        "Creates the adjacency matrix that corresponding to undirected graph."};
    private String graphTryHelp[] = {
        "Generates a undirected graph with a random number of edges.",
        "You will be asked to click the two matrix cells that correspond to the edge whose
nodes are highlighted in white, for each edge in the graph."};
    private String matrixShowHelp[]= {
        "Generates a adjacency matrix with a random number of edges.",
        "Creates the undirected graph that corresponds to the adjacency matrix."};
    private String matrixTryHelp[] = {
```

```
        "Generates a adjacency matrix with a random number of edges.",
        "You will be asked to click the pair of nodes that define the edge that corresponds
to the two adjancency matrix entries that are highlighted in white, for each edge in the
matrix. "};

    public MatrixPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, matrixShowHelp, matrixTryHelp, first, allowTry,
            null, false, backgroundColor);
        addPanel(matrixGraphGroup);
        Screen.canvas.add(matrixgraph);
        matrixgraph.make();
        matrixgraph.show();
        }
    synchronized public boolean handleEvent(Event event)
        {
        if (event.target == matrix && event.id == Event.ACTION_EVENT)
            {
            matrixgraph.reset(true);
            shownOrTried[CREATE_BUTTON] = true;
            showMessages = matrixShowHelp;
            tryMessages = matrixTryHelp;
            changeMessages();
            enableButtons();
            return true;
            }
        if (event.target == graph && event.id == Event.ACTION_EVENT)
            {
            matrixgraph.reset(false);
            shownOrTried[CREATE_BUTTON] = true;
            showMessages = graphShowHelp;
            tryMessages = graphTryHelp;
            changeMessages();
            enableButtons();
            return true;
            }
        return super.handleEvent(event);
        }
    public void enableSpecials()
        {matrix.enable(); graph.enable();}
    public void disableSpecials()
        {matrix.disable(); graph.disable();}
    public void changeMessagesSpecials()
        {
        if (running)
            matrixGraphGroup.changeMessage(runningMessage);
        else
            matrixGraphGroup.changeMessage(matrixGraphHelp);
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        if (buttonNumber == MAKE_BUTTON)
            matrixgraph.make();
        else if (buttonNumber == CREATE_BUTTON)
            matrixgraph.create();
        Screen.canvas.repaint();
        }
    }

// Classes that define graphs

class Graph extends DrawableGraph
    {
    private protected static final int RADIUS = 5;
```

```
private protected boolean selectAny = true;
private protected boolean marked[];
private protected Vector currentNextQueue;

private int edgeList[];
private int edgeCount;

public Graph(Position position, int rows, int cols, boolean named)
    {
    super(position, rows, cols, named, 55);
    marked = new boolean[noOfNodes];
    edgeList = new int[noOfNodes * (noOfNodes - 1) / 2];
    }
public int xCoor(int node) {return node % cols * spacing + x;}
public int yCoor(int node) {return node / cols * spacing + y;}
public boolean isSelectable(int node)
    {
    if (selectAny)
        return nodes[node];
    else
        return currentNextQueue.contains(new Integer(node));
    }
public void makeGraph(int more, int less)
    {
    int nodeList[] = new int[noOfNodes];
    int nodeCount = 0;
    char lowerLetter = 'a', upperLetter = 'A';
    for (int node = 0; node < noOfNodes; node++)
        if (nodes[node] = random.nextInt() > 0)
            {
            nodeList[nodeCount] = node;
            if (named)
                if (nodeCount < 26)
                    names[node] = "" + upperLetter++;
                else
                    names[node] = "" + lowerLetter++;
            nodeCount++;
            }
    edgeCount = 0;
    for (int from = 0; from < noOfNodes; from++)
        for (int to = 0; to < noOfNodes; to++)
            edges[from][to] = false;
    boolean nodeConnected[] = new boolean[nodeCount];
    for (int node = 0; node < nodeCount; node++)
        nodeConnected[node] = false;
    nodeConnected[0] = true;
    int notConnected = nodeCount - 1;
    for (int edgeNo = nodeCount - 1 + randomNo(more); edgeNo > 0; edgeNo--)
        {
        int minDistance = Integer.MAX_VALUE, minFrom = -1, minTo = -1;
        for (int from = 0; from < nodeCount; from++)
            if (nodeConnected[from])
                for (int to = 0; to < nodeCount; to++)
                    {
                    boolean acceptEdge;
                    if (notConnected > 0)
                        acceptEdge = !nodeConnected[to];
                    else
                        acceptEdge = nonOverlap(nodeList[from], nodeList[to]);
                    if (acceptEdge)
                        {
                        int thisDistance = distance(nodeList[from], nodeList[to]);
                        if (thisDistance < minDistance)
                            {
                            minFrom = from;
```

```
                            minTo = to;
                            minDistance = thisDistance;
                            }
                        }
                    }
            nodeConnected[minTo] = true;
            addEdge(nodeList[minFrom], nodeList[minTo]);
            notConnected--;
            }
        for (int fewerEdges = randomNo(less); fewerEdges > 0; fewerEdges--)
            {
            int removalEdge = edgeList[randomNo(edgeCount)];
            removeEdge(removalEdge / noOfNodes, removalEdge % noOfNodes);
            }
        }
    private protected Position nodePosition(int node)
        {return new Position(xCoor(node), yCoor(node));}
    private protected Position namePosition(int node)
        {return new Position(xCoor(node) - X_NAME_OFFSET, yCoor(node) - Y_NAME_OFFSET);}

    private void addEdge(int from, int to)
        {
        edges[from][to] = edges[to][from] = true;
        edgeList[edgeCount++] = from * noOfNodes + to;
        }
    private void removeEdge(int from, int to)
        {
        edges[from][to] = edges[to][from] = false;
        edgeCount--;
        }
    private boolean nonOverlap(int from, int to)
        {
        if (edges[from][to] || from == to)
            return false;
        int fromRow = from / cols;
        int toRow = to / cols;
        if (toRow < fromRow)
            {
            fromRow = to / cols;
            toRow = from / cols;
            }
        int fromCol = from % cols;
        int toCol = to % cols;
        if (toCol < fromCol)
            {
            fromCol = to % cols;
            toCol = from % cols;
            }
        if (fromRow == toRow)
            for (int col1 = fromCol; col1 <= toCol; col1++)
                for (int col2 = fromCol; col2 <= col1; col2++)
                    if (edges[fromRow * cols + col1][fromRow * cols + col2])
                        return false;
        if (fromCol == toCol)
            for (int row1 = fromRow; row1 <= toRow; row1++)
                for (int row2 = fromRow; row2 <= row1; row2++)
                    if (edges[row1 * cols + fromCol][row2 * cols + fromCol])
                        return false;
        return true;
        }
    private int distance(int from, int to)
        {
        return Math.abs(from % cols - to % cols) +
            Math.abs(from / cols - to / cols);
        }
    }
```

```
// Classes that define the graph search lesson

class SearchGraph extends Graph
    {
    private static final Position LIST_POSITION = new Position(0, 250);

    private boolean arrowShowing;
    private DrawableCircle circle = new DrawableCircle(new Position(0, 0),
        RADIUS * 2, false, false);
    private DrawableArrow arrow = new DrawableArrow();
    private DrawableString nodeLabel = new DrawableString(new Position(0, 0));
    private TraversalList searchList = new TraversalList(LIST_POSITION, noOfNodes);

    public SearchGraph(Position position, int rows, int cols, boolean named)
        {
        super(position, rows, cols, named);
        }
    public void init()
        {
        Screen.canvas.add(circle);
        Screen.canvas.add(arrow);
        Screen.canvas.add(searchList);
        Screen.canvas.add(nodeLabel);
        searchList.setColor(Color.gray);
        searchList.show();
        circle.setColor(Color.white);
        arrow.setColor(Color.white);
        }
    public void doDepthFirstSearch()
        {
        searchList.init();
        depthFirstSearch(initSearch(true));
        arrow.hide();
        selectAny = true;
        }
    public void doBreadthFirstSearch()
        {
        int depth = 0;
        searchList.init();
        int current = initSearch(false);
        Vector nextQueues[] = new Vector[noOfNodes];
        nextQueues[depth] = new Vector();
        Position positions[] = new Position[1];
        positions[0] = nodePosition(current);
        circle.moveTo(positions);
        traceAndMark(current);
        setNexts(current, nextQueues[depth]);
        nextQueues[++depth] = new Vector();
        while (!empty(nextQueues[depth - 1]))
            {
            current = getNext(nextQueues[depth - 1]);
            positions[0] = nodePosition(current);
            circle.moveTo(positions);
            traceAndMark(current);
            setNexts(current, nextQueues[depth]);
            if (nextQueues[depth - 1].size() == 0)
                nextQueues[++depth] = new Vector();
            }
        circle.hide();
        selectAny = true;
        }
    private void depthFirstSearch(int current)
        {
        if (!marked[current])
            {
```

```
        arrow.setDirection(nodePosition(current), true);
        Position positions[] = new Position[1];
        positions[0] = nodePosition(current);
        arrow.moveTo(positions);
        traceAndMark(current);
        Vector nextQueue = new Vector();
        setNexts(current, nextQueue);
        while (!empty(nextQueue))
            {
            int next = getNext(nextQueue);
            depthFirstSearch(next);
            arrow.setDirection(nodePosition(current), false);
            positions[0] = nodePosition(current);
            arrow.moveTo(positions);
            traceAndMark(current);
            }
        }
    }
private void traceAndMark(int node)
    {
    Screen.canvas.awaitMovingCompletion();
    if (!marked[node])
        {
        Screen.canvas.highlight(new DrawableCross(nodePosition(node)),
           Color.black, false);
        nodeLabel.reposition(namePosition(node));
        searchList.append(names[node], nodeLabel);
        }
    if (!arrowShowing)
        {
        arrow.show();
        arrowShowing = false;
        }
    marked[node] = true;
    }
private int initSearch(boolean directional)
    {
    int first = getStart();
    searchList.init();
    if (directional)
        {
        arrow.reposition(nodePosition(first));
        arrowShowing = false;
        }
    else
        {
        circle.reposition(nodePosition(first));
        circle.show();
        arrowShowing = true;
        }
    selectAny = false;
    return first;
    }
private int getStart()
    {
    int start;
    for (int node = 0; node < noOfNodes; node++)
        marked[node] = false;
    if (Screen.panel.userPicks())
        {
        Screen.canvas.acceptSelection(this);
        if ((start = Screen.canvas.awaitSelection("Click Starting Node")) > 0)
            {
            Screen.prompt.displayPrompt("");
            return start;
            }
```

```
        }
    else
        while (true)
            {
            start = randomNo(noOfNodes);
            if (nodes[start]) return start;
            }
    for (int node = 0; node < noOfNodes; node++)
        if (nodes[node] && !marked[node])
            return node;
    return -1;
    }
private boolean empty(Vector nextQueue)
    {
    boolean isEmpty = true;
    Enumeration list = nextQueue.elements();
    while (list.hasMoreElements())
        {
        Integer object = (Integer) list.nextElement();
        int value = object.intValue();
        if (!marked[value])
            isEmpty =  false;
        }
    return isEmpty;
    }
private void setNexts(int current, Vector nextQueue)
    {
    for (int next = 0; next < noOfNodes; next++)
        if (edges[current][next] && !marked[next])
            if (!nextQueue.contains(new Integer(next)))
                nextQueue.addElement(new Integer(next));
    }
private int getNext(Vector nextQueue)
    {
    int next = -1;
    boolean validResponse = false;
    if (Screen.panel.isInteractive())
        {
        currentNextQueue = nextQueue;
        Screen.canvas.acceptSelection(this);
        if ((next = Screen.canvas.awaitSelection(
            "Click A Correct Next Node")) >= 0)
            {
            nextQueue.removeElement(new Integer(next));
            Screen.interaction.displayMessage("Very Good, That's Correct");
            Screen.prompt.displayPrompt("");
            validResponse = true;
            }
        if (validResponse)
            Screen.progress.oneMoreRight(Screen.thread.button());
        else
            Screen.progress.oneMoreWrong(Screen.thread.button());
        }
    else
        Screen.controls.nextStep(true, true);
    if (!validResponse)
        {
        int choice = randomNo(nextQueue.size());
        Integer node = (Integer) nextQueue.elementAt(choice);
        nextQueue.removeElementAt(choice);
        next = node.intValue();
        }
    return next;
    }
}
```

```
class SearchPanel extends ScreenPanel
    {
    private static final int MAKE_GRAPH_BUTTON = 0;
    private static final int DEPTH_FIRST_BUTTON = 1;
    private static final int BREADTH_FIRST_BUTTON = 2;
    private static final Position GRAPH_POSITION = new Position(80, 45);

    private SearchGraph graph = new SearchGraph(GRAPH_POSITION, 4, 8, true);
    private String randomPickHelp = "Selecting the Random button causes the searches to begin
at a randomly selected node.  Selecting I'll Pick allows you to select the starting node.";
    private String showHelp[] = {
        "Generates a undirected graph with a random number of nodes.",
        "Demonstrates a breadth-first search of the graph on the screen.",
        "Demonstrates a depth-first search of the graph on the screen."};
    private String tryHelp[] = {
        "Generates a undirected graph with a random number of nodes.",
        "You will be asked to click the nodes of this graph in the order of a depth-first
search.",
        "You will be asked to click the nodes of this graph in the order of a breadth-first
search."};

    public SearchPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, randomPickHelp,
            false, backgroundColor);
        Screen.canvas.add(graph);
        graph.init();
        graph.makeGraph(2, 2);
        graph.show();
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        Screen.canvas.clear();
        if (buttonNumber == MAKE_GRAPH_BUTTON)
            graph.makeGraph(2, 2);
        else if (buttonNumber == DEPTH_FIRST_BUTTON)
            graph.doDepthFirstSearch();
        else if (buttonNumber == BREADTH_FIRST_BUTTON)
            graph.doBreadthFirstSearch();
        Screen.canvas.repaint();
        }
    }

// Classes that define the graph categories lesson

class GraphCategories extends Graph
    {
    private int path[];
    private int pathLength = 0;
    private boolean cycles;

    public GraphCategories(Position position, int rows, int cols, boolean named)
        {
        super(position, rows, cols, named);
        path = new int[noOfNodes + 1];
        }
    public boolean isConnected(boolean checkWho, boolean showConnections)
        {
        boolean connected = true;
        int start = getStart(true);
        search(start, checkWho, false);
        if (showConnections)
            Screen.canvas.highlight(new DrawableCircle(nodePosition(start),
                RADIUS * 2, false, false), Color.blue, false);
```

```
        Screen.canvas.repaint();
        for (int node = 0; node < noOfNodes; node++)
            if (nodes[node] && !marked[node])
                {
                if (showConnections)
                    {
                    Screen.canvas.highlight(new DrawableCircle(nodePosition(node),
                        RADIUS * 2, false, true), Color.white, checkWho);
                    }
                connected = false;
                }
        return connected;
        }
    public boolean hasNoCycles(boolean checkWho, boolean markCycles)
        {
        cycles = false;
        int nextStart = getStart(true);
        search(nextStart, checkWho, markCycles);
        while ((nextStart = getStart(false)) >= 0)
            search(nextStart, checkWho, markCycles);
        return !cycles;
        }
    private void search(int current, boolean checkWho, boolean markCycles)
        {
        path[pathLength++] = current;
        int currentLength = pathLength;
        marked[current] = true;
        checkCycles(current, checkWho, markCycles);
        for (int next = 0; next < noOfNodes; next++)
            if (edges[current][next] && !marked[next])
                {
                search(next, checkWho, markCycles);
                pathLength = currentLength;
                }
        }
    private void checkCycles(int node, boolean checkWho, boolean markCycles)
        {
        for (int pathIndex = pathLength - 3; pathIndex >= 0; pathIndex--)
            if (edges[node][path[pathIndex]])
                {
                path[pathLength] = path[pathIndex];
                cycles = true;
                if (markCycles)
                    for (int edge = pathIndex; edge < pathLength; edge++)
                        Screen.canvas.highlight(new DrawableLine
                            (nodePosition(path[edge]), nodePosition(path[edge + 1]), 1),
                            Color.white, checkWho);
                }
        }
    private int getStart(boolean first)
        {
        pathLength = 0;
        if (first)
            for (int node = 0; node < noOfNodes; node++)
                marked[node] = false;
        for (int node = 0; node < noOfNodes; node++)
            if (nodes[node] && !marked[node])
                return node;
        return -1;
        }
    }

class GraphsPanel extends ScreenPanel
    {
    private static final int MAKE_GRAPH_BUTTON = 0;
    private static final int CONNECTED_BUTTON = 1;
```

```
    private static final int CYCLES_BUTTON = 2;
    private static final int TREE_BUTTON = 3;
    private static final Position GRAPH_POSITION = new Position(80, 65);

    private GraphCategories graph = new GraphCategories(GRAPH_POSITION, 4, 8, false);
    private String showHelp[] = {
        "Generates a undirected graph with a random number of nodes.",
        "A message will appear at the top of the screen indicating whether or not the graph
is connected.",
        "A message will appear at the top of the screen indicating whether or not the graph
has cycles.  If there are any cycles, they will be highlighted in white.",
        "A message will appear at the top of the screen indicating whether or not the graph
on the screen is an undirected tree ."};
    private String tryHelp[] = {
        "Generates a undirected graph with a random number of nodes.",
        "You will be asked to click on the button 'All Connected To Blue' if the graph on the
screen is connected, otherwise you are to click on any node unreachable from the node circled
in blue.",
        "You will be asked to click on the button 'It Has No Cycles' if the graph on the
screen contains no cycles, otherwise you are to click on the edge of any cycle in the
graph.",
        "You will be asked to click on the appropriate button at the top of the screen
indicating whether or not the graph on the screen is an undirected tree."};

    public GraphsPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry, null, false,
            backgroundColor);
        Screen.canvas.add(graph);
        graph.makeGraph(3, 3);
        graph.show();
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        Screen.canvas.clear();
        if (buttonNumber == MAKE_GRAPH_BUTTON)
            graph.makeGraph(3, 3);
        else if (buttonNumber == CONNECTED_BUTTON)
            {
            if (!Screen.answer.oneButton(graph.isConnected(true, true),
                "Click Unreachable From Blue or ",
                "All Connected To Blue", "Blue Disconnected From White",
                "That Node Is Unreachable"))
                graph.isConnected(false, true);
            }
        else if (buttonNumber == CYCLES_BUTTON)
            {
            if (!Screen.answer.oneButton(graph.hasNoCycles(true, true), "Click Edge Of Cycle
or",
                "It Has No Cycles", "It Has Cycles", "That Is A Cycle Edge"))
                graph.hasNoCycles(false, true);
            }
        else if (buttonNumber == TREE_BUTTON)
            {
            boolean connected = graph.isConnected(false, false);
            boolean acyclic =  graph.hasNoCycles(false, false);
            boolean isTree = connected && acyclic;
            Screen.answer.twoButtons("Click One Of The Choices",
                "It Is A Tree", "It Is Not A Tree", "It Is A Tree",
                "It Is Not A Tree", isTree, true);
            }
        Screen.canvas.repaint();
        }
    }
```

```
// Classes that define general sort operations

class SwapableList extends DrawableList
    {
    private static final Color BOX_COLOR = new Color(128, 0, 64);
    private static final int VERTICAL_OFFSET = 18;
    private static final int STRING_MOVES = 3;

    private protected int list[];
    private boolean sorted[];
    private Color sortedColor;
    private DrawableString value1 = new DrawableString(new Position(0, 0));
    private DrawableString value2 = new DrawableString(new Position(0, 0));

    public SwapableList(Position position, int size, Color sortedColor)
        {
        super(position, size);
        list = new int[size + 1];
        sorted = new boolean[size + 1];
        for (int index = 1; index <= size; index++)
            sorted[index] = false;
        this.sortedColor = sortedColor;
        color = BOX_COLOR;
        }
    public void draw(Graphics g)
        {
        super.draw(g);
        int numberX = x + INSET_X;
        for (int i = 1; i <= count; i++)
            {
            if (sorted[i])
                g.setColor(sortedColor);
            else
                g.setColor(Color.white);
            if (list[i] >= 0)
                g.drawString("" + list[i], numberX, y + OFFSET_Y);
            numberX += DELTA_X;
            }
        value1.drawObject(g);
        value2.drawObject(g);
        }
    public void setValues(int values[], int size)
        {
        for (int index = 1; index <= size; index++)
            {list[index] = values[index]; sorted[index] = false;}
        count = size;
        }
    public void markSorted(int index) {sorted[index] = true;}
    public void nullify(int index) {list[index] = -1;}
    public void startSwap(int left, int right)
        {
        move(left, right, value1);
        move(right, left, value2);
        list[left] = -list[left];
        list[right] = -list[right];
        }
    public void finishSwap(int left, int right)
        {
        int leftValue = -list[left];
        int rightValue = -list[right];
        list[left] = rightValue;
        list[right] = leftValue;
        value1.hide();
        value2.hide();
        }
```

```
    public void startRotate(int left, int right)
        {
        move(right, left, value1);
        int temp = list[right];
        for (int index = right; index > left; index--)
            list[index] = list[index - 1];
        list[left] = -temp;
        }
    public void finishRotate(int left, int right)
        {
        list[left] = -list[left];
        value1.hide();
        }
    private void move(int from, int to, DrawableString value)
        {
        Position positions[] = new Position[STRING_MOVES];
        value.show();
        value.setString("" + list[from]);
        value.reposition(ListStringPosition(from, to, 0));
        for (int move = 0; move < STRING_MOVES; move++)
            positions[move] = ListStringPosition(from, to, move + 1);
        value.moveTo(positions);
        }
    private Position ListStringPosition(int from, int to, int move)
        {
        switch (move)
            {
            case 0:
                return new Position(x + INSET_X + (from - 1) * DELTA_X, y + OFFSET_Y);
            case 1:
                return new Position(x + INSET_X + (from - 1) * DELTA_X, y +
                    OFFSET_Y - VERTICAL_OFFSET);
            case 2:
                return new Position(x + INSET_X + (to - 1) * DELTA_X, y +
                    OFFSET_Y - VERTICAL_OFFSET);
            case 3:
                return new Position(x + INSET_X + (to - 1) * DELTA_X, y + OFFSET_Y);
            }
        return null;
        }
    }

class SwapableArray extends Drawable
    {
    private static final int DELTA_X = 17;
    private static final int WIDTH = 10;
    private static final int X_CENTER = (DELTA_X - WIDTH) / 2;
    private static final int Y_LINE = 222;
    private static final int Y_ARROW = 230;
    private static final int MAX_LINES = 29;
    private static final int PAUSE = 1000;
    private static final Color BAR_COLOR = new Color(128, 64, 0);
    private static final Color SORTED_COLOR = new Color(0, 72, 0);

    private int array[], size, yIncrement = 2;
    private boolean sorted[];
    private int leftValue, rightValue;
    private int lineCount = 0;
    private DrawableArrow leftArrow = new DrawableArrow();
    private DrawableArrow rightArrow = new DrawableArrow();
    private QuicksortBar bar = new QuicksortBar(new Position(0, 0));
    private HorizontalLine lines[] = new HorizontalLine[MAX_LINES];
    private DrawableBar bar1 = new DrawableBar(new Position(0, 0));
    private DrawableBar bar2 = new DrawableBar(new Position(0, 0));

    public SwapableArray(Position position, int size)
```

```
    {
    x = position.x();
    y = position.y();
    this.size = size;
    }
public void draw(Graphics g)
    {
    for (int index = 1; index <= size; index++)
        {
        g.setColor(BAR_COLOR);
        if (sorted[index])
            g.setColor(SORTED_COLOR);
        g.fillRect(xPos(index), yPos(index), WIDTH, height(index));
        }
    bar.drawObject(g);
    bar1.drawObject(g);
    bar2.drawObject(g);
    g.setColor(SORTED_COLOR);
    if (!Screen.panel.isInteractive())
        {
        leftArrow.drawObject(g);
        rightArrow.drawObject(g);
        }
    for (int line = 0; line < lineCount; line++)
        lines[line].drawObject(g);
    }
public void setValues(int array[], int size)
    {
    this.array = array;
    this.size = size;
    sorted = new boolean[size + 1];
    for (int index = 1; index <= size; index++)
        sorted[index] = false;
    lineCount = 0;
    }
public void markSorted(int index)
    {sorted[index] = true;}
public void startSwap(int left, int right)
    {
    activateBar(left, bar1);
    activateBar(right, bar2);
    if (Screen.panel.isInteractive())
        Screen.canvas.awaitClick("Click Bars To Swap", true, false);
    else
        Screen.controls.nextStep(true, true);
    Position rightPosition = new Position(xPos(right), yPos(left));
    moveBar(left, rightPosition, bar1);
    leftValue = array[left];
    Position leftPosition = new Position(xPos(left), yPos(right));
    moveBar(right, leftPosition, bar2);
    rightValue = array[right];
    array[left] = 0;
    array[right] = 0;
    bar1.show(); bar2.show();
    }
public void finishSwap(int left, int right)
    {
    array[left] = rightValue;
    array[right] = leftValue;
    bar1.hide();
    bar2.hide();
    }
public void startRotate(int left, int right)
    {
    activateBar(right, bar1);
    if (Screen.panel.isInteractive())
```

```
        {
        Screen.canvas.awaitClick("Click The Bar To Insert", true, false);
        bar1.show();
        try {Thread.sleep(PAUSE);} catch (InterruptedException e) {}
        activateBar(left, bar2);
        Screen.canvas.awaitClick("Click Location To Insert", true, false);
        }
    else
        Screen.controls.nextStep(true, true);
    Position leftPosition = new Position(xPos(left), yPos(right));
    moveBar(right, leftPosition, bar1);
    leftValue = array[right];
    for (int index = right; index > left; index--)
        array[index] = array[index - 1];
    array[left] = 0;
    bar1.show();
    }
public void finishRotate(int left, int right)
    {
    array[left] = leftValue;
    bar1.hide();
    bar2.hide();
    }
public void doCopy(int from, int to, SwapableArray copy)
    {
    int temp = array[from];
    activateBar(from, bar1);
    if (Screen.panel.isInteractive())
        Screen.canvas.awaitClick("Click Bar To Move", true, false);
    else
        Screen.controls.nextStep(true, true);
    copy.array[to] = temp;
    Position toPosition = new Position(copy.xPos(to), copy.yPos(to));
    array[from] = copy.array[to] = 0;
    moveBar(from, toPosition, bar1);
    bar1.show();
    Screen.canvas.awaitMovingCompletion();
    copy.array[to] = temp;
    bar1.hide();
    }
public void toggleHeight()
    {
    if (yIncrement == 2)
        yIncrement = 1;
    else
        yIncrement = 2;
    }
public void positionArrows(int left, int right)
    {
    leftArrow.reposition(new Position(xPos(left) + WIDTH, Y_ARROW));
    leftArrow.setDirection(new Position(xPos(left + 1), Y_ARROW), true);
    rightArrow.reposition(new Position(xPos(right), Y_ARROW));
    rightArrow.setDirection(new Position(xPos(right - 1), Y_ARROW), true);
    }
public void moveArrows(int left, int right)
    {
    Position positions[] = new Position[1];
    positions[0] = new Position(xPos(left)+ WIDTH, Y_ARROW);
    leftArrow.moveTo(positions);
    positions[0] = new Position(xPos(right), Y_ARROW);
    rightArrow.moveTo(positions);
    Screen.canvas.awaitMovingCompletion();
    }
public void showArrows()
    {
    leftArrow.show();
```

```
            rightArrow.show();
            bar.show();
            }
        public void hideArrows()
            {
            leftArrow.hide();
            rightArrow.hide();
            bar.hide();
            }
        public void setBar(int first, int last)
            {
            bar.reposition(new Position(xPos(first), yPos(first)));
            bar.setDimensions(height(first), xPos(first), xPos(last));
            }
        public void makeLine(int first, int last, int level)
            {
            lines[lineCount] = new HorizontalLine(xPos(first), xPos(last) + WIDTH, Y_LINE +
                level *2);
            lines[lineCount++].show();
            }
        private void activateBar(int from, DrawableBar bar)
            {
            bar.reposition(new Position(xPos(from), yPos(from)));
            bar.setHeight(height(from));
            if (Screen.panel.isInteractive())
                Screen.canvas.acceptClick(bar);
            else
                {
                Screen.controls.nextStep(false, true);
                bar.show();
                }
            }
        private void moveBar(int from, Position to, DrawableBar bar)
            {
            Position positions[] = new Position[1];
            positions[0] = to;
            bar.moveTo(positions);
            }
        private int height(int index) {return array[index] * yIncrement;}
        private int xPos(int index) {return x + X_CENTER + (index - 1) * DELTA_X;}
        private int yPos(int index) {return y - height(index);}
        }

    abstract class SortPanel extends ScreenPanel
        {
        private static final Color SORTED_COLOR = new Color(255, 255, 172);
        private static final Position LIST_POSITION = new Position(18, 240);
        private protected static final Position ARRAY_POSITION =
            new Position(18, 220);
        private protected static final int SIZE = 30;

        private protected SwapableArray swapableArray = new SwapableArray(ARRAY_POSITION, SIZE);
        private protected SwapableList swapableList = new SwapableList(LIST_POSITION, SIZE,
            SORTED_COLOR);
        private protected int numericArray[] = new int[SIZE + 1];

        public void init(String names[], String showHelp[], String tryHelp[],
            int first, boolean allowTry, Color backgroundColor)
            {
            super.init(names, showHelp, tryHelp, first, allowTry, null, false,
                backgroundColor);
            makeArray();
            Screen.canvas.add(swapableArray);
            Screen.canvas.add(swapableList);
            swapableArray.show();
            swapableList.show();
```

```
        swapableList.setColor(backgroundColor);
        Screen.canvas.repaint();
        }
    public void makeArray()
        {
        for (int index = 1; index <= SIZE; index++)
            numericArray[index] = getRandom(10, 100);
        swapableArray.setValues(numericArray, SIZE);
        swapableList.setValues(numericArray, SIZE);
        }
    public void swap(int i, int j)
        {
        if (i == j) return;
        swapableArray.startSwap(i, j);
        swapableList.startSwap(i, j);
        Screen.canvas.awaitMovingCompletion();
        swapableList.finishSwap(i, j);
        swapableArray.finishSwap(i, j);
        }
    public void markSorted(int index)
        {
        swapableArray.markSorted(index);
        swapableList.markSorted(index);
        }
    }

// Class that defines quadratic sorting lesson

class QuadSortPanel extends SortPanel
    {
    private static final int MAKE_ARRAY_BUTTON = 0;
    private static final int BUBBLE_BUTTON = 1;
    private static final int INSERTION_BUTTON = 2;
    private static final int SELECTION_BUTTON = 3;

    private String showHelp[] = {
        "Generates an array with a random set of values.",
        "Sorts the array using the bubble sort algorithm.",
        "Sorts the array using the insertion sort algorithm.",
        "Sorts the array using the selection sort algorithm."};
    private String tryHelp[] = {
        "Generates an array with a random set of values.",
        "You will be asked to click the pair of bars to swap at each step to sort the array
using the bubble sort algorithm.",
        "You will be asked to click the bar to be inserted and the location at which it
should be inserted at each step of the insertion sort algorithm.",
        "You will be asked to click the pair of bars to swap at each step to sort the array
using the selection sort algorithm."};

    public QuadSortPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry,
            backgroundColor);
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        if (buttonNumber == MAKE_ARRAY_BUTTON)
            makeArray();
        else if (buttonNumber == BUBBLE_BUTTON)
            bubbleSort();
        else if (buttonNumber == INSERTION_BUTTON)
            insertionSort();
        else if (buttonNumber == SELECTION_BUTTON)
            selectionSort();
```

```
        Screen.canvas.repaint();
        }
    private void bubbleSort()
        {
        int i;
        for (i = 1; i < SIZE; i++)
            {
            for (int j = SIZE; j > i; j--)
                if (numericArray[j] < numericArray[j - 1])
                    swap(j, j-1);
            markSorted(i);
            }
        markSorted(i);
        }
    private void insertionSort()
        {
        int i;
        markSorted(1);
        for (i = 2; i <= SIZE; i++)
            {
            int j = i - 1;
            while (numericArray[i] < numericArray[j])
                if (j-- <= 1) break;
            if (i > j + 1)
                {
                swapableArray.startRotate(j + 1, i);
                swapableList.startRotate(j + 1, i);
                Screen.canvas.awaitMovingCompletion();
                swapableList.finishRotate(j + 1, i);
                swapableArray.finishRotate(j + 1, i);
                }
            markSorted(i);
            }
        markSorted(i - 1);
        }
    private void selectionSort()
        {
        int i;
        for (i = 1; i < SIZE; i++)
            {
            int minIndex = i;
            for (int j = i + 1; j <= SIZE; j++)
                if (numericArray[j] < numericArray[minIndex])
                    minIndex = j;
            swap(i, minIndex);
            markSorted(i);
            }
        markSorted(i);
        }
    }

// Class that defines efficient sorting lesson

class EffSortPanel extends SortPanel
    {
    private static final Position copyPosition = new Position(18, 100);
    private static final int MAKE_ARRAY_BUTTON = 0;
    private static final int MERGE_BUTTON = 1;
    private static final int QUICK_BUTTON = 2;
    private static final int SHELL_BUTTON = 3;

    private int copy[] = new int[SIZE + 1];
    private SwapableArray swapableCopy = new SwapableArray(copyPosition, SIZE);
    private String showHelp[] = {
        "Generates an array with a random set of values.",
        "Sorts the array using the merge sort algorithm.",
```

```
        "Sorts the array using the quick sort algorithm."};
    private String tryHelp[] = {
        "Generates an array with a random set of values.",
        "You will be asked to click the bar to move at each step to sort the array using the
merge sort algorithm.",
        "You will be asked to click the pair of bars to swap at each step to sort the array
using the quick sort algorithm."};

    public EffSortPanel(String names[], int first, boolean allowTry,
        Color backgroundColor)
        {
        super.init(names, showHelp, tryHelp, first, allowTry,
            backgroundColor);
        swapableCopy.toggleHeight();
        Screen.canvas.add(swapableCopy);
        }
    public void buttonClicked(int buttonNumber)
        {
        Screen.interaction.displayMessage("");
        if (buttonNumber == MAKE_ARRAY_BUTTON)
            makeArray();
        else if (buttonNumber == MERGE_BUTTON)
            {
            swapableArray.toggleHeight();
            for (int index = 0; index < SIZE; index++)
                copy[index] = 0;
            swapableCopy.setValues(copy, SIZE);
            swapableCopy.show();
            mergeSort(1, SIZE, 1);
            swapableCopy.hide();
            swapableArray.toggleHeight();
            }
        else if (buttonNumber == QUICK_BUTTON)
            quickSort(1, SIZE);
        else if (buttonNumber == SHELL_BUTTON)
            ShellSort(1, SIZE);
        Screen.canvas.repaint();
        }
    private void doCopy(int from, int to)
        {
        swapableArray.doCopy(from, to, swapableCopy);
        swapableList.nullify(from);
        }
    private void merge(int first, int middle, int last)
        {
        int index = first, left = first, right = middle + 1;
        while (left <= middle && right <= last)
            if (numericArray[left] < numericArray[right])
                doCopy(left++, index++);
            else
                doCopy(right++, index++);
        while (left <= middle)
            doCopy(left++, index++);
        while (right <= last)
            doCopy(right++, index++);
        }
    private void mergeSort(int first, int last, int level)
        {
        if (first < last)
            {
            int middle = (first + last) / 2;
            mergeSort(first, middle, level + 1);
            mergeSort(middle + 1, last, level + 1);
            merge(first, middle, last);
            Position positions[] = new Position[1];
            positions[0] = ARRAY_POSITION;
```

```
            swapableCopy.moveTo(positions);
            Screen.canvas.awaitMovingCompletion();
            for (int index = first; index <= last; index++)
                {
                numericArray[index] = copy[index];
                copy[index] = 0;
                }
            swapableCopy.reposition(copyPosition);
            swapableList.setValues(numericArray, SIZE);
            for (int index = first; index <= last; index++)
                if (level == 1)
                    markSorted(index);
            swapableCopy.makeLine(first, last, level);
            }
        }
    private int split(int first, int last)
        {
        int key = numericArray[first];
        swapableArray.setBar(first, last);
        int left = first + 1, right = last;
        do
            {
            swapableArray.positionArrows(left, right);
            while (left <= right && key >= numericArray[left]) left++;
            while (left <= right && key < numericArray[right]) right--;
            swapableArray.moveArrows(left, right);
            if (left < right)  swap(left++, right--);
            }
        while (left <= right);
        swap(first, right);
        return right;
        }
    private void quickSort(int first, int last)
        {
        swapableArray.showArrows();
        if (first == last)
            markSorted(first);
        else if (first < last)
            {
            int pivot = split(first, last);
            markSorted(pivot);
            quickSort(first, pivot - 1);
            quickSort(pivot + 1, last);
            }
        swapableArray.hideArrows();
        }
    private void ShellSort(int first, int last)
        {
        int top = first, bottom = last;
        for (int distance = bottom / 2; distance > 0; distance /= 2)
            for (int current = top + distance; current <= bottom; current++)
                for (int position = current; position >= top + distance &&
                    numericArray[position] < numericArray[position - distance];
                    position -= distance)
                    swap(position, position - distance);
        }
    }
```

## A.2 JavaScript control code

```
var appletRunning = false;
var codeDisplayed = false;
var screen;
var defNo = 0;
```

```
var studentName;

function menu1Line(segment, longName, shortName)
  {
  if ("" + segment == screen.applet)
    this.menu1.document.write('   <FONT COLOR="#000080">' + shortName + '</FONT><BR>');
  else
    this.menu1.document.write('   <A HREF="javascript:parent.setScreen(' + segment + ',' +
"'" + longName + "'" +'); parent.load(1);">' + shortName + '</A><BR>');
  }

function loadPage()
  {
  screen = new Object();
  screen.applet = 0;
  if (this.applet.document.screen != null)
    this.applet.document.screen.stop();
  studentName = prompt("Please enter your name: ", "");
  if (studentName == null)
      studentName = "Anonymous";
  this.title.open("title.html","title");
  this.main.open("intro.html","main");
  this.menu2a.open("white.html","menu2a");
  this.menu2b.open("white.html","menu2b");
  makeMenu1();
  }
function makeMenu1()
  {
  this.menu1.document.write('<HTML><BODY BGCOLOR="#C0C0C0">');
  this.menu1.document.write('<P><PRE><FONT FACE="Arial"><B><FONT COLOR="#000080">Week 1 -
Graphs</FONT></B><BR>');
  menu1Line(7, "Graph Representation", "Representation");
  menu1Line(8, "Graphs Searches", "Searches");
  menu1Line(9, "Categorizing Graphs", "Categorizing");
  this.menu1.document.write('</FONT></PRE><PRE><FONT FACE="Arial"><B><FONT
COLOR="#000080">Week 2 - Trees</FONT></B><BR>');
  menu1Line(1, "Binary Tree Traversals", "Traversals");
  menu1Line(4, "Binary Search Trees", "Search Trees");
  menu1Line(2, "Priority Queue with Heap", "Priority Queue");
  menu1Line(5, "Height Balanced Trees", "Height Balancing");
  menu1Line(6, "Categorizing Binary Trees", "Categorizing");
  this.menu1.document.write('</FONT></PRE><PRE><FONT FACE="Arial"><B><FONT
COLOR="#000080">Week 3 - Sorting</FONT></B><BR>');
  menu1Line(3, "Heap Sort", "Heap Sort", 0);
  menu1Line(10, "Quadratic Sorts", "Quadratic Sorts");
  menu1Line(11, "Efficient Sorts", "Efficient Sorts");
  this.menu1.document.write('</FONT></PRE></BODY></HTML>');
  this.menu1.document.close();
  }
function makeMenu2()
  {
  var button1, button2, action1, action2;
  if (!codeDisplayed)
    {
    button1 = "     Ada Code     ";
    action1 = "parent.loadMain(2)";
    }
  else
    {
    button1 = "   Explanation   ";
    action1 = "parent.loadMain(1);"
    }
  if (appletRunning)
    {
    button2 = " Stop Visualization";
    action2 = "parent.visualClicked(2)";
```

```
    }
  else
    {
    button2 = "Start Visualization";
    action2 = "parent.visualClicked(1);"
    }
  this.menu2a.document.write('<HTML><BODY BGCOLOR="#FFFFFF"><FORM><CENTER>' +
    '<INPUT TYPE="button" VALUE="' + button1 + '" onClick="' + action1 + '">' +
    '</CENTER></FORM></BODY><HTML>');
  this.menu2a.document.close();
  this.menu2b.document.write('<HTML><BODY BGCOLOR="#FFFFFF"><FORM><CENTER>' +
    '<INPUT TYPE="button" VALUE="' + button2 + '" onClick="' + action2 + '">' +
    '</CENTER></FORM></BODY><HTML>');
  this.menu2b.document.close();
  }
function setScreen(applet, title)
  {
  if (appletRunning)
    if (!confirm('Changing Segments Will Stop Visualization, Do You Wish To Proceed?'))
      return;
    else
      visualization();
  screen.applet = applet;
  screen.title = title;
  }
function visualClicked(choice)
  {
  if (choice == 1)
    loadApplet();
  else if (choice == 2)
    unloadApplet();
  makeMenu2();
  }
function loadMain(choice)
  {
  if (choice == 1)
    {
    open("text" + screen.applet + ".html", "main");
    codeDisplayed = false;
    }
  else if (choice == 2)
    {
    this.main.open("code" + screen.applet + ".html", "main");
    codeDisplayed = true;
    }
  makeMenu2();
  }
function load(choice)
  {
  this.title.document.write('<HTML><BODY BGCOLOR="#FFFFFF" text="#000080"><H3><CENTER><FONT
FACE="Arial">Interactive Data Structure Visualizations<BR>' +
    screen.title + '</FONT></CENTER></H4></BODY></HTML>');
  this.title.document.close();
  loadMain(choice);
  makeMenu1();
  }
function loadApplet()
  {
  this.applet.document.write(
    '<BODY><APPLET NAME="screen" ARCHIVE="idsv.jar" CODE="Animator.class" WIDTH=0 HEIGHT=0>'
    + '<PARAM NAME="studentName" VALUE="' + studentName +
    '">' + '<PARAM NAME="panelNo" VALUE=' + screen.applet +
    '><PARAM NAME="allowTry" VALUE="true">' +
    '<PARAM NAME="showProgress" VALUE="true">' +
    '</APPLET></BODY></HTML>');
  this.applet.document.close();
```

```
  eventLog = studentName + "_";
  appletRunning = true;
  }
function unloadApplet()
  {
//  if (this.applet.document.screen != null)
//    this.applet.document.screen.stop();
  this.applet.document.write('<BODY></BODY></HTML>');
  this.applet.document.close();
  appletRunning = false;
  }
function define(term, number)
  {
  var def;
  if (term == "Almost_Complete_Binary_Tree")
    def = "An almost complete binary tree is a tree of height <I>n</I> that contains all
possible nodes on levels 1 through <I>n</I>-1.  On level <I>n</I> if a node is missing, all
its right siblings must be missing also.";
  else if (term == "AVL_Tree")
    def = "An AVL tree, named for Adelson-Velskii and Landis, is a binary search tree that is
balanced--meaning the balance factor of every node is either -1, 0, or 1.";
  else if (term == "Balance_Factors")
    def = "The balance factor of the node of a binary tree is computed by subtracting the
height of its left subtree from the height of its right subtree.";
  else if (term == "Balanced_Tree")
    def = "A balanced tree is a binary tree in which the balance factor of every node is
either -1, 0, or 1.";
  else if (term == "Binary_Search")
    def = "A binary search examines the middle value of an array first and repeats the
process of the left half is the value to be found is less and the right half if it is
greater. It's efficiency is O(log n).";
  else if (term == "Binary_Search_Tree")
    def = "A binary search tree is a binary tree each of whose nodes contains a value that is
greater than all the values in the nodes in its left subtree and less than all the values in
the nodes in its right subtree.";
  else if (term == "Binary_Tree_Traversal")
    def = "Traversing a binary tree means visiting each node of that tree.  There are several
different kinds of binary tree traversals.  Different kinds of traversals visit the nodes of
the tree in different orders.";
  else if (term == "Breadth-First")
    def = "A breadth-first tree traversal or graph search always visits both the siblings of
a node, or all of its neighbors in a graph, before it visits any of its grandchildren, or
neighbor's neighbors in a graph.";
  else if (term == "Complete_Binary_Tree")
    def = "A complete binary tree is a tree of height <I>n</I> that contains all possible
nodes on levels 1 through <I>n</I>. A complete tree of height n always has exactly
2<I><SUP>n</SUP></I>-1 nodes.  Complete binary trees are sometimes referred to a full
trees.";
  else if (term == "Connected")
    def = "An undirected graph is connected if there is a path between every pair of nodes.";
  else if (term == "Cycle")
    def = "A cycle is a path in which the source of first edge is the same as the destination
of the last edge.";
  else if (term == "Depth-First")
    def = "A depth-first tree traversal or graph search always moves as far down one branch
of the tree or graph as it can. After visiting the child or neighbor in a graph, it visits
the grandchild or neighbor's neighbor and so on until it reaches a leaf node. Each of the
depth-first traversals is naturally recursive, because a stack is needed to maintain a record
of all of the nodes visited along any particular branch.";
  else if (term == "Edge")
    def = "In an <I>un</I>directed graph, an edge is an <I>un</I>ordered pair of nodes.";
  else if (term == "First-In_First-Out_Queue")
    def = "A first-in first-out queue is a data structure that has two operations, enqueue,
adding to the queue and dequeue, removing from the queue. The essential property of such a
queue is that the first element enqueued in the first element dequeued.";
  else if (term == "Heap")
```

```
    def = "A heap is an almost complete binary tree with the property that the value in each
node is greater than the value in both children. Because a heap is an almost complete binary
tree, each node of such a tree with n nodes can be mapped to an element of an array with n
elements. A heap has the property that the root node always contains the largest value.";
  else if (term == "Height")
    def = "The height or depth of a binary tree is measured by the number of nodes along its
longest branch.  Sometimes branches are counted instead of nodes resulting in a value one
less.  Height can be defined recursively as follows:  Empty trees have height 0, the height
of a nonempty trees is the maximum of its two subtrees plus 1.";
  else if (term == "Height_Balanced_Tree")
    def = "A binary tree is height balanced-<I>n</I> if the balance factors of each of its
nodes are between -<I>n</I> and <I>n</I>, inclusive.";
  else if (term == "Inorder_Binary_Tree_Traversal")
    def = "A inorder binary tree traversal uses the order LDR, left, data--extracting the
data, and then right. When an arithmetic expression binary tree is traversed in inorder the
arithmetic expression is generated in infix notation.";
  else if (term == "Keyed_Table")
    def = "A keyed table is a table that is ordered according to some key.  The most
fundamental operations of a keyed table are the ability to add elements, delete them and find
them by providing the key.";
  else if (term == "Level_Order_Binary_Tree_Traversal")
    def = "A level order traversal always visits the nodes of the tree, level by level. It
begins at the root and proceeds top down. It visits the nodes on each level from left to
right.";
  else if (term == "Merge")
    def = "Two sorted arrays are merged by selecting the smallest of the next values of each
and appending it to a new array.  The new array will contain all the values of the original
two and be in sorted order.";
  else if (term == "Path")
    def = "A sequence of edges in which the destination of each edge is the source of the
next edge.";
  else if (term == "Postorder_Binary_Tree_Traversal")
    def = "A postorder binary tree traversal uses the order LRD, first left, then right, then
data--extracting the data. When an arithmetic expression binary tree is traversed in
postorder the arithmetic expression is generated in postfix notation.";
  else if (term == "Preorder_Binary_Tree_Traversal")
    def = "A preorder binary tree traversal uses the order DLR, data--extracting the data,
left and then right. When an arithmetic expression binary tree is traversed in preorder the
arithmetic expression is generated in prefix notation.";
  else if (term == "Priority_Queue")
    def = "Like a first-in first-out queue, a priority queue has two fundamental operations,
entering the queue called enqueue, and leaving the queue, called dequeue. Unlike a first-in
first-out queue, the criteria for deciding which element to dequeue is based on the priority
of the element, not the order of arrival.";
  else if (term == "Right_Rotation")
    def = "Right rotating an array means shifting all values one position to the right, and
moving the value inthe last position into the first position.";
  else if (term == "Stack")
    def = "A stack is a data structure that has two operations, push, adding to the stack and
pop, removing from the stack. The essential property of such a stack is that the last element
pushed in the first element popped.";
  else if (term == "Swap")
    def = "Swapping two values involves interchanging them.  A swap involves three
assignments and a temporary variable.";
  else if (term == "Undirected_Graph")
    def = "An undirected graph is an ordered pair of nodes, or vertices, and edges.  The
edges are <I>un</I>ordered pairs of nodes.";
  else if (term == "Undirected_Tree")
    def = "An undirected tree is an undirected graph that is connected and contains no
cycles.";
  termName = "";
  for (i = 0; i < term.length; i++)
    if (term.charAt(i) == '_')
      termName += ' ';
    else
      termName += term.charAt(i);
```

```
  var height = 110 + def.length / 3;
  win = window.open("", "window"+defNo++, "width=400,height="+height);
  win.document.write('<HTML><HEAD><TITLE>' + termName + ' Definition</TITLE></HEAD><BODY
TEXT="#800000">');
  win.document.write('<H4><CENTER><FONT COLOR="#000000">' + termName  +
'</FONT></CENTER></H4>');
  win.document.write('<P>' + def + '</P>');
  win.document.write('<FORM><CENTER><INPUT TYPE="button" VALUE="OK"
onClick="self.close();"></CENTER></FORM></BODY></HTML>');
  win.document.close();
  }
```

## A.3 Ada code

```
-- Binary tree traversal lesson code

GENERIC
  TYPE Elements IS PRIVATE;
  WITH PROCEDURE Put(Item: IN Elements)IS <>;
PACKAGE Binary_Tree_Package IS
  TYPE Trees IS LIMITED PRIVATE;
  PROCEDURE Make (Root: OUT Trees; Left, Right: IN Trees;
    Data: IN Elements);
  PROCEDURE Preorder (Root: IN Trees);
  PROCEDURE Inorder (Root: IN Trees);
  PROCEDURE Postorder (Root: IN Trees);
  PROCEDURE Level_Order (Root: IN Trees);
  Empty_Tree: CONSTANT Trees;
PRIVATE
  TYPE Nodes IS
    RECORD
      Left: Trees;
      Data: Elements;
      Right: Trees;
    END RECORD;
  TYPE Trees IS ACCESS Nodes;
  Empty_Tree: CONSTANT Trees := NULL;
END Binary_Tree_Package;

WITH FIFO_Queue_Package;
PACKAGE BODY Binary_Tree_Package IS
  PROCEDURE Make (Root: OUT Trees; Left, Right: IN Trees;
    Data: IN Elements) IS
  BEGIN
    Root := New Nodes'(Left, Data, Right);
  END Make;

  PROCEDURE Preorder (Root: IN Trees) IS
  BEGIN
    IF Root /= NULL THEN
      Put(Root.Data);
      Preorder(Root.Left);
      Preorder(Root.Right);
    END IF;
  END Preorder;

  PROCEDURE Inorder (Root: IN Trees) IS
  BEGIN
    IF Root /= NULL THEN
      Inorder(Root.Left);
      Put(Root.Data);
      Inorder(Root.Right);
    END IF;
  END Inorder;
```

```
  PROCEDURE Postorder (Root: IN Trees) IS
  BEGIN
    IF Root /= NULL THEN
      Postorder(Root.Left);
      Postorder(Root.Right);
      Put(Root.Data);
    END IF;
  END Postorder;

  PROCEDURE Level_Order (Root: IN Trees) IS
    PACKAGE Trees_Queue IS NEW
      FIFO_Queue_Package(Trees);
    USE Trees_Queue;
    Tree: Trees;
    Queue: Queues;
  BEGIN
    Enqueue(Queue,Root);
    WHILE NOT Empty(Queue) LOOP
      Dequeue(Queue,Tree);
      Put(Tree.Data);
      IF Tree.Left /= NULL THEN
        Enqueue(Queue,Tree.Left);
      END IF;
      IF Tree.Right /= NULL THEN
        Enqueue(Queue,Tree.Right);
      END IF;
    END LOOP;
  END Level_Order;
END Binary_Tree_Package;

-- Queue used by level order traversal

GENERIC
  TYPE Elements IS PRIVATE;
PACKAGE FIFO_Queue_Package IS
  TYPE Queues IS LIMITED PRIVATE;
  FUNCTION Empty(Queue: Queues)
    RETURN Boolean;
  PROCEDURE Enqueue(Queue: IN OUT Queues;
    Item: IN Elements);
  PROCEDURE Dequeue(Queue: IN OUT Queues;
    Item: OUT Elements);
  Empty_Queue, Full_Queue: EXCEPTION;
PRIVATE
  Size: CONSTANT := 100;
  TYPE Lists IS ARRAY(1..Size) OF Elements;
  TYPE Queues IS
    RECORD
      Front: Integer RANGE 1..Size := 1;
      Back: Integer RANGE 1..Size := 1;
      List: Lists;
    END RECORD;
END FIFO_Queue_Package;

PACKAGE BODY FIFO_Queue_Package IS
  FUNCTION Empty(Queue: Queues)
    RETURN Boolean IS
  BEGIN
    RETURN Queue.Front = Queue.Back;
  END Empty;

  PROCEDURE Enqueue(Queue: IN OUT Queues;
    Item: IN Elements) IS
  BEGIN
    IF Queue.Back MOD Size + 1 /=
```

```
    Queue.Front THEN
      Queue.List(Queue.Back) := Item;
      Queue.Back := Queue.Back MOD Size+ 1;
    ELSE
      RAISE Full_Queue;
    END IF;
  END Enqueue;

  PROCEDURE Dequeue(Queue: IN OUT Queues;
    Item: OUT Elements) IS
  BEGIN
    IF Queue.Back /= Queue.Front THEN
      Item := Queue.List(Queue.Front);
      Queue.Front :=
        Queue.Front MOD Size + 1;
    ELSE
      RAISE Empty_Queue;
    END IF;
  END Dequeue;
END FIFO_Queue_Package;

-- Priority queue lesson code

GENERIC
  TYPE Elements IS PRIVATE;
  TYPE Priorities IS PRIVATE;
  WITH FUNCTION Get_Rank(Element: Elements)
    RETURN Priorities;
  WITH FUNCTION "<"(Left,Right: Priorities)
    RETURN Boolean IS <>;
PACKAGE Priority_Queue_Package IS
  TYPE Queues IS LIMITED PRIVATE;
  FUNCTION Empty(Queue: Queues)
    RETURN Boolean;
  PROCEDURE Enqueue(Queue: IN OUT Queues;
    Item: IN Elements);
  PROCEDURE Dequeue(Queue: IN OUT Queues;
    Item: OUT Elements);
  Empty_Queue, Full_Queue: EXCEPTION;
PRIVATE
  Size: CONSTANT := 100;
  TYPE Lists IS ARRAY(Integer RANGE <>)
    OF Elements;
  TYPE Queues IS
    RECORD
      Highest: Integer RANGE 0..Size := 0;
      List: Lists(1..Size);
    END RECORD;
END Priority_Queue_Package;

WITH Heap_Package, Exchange;
PACKAGE BODY Priority_Queue_Package IS
  PACKAGE Reheap_Package IS NEW Heap_Package
      (Elements,Priorities,Lists,Get_Rank);
  USE Reheap_Package;
  PROCEDURE Swap IS NEW Exchange(Elements);

  FUNCTION Empty(Queue: Queues) RETURN Boolean IS
  BEGIN
    RETURN Queue.Highest = 0;
  END Empty;

  PROCEDURE Enqueue(Queue: IN OUT Queues;
    Item: IN Elements) IS
    Index: Positive;
  BEGIN
```

```
    IF Queue.Highest = Size THEN
      RAISE Full_Queue;
    END IF;
    Queue.Highest := Queue.Highest + 1;
    Queue.List(Queue.Highest) := Item;
    Index := Queue.Highest;
    Sift_Up(Queue.List(1..Queue.Highest));
  END Enqueue;

  PROCEDURE Dequeue(Queue: IN OUT Queues;
    Item: OUT Elements) IS
  BEGIN
    IF Queue.Highest = 0 THEN
      RAISE Empty_Queue;
    END IF;
    Item := Queue.List(1);
    Queue.List(1) :=
      Queue.List(Queue.Highest);
    Queue.Highest := Queue.Highest - 1;
    IF Queue.Highest > 1 THEN
      Sift_Down(Queue.List(1..Queue.Highest));
    END IF;
  END Dequeue;
END Priority_Queue_Package;

-- Heap sort lesson code

GENERIC
  TYPE Elements IS PRIVATE;
  TYPE Keys IS PRIVATE;
  TYPE Tables IS ARRAY(Integer RANGE <>) OF
    Elements;
  WITH FUNCTION Get_Key(Element: Elements)
    RETURN Keys;
  WITH FUNCTION "<"(Left,Right: Keys)
    RETURN Boolean IS <>;
PACKAGE Heap_Sort_Package IS
  PROCEDURE Heap_Sort(Table: IN OUT Tables);
END Heap_Sort_Package;

WITH Heap_Package, Exchange;
PACKAGE BODY Heap_Sort_Package IS
  PACKAGE Reheap_Package IS NEW Heap_Package
    (Elements,Keys,Tables,Get_Key);
  USE Reheap_Package;
  PROCEDURE Swap IS NEW Exchange(Elements);

  PROCEDURE Heap_Sort(Table: IN OUT Tables) IS
  BEGIN
    FOR I IN Table'Range LOOP
      Sift_Up(Table(Table'First..I));
    END LOOP;
    FOR I IN REVERSE Table'First..Table'Last-1 LOOP
      Swap(Table(I + 1),Table(Table'First));
      Sift_Down(Table(Table'First..I));
    END LOOP;
  END Heap_Sort;
END Heap_Sort_Package;

-- Heap code used by priority queue and heap sort

GENERIC
  TYPE Elements IS PRIVATE;
  TYPE Keys IS PRIVATE;
  TYPE Heaps IS ARRAY(Integer RANGE <>) OF Elements;
  WITH FUNCTION Get_Key(Element: Elements) RETURN Keys;
```

```
  WITH FUNCTION "<"(Left,Right: Keys) RETURN
    Boolean IS <>;
PACKAGE Heap_Package IS
  PROCEDURE Sift_Up(Heap: IN OUT Heaps);
  PROCEDURE Sift_Down(Heap: IN OUT Heaps);
END Heap_Package;

PACKAGE BODY Heap_Package IS
  PROCEDURE Sift_Up(Heap: IN OUT Heaps) IS
    Leaf: Elements := Heap(Heap'Last);
    Node: Integer := Heap'Last;
  BEGIN
    WHILE Node > Heap'First LOOP
      EXIT WHEN Get_Key(Leaf) < Get_Key(Heap(Node / 2));
      Heap(Node) := Heap(Node / 2);
      Node := Node / 2;
    END LOOP;
    Heap(Node) := Leaf;
  END Sift_Up;

  PROCEDURE Sift_Down(Heap: IN OUT Heaps) IS
    Root: Elements := Heap(Heap'First);
    Node: Integer := Heap'First * 2;
  BEGIN
    WHILE Node <= Heap'Last LOOP
      IF Node < Heap'Last AND THEN
        Get_Key(Heap(Node)) <
        Get_Key(Heap(Node + 1)) THEN
          Node := Node + 1;
      END IF;
      EXIT WHEN Get_Key(Heap(Node)) < Get_Key(Root);
      Heap(Node / 2) := Heap(Node);
      Node := Node * 2;
    END LOOP;
    Heap(Node / 2) := Root;
  END Sift_Down;
END Heap_Package;

-- Binary search tree lesson code

GENERIC
  TYPE Keys IS PRIVATE;
  WITH FUNCTION Get_Key(Element: Elements)
    RETURN Keys;
  WITH FUNCTION ">"(Left,Right: Keys) RETURN
    Boolean IS <>;
PACKAGE Binary_Tree_Package.Search_Tree IS
  PROCEDURE Insert(Tree: IN OUT Trees;
    Element: IN Elements);
  PROCEDURE Delete(Tree: IN OUT Trees;
    Key: IN Keys);
  PROCEDURE Find(Tree: IN OUT Trees;
    Key: IN Keys; Element: OUT Elements;
    Found: OUT Boolean);
  Full_Tree,Duplicate_Key: EXCEPTION;
END Binary_Tree_Package.Search_Tree;

WITH Unchecked_Deallocation;
PACKAGE BODY Binary_Tree_Package.Search_Tree IS
  PROCEDURE Deallocate_Node IS NEW
  Unchecked_Deallocation(Nodes, Trees);

  PROCEDURE Search(Tree: IN Trees;
    Key: IN Keys; Current,Parent: OUT Trees;
    Found: OUT Boolean) IS
  BEGIN
```

```
  Parent := NULL;
  Current:= Tree;
  WHILE Current /= NULL LOOP
    EXIT WHEN Key = Get_Key(Current.Data);
    Parent := Current;
    IF Key > Get_Key(Current.Data) THEN
      Current := Trees(Current.Right);
    ELSE
      Current := Trees(Current.Left);
    END IF;
  END LOOP;
  Found := Current /= NULL AND THEN
    Key = Get_Key(Current.Data);
END Search;

PROCEDURE Insert(Tree: IN OUT Trees;
  Element: IN Elements) IS
  Key: Keys := Get_Key(Element);
  Leaf,Current,Parent: Trees;
  Found: Boolean;
BEGIN
  Search(Tree,Key,Current,Parent,Found);
  IF Found THEN
    RAISE Duplicate_Key;
  END IF;
  Leaf := NEW Nodes'(NULL, Element, NULL);
  IF Parent = NULL THEN
    Tree := Leaf;
  ELSIF Key > Get_Key(Parent.Data) THEN
    Parent.Right := Leaf;
  ELSE
    Parent.Left := Leaf;
  END IF;
EXCEPTION
  WHEN Storage_Error =>
    RAISE Full_Tree;
END Insert;

PROCEDURE Delete(Tree: IN OUT Trees; Key: IN Keys) IS
  Current,Parent: Trees;
  Found: Boolean;
  PROCEDURE Delete_Node(Node: IN OUT Trees) IS
    Max,Par,Remove: Trees;
  BEGIN
    Remove := Node;
    IF Node.Right = NULL THEN
      Node := Node.Left; -- 0-1 Child
    ELSIF Node.Left = NULL THEN
      Node := Node.Right; -- 1 Child
    ELSE -- 2 Children
      Par := Node.Left;
      Max := Par.Right;
      IF Max = NULL THEN
        Node.Left := Par.Left;
        Node.Data := Par.Data;
        Remove := Par;
      ELSE
        WHILE Max.Right /= NULL LOOP
          Par := Max;
          Max := Max.Right;
        END LOOP;
        Par.Right := Max.Left;
        Node.Data := Max.Data;
        Remove := Max;
      END IF;
    END IF;
```

```
      Deallocate_Node(Remove);
    END Delete_Node;
  BEGIN
    Search(Tree,Key,Current,Parent,Found);
    IF Found THEN
      IF Current = Tree THEN
        Delete_Node(Tree);
      ELSIF Parent.Left = Current THEN
        Delete_Node(Parent.Left);
      ELSE
        Delete_Node(Parent.Right);
      END IF;
    END IF;
  END Delete;

  PROCEDURE Find(Tree: IN OUT Trees;
    Key: IN Keys; Element: OUT Elements;
    Found: OUT Boolean) IS
    Parent,Current: Trees;
  BEGIN
    Search(Tree,Key,Current,Parent,Found);
    IF Found THEN
      Element := Current.Data;
    END IF;
  END Find;
END Binary_Tree_Package.Search_Tree;

-- Height balanced tree lesson code

GENERIC
PACKAGE Binary_Tree_Package.Height_Balancing IS
  FUNCTION Height(Tree: Trees) RETURN Natural;
  FUNCTION Balance_Factor(Tree: Trees) RETURN Integer;
  FUNCTION Is_AVL_Tree(Tree: Trees) RETURN Boolean;
  FUNCTION Is_Height_Balanced_n(Tree: Trees; n: Integer)
    RETURN Boolean;
END Binary_Tree_Package.Height_Balancing;

PACKAGE BODY Binary_Tree_Package.Height_Balancing IS
  FUNCTION Height(Tree: Trees) RETURN Natural IS
  BEGIN
    IF Tree = NULL THEN
      RETURN 0;
    ELSIF Height(Tree.Left) > Height(Tree.Right) THEN
      RETURN Height(Tree.Left) + 1;
    ELSE
      RETURN Height(Tree.Right) + 1;
    END IF;
  END Height;

  FUNCTION Balance_Factor(Tree: Trees) RETURN Integer IS
  BEGIN
    IF Tree = NULL THEN
      RETURN 0;
    ELSE
      RETURN Height(Tree.Left) - Height(Tree.Right);
    END IF;
  END Balance_Factor;

  FUNCTION Is_AVL_Tree(Tree: Trees) RETURN Boolean IS
  BEGIN
    RETURN Is_Height_Balanced_n(Tree,1);
  END Is_AVL_Tree;

  FUNCTION Is_Height_Balanced_n(Tree: Trees; n: Integer)
    RETURN Boolean IS
```

```
  BEGIN
    IF Tree = NULL THEN
      RETURN True;
    ELSE
      RETURN Balance_Factor(Tree) >= -n AND
        Balance_Factor(Tree) <= n AND THEN
        Is_Height_Balanced_n(Tree.Left,n) AND THEN
        Is_Height_Balanced_n(Tree.Right,n);
    END IF;
  END Is_Height_Balanced_n;
END Binary_Tree_Package.Height_Balancing;

-- Tree categorization code

GENERIC
  TYPE Keys IS PRIVATE;
  WITH FUNCTION Get_Key(Element: Elements)
    RETURN Keys;
  WITH FUNCTION ">"(Left,Right: Keys) RETURN
    Boolean IS <>;
PACKAGE Binary_Tree_Package.Height_Balancing.Categories IS
  FUNCTION Is_Almost_Complete(Tree: Trees)
    RETURN Boolean;
  FUNCTION Is_Complete(Tree: Trees) RETURN Boolean;
  FUNCTION Is_Binary_Search_Tree(Tree: Trees)
    RETURN Boolean;
  FUNCTION Is_Heap(Tree: Trees) RETURN Boolean;
END Binary_Tree_Package.Height_Balancing.Categories;

PACKAGE BODY Binary_Tree_Package.Height_Balancing.Categories IS
  FUNCTION Count_Nodes(Tree: Trees) RETURN Natural IS
  BEGIN
    IF Tree = NULL THEN
      RETURN 0;
    ELSE
      RETURN 1 + Count_Nodes(Tree.Left) +
        Count_Nodes(Tree.Right);
    END IF;
  END Count_Nodes;

  FUNCTION Is_Almost(Tree: Trees; Total_Nodes,
    Node : Natural) RETURN Boolean IS
  BEGIN
    RETURN Node > Total_Nodes OR ELSE
      (Tree /= NULL AND THEN Is_Almost(Tree.Left,
      Total_Nodes,Node * 2) AND THEN
      Is_Almost(Tree.Right,Total_Nodes,Node * 2 + 1));
  END Is_Almost;

  FUNCTION Is_Almost_Complete(Tree: Trees)
    RETURN Boolean IS
  BEGIN
    RETURN Is_Almost(Tree,Count_Nodes(Tree),1);
  END Is_Almost_Complete;

  FUNCTION Is_Complete(Tree: Trees) RETURN Boolean IS
  BEGIN
    RETURN Count_Nodes(Tree) = 2 ** Height(Tree) - 1
      AND THEN Is_Almost(Tree,Count_Nodes(Tree),1);
  END Is_Complete;

  FUNCTION Minimum(Tree: Trees) RETURN Keys IS
    Smallest: Keys := Get_Key(Tree.Data);
  BEGIN
    IF Tree.Left /= NULL AND THEN
      Smallest > Minimum(Tree.Left) THEN
```

```
      Smallest := Minimum(Tree.Left);
    END IF;
    IF Tree.Right /= NULL AND THEN
      Smallest > Minimum(Tree.Right) THEN
      Smallest := Minimum(Tree.Right);
    END IF;
    RETURN Smallest;
  END Minimum;

  FUNCTION Maximum(Tree: Trees) RETURN Keys IS
    Largest: Keys := Get_Key(Tree.Data);
  BEGIN
    IF Tree.Left /= NULL AND THEN
      Maximum(Tree.Left) > Largest THEN
      Largest := Maximum(Tree.Left);
    END IF;
    IF Tree.Right /= NULL AND THEN
      Maximum(Tree.Right) > Largest THEN
      Largest := Maximum(Tree.Right);
    END IF;
    RETURN Largest;
  END Maximum;

  FUNCTION Is_Binary_Search_Tree(Tree: Trees)
      RETURN Boolean IS
  BEGIN
    IF Tree = NULL THEN
      RETURN True;
    ELSE
      RETURN ((Tree.Left = NULL OR ELSE
      Get_Key(Tree.Data) > Maximum(Tree.Left)) AND
      (Tree.Right = NULL OR ELSE
      Minimum(Tree.Right) > Get_Key(Tree.Data))) AND THEN
      Is_Binary_Search_Tree(Tree.Left) AND THEN
      Is_Binary_Search_Tree(Tree.Right);
    END IF;
  END Is_Binary_Search_Tree;

  FUNCTION Is_Heap(Tree: Trees) RETURN Boolean IS
  BEGIN
    IF Tree = NULL THEN
      RETURN True;
    ELSE
      RETURN ((Tree.Left = NULL OR ELSE
      (Get_Key(Tree.Data) > Get_Key(Tree.Left.Data)
      OR Get_Key(Tree.Data) = Get_Key(Tree.Left.Data))) AND
      (Tree.Right = NULL OR ELSE
      (Get_Key(Tree.Data) > Get_Key(Tree.Right.Data)
      OR Get_Key(Tree.Data) = Get_Key(Tree.Right.Data)))) AND THEN
      Is_Heap(Tree.Left) AND THEN Is_Heap(Tree.Right);
    END IF;
  END Is_Heap;
END Binary_Tree_Package.Height_Balancing.Categories;

-- Code common to graph lessons

GENERIC
  TYPE Vertices IS (<>);
PACKAGE Graph_Package IS
  TYPE Graphs IS PRIVATE;
  PROCEDURE Add_Edge(Graph: IN OUT Graphs;
    Left,Right: IN Vertices);
  Empty_Graph: CONSTANT Graphs;
PRIVATE
  TYPE Graphs IS ARRAY(Vertices,Vertices) OF Boolean;
  Empty_Graph: CONSTANT Graphs := (OTHERS =>
```

```
    (OTHERS => False));
END Graph_Package;

PACKAGE BODY Graph_Package IS
  PROCEDURE Add_Edge(Graph: IN OUT Graphs;
    Left,Right: IN Vertices) IS
  BEGIN
    Graph(Left,Right) := True;
    Graph(Right,Left) := True;
  END Add_Edge;
END Graph_Package;

-- Graph search code

GENERIC
  WITH PROCEDURE Put(Item: IN Vertices);
PACKAGE Graph_Package.Search IS
  PROCEDURE Depth_First_Search(Graph: IN Graphs;
    Start: IN Vertices);
  PROCEDURE Breadth_First_Search(Graph: IN Graphs;
    Start: IN Vertices);
PRIVATE
  TYPE Mark_List IS ARRAY(Vertices) OF Natural;
  TYPE Search_Graphs IS RECORD
    Graph: Graphs;
    Marked: Mark_List;
    Cycles: Boolean;
  END RECORD;
  PROCEDURE DF_Search(Search_Graph: IN OUT Search_Graphs;
    Start: IN Vertices; Depth: IN Positive;
    Output: IN Boolean);
  PROCEDURE BF_Search(Search_Graph: IN OUT Search_Graphs;
    Start: IN Vertices);
END Graph_Package.Search;

WITH FIFO_Queue_Package;
PACKAGE BODY Graph_Package.Search IS
  PROCEDURE DF_Search(Search_Graph: IN OUT Search_Graphs;
    Start: IN Vertices; Depth: IN Positive;
    Output: IN Boolean) IS
  BEGIN
    IF Search_Graph.Marked(Start) = 0 THEN
      Search_Graph.Marked(Start) := Depth;
      IF Output THEN
        Put(Start);
      END IF;
      FOR Next IN Vertices LOOP
        IF Search_Graph.Graph(Start, Next) THEN
          DF_Search(Search_Graph,Next,Depth+1,Output);
        END IF;
      END LOOP;
    ELSIF Search_Graph.Marked(Start) - Depth > 1 THEN
      Search_Graph.Cycles := True;
    END IF;
  END DF_Search;

  PROCEDURE Depth_First_Search(Graph: IN Graphs;
    Start: IN Vertices) IS
    Search_Graph: Search_Graphs;
  BEGIN
    Search_Graph.Graph := Graph;
    Search_Graph.Marked := (OTHERS => 0);
    DF_Search(Search_Graph,Start,1,True);
  END Depth_First_Search;

  PROCEDURE BF_Search(Search_Graph: IN OUT Search_Graphs;
```

```
    Start: IN Vertices) IS
    PACKAGE Search_Queue IS NEW
      FIFO_Queue_Package(Vertices);
    USE Search_Queue;
    Queue: Queues;
    Current: Vertices := Start;
  BEGIN
    Enqueue(Queue,Current);
    Search_Graph.Marked(Current) := 1;
    Put(Current);
    WHILE NOT Empty(Queue) LOOP
      Dequeue(Queue, Current);
      FOR Next IN Vertices LOOP
          IF Search_Graph.Graph(Current,Next) AND
            THEN Search_Graph.Marked(Next) = 0 THEN
            Enqueue(Queue,Next);
            Search_Graph.Marked(Next) := 1;
            Put(Next);
          END IF;
      END LOOP;
    END LOOP;
  END BF_Search;

  PROCEDURE Breadth_First_Search(Graph: IN Graphs;
    Start: IN Vertices) IS
    Search_Graph: Search_Graphs;
  BEGIN
    Search_Graph.Graph := Graph;
    Search_Graph.Marked := (OTHERS => 0);
    BF_Search(Search_Graph,Start);
  END Breadth_First_Search;
END Graph_Package.Search;

-- Graph categories lesson code

PACKAGE BODY Graph_Package.Categories IS
  FUNCTION Is_Connected(Graph: Graphs) RETURN Boolean;
  FUNCTION Has_Cycles(Graph: Graphs) RETURN Boolean;
  FUNCTION Is_Tree(Graph: Graphs) RETURN Boolean;
END Graph_Package.Categories;

PACKAGE BODY Graph_Package.Search.Categories IS
  FUNCTION Is_Connected(Graph: Graphs) RETURN Boolean IS
    Search_Graph: Search_Graphs;
  BEGIN
    Search_Graph.Graph := Graph;
    Search_Graph.Marked := (OTHERS => 0);
    DF_Search(Search_Graph,Vertices'First,1,False);
    FOR Vertex IN Vertices LOOP
      IF Search_Graph.Marked(Vertex) = 0 THEN
        RETURN FALSE;
      END IF;
    END LOOP;
    RETURN True;
  END Is_Connected;

  FUNCTION Has_Cycles(Graph: Graphs) RETURN Boolean IS
    Search_Graph: Search_Graphs;
  BEGIN
    Search_Graph.Graph := Graph;
    Search_Graph.Marked := (OTHERS => 0);
    Search_Graph.Cycles := False;
    FOR Start IN Vertices LOOP
      IF Search_Graph.Marked(Start) = 0 THEN
        DF_Search(Search_Graph,Start,1,False);
      END IF;
```

```
      IF Search_Graph.Cycles THEN
        RETURN True;
      END IF;
    END LOOP;
    RETURN False;
  END Has_Cycles;

  FUNCTION Is_Tree(Graph: Graphs) RETURN Boolean IS
  BEGIN
    RETURN Is_Connected(Graph) AND THEN NOT
      Has_Cycles(Graph);
  END Is_Tree;
END Graph_Package.Search.Categories;

-- Code common to sort lessons

GENERIC
  TYPE Elements IS PRIVATE;
PROCEDURE Exchange(Left,Right: IN OUT Elements);

PROCEDURE Exchange(Left,Right: IN OUT Elements) IS
  Temp: Elements := Left;
BEGIN
  Left := Right;
  Right := Temp;
END Exchange;

-- Quadratic sort lesson code

GENERIC
  TYPE Elements IS PRIVATE;
  TYPE Keys IS PRIVATE;
  TYPE Tables IS ARRAY(Integer RANGE <>) OF
    Elements;
  WITH FUNCTION Get_Key(Element: Elements)
    RETURN Keys;
  WITH FUNCTION "<"(Left,Right: Keys)
    RETURN Boolean IS <>;
PACKAGE Quadratic_Sort_Package IS
  PROCEDURE Bubble_Sort(Table: IN OUT
    Tables);
  PROCEDURE Insertion_Sort(Table: IN OUT
    Tables);
  PROCEDURE Selection_Sort(Table: IN OUT
    Tables);
END Quadratic_Sort_Package;

WITH Exchange;
PACKAGE BODY Quadratic_Sort_Package IS
  PROCEDURE Swap IS NEW Exchange(Elements);
  PROCEDURE Rotate(Table: IN OUT Tables) IS
    Temp: Elements := Table(Table'Last);
  BEGIN
    Table(Table'First+1..Table'Last) :=
      Table(Table'First..Table'Last-1);
    Table(Table'First) := Temp;
  END Rotate;

  PROCEDURE Bubble_Sort(Table: IN OUT Tables) IS
  BEGIN
    FOR I IN Table'First..Table'Last-1 LOOP
      FOR J IN REVERSE I+1..Table'Last LOOP
        IF Get_Key(Table(J)) <
          Get_Key(Table(J-1)) THEN
          Swap(Table(J),Table(J-1));
        END IF;
```

```
      END LOOP;
    END LOOP;
  END Bubble_Sort;

  PROCEDURE Insertion_Sort(Table: IN OUT
    Tables) IS
    J: Integer;
  BEGIN
    FOR I IN Table'First+1..Table'Last LOOP
      J := I - 1;
      WHILE Get_Key(Table(I)) <
        Get_Key(Table(J)) LOOP
        J := J -1;
        EXIT WHEN J < Table'First;
      END LOOP;
      IF I > J + 1 THEN
        Rotate(Table(J+1..I));
      END IF;
    END LOOP;
  END Insertion_Sort;

  PROCEDURE Selection_Sort(Table: IN OUT
      Tables) IS
    Minimum : Integer;
  BEGIN
    FOR I IN Table'First..Table'Last-1 LOOP
      Minimum := I;
      FOR J IN I+1..Table'Last LOOP
        IF Get_Key(Table(J)) <
          Get_Key(Table(Minimum)) THEN
          Minimum := J;
        END IF;
      END LOOP;
      Swap(Table(I),Table(Minimum));
    END LOOP;
  END Selection_Sort;
END Quadratic_Sort_Package;

-- Efficient sort lesson code

GENERIC
  TYPE Elements IS PRIVATE;
  TYPE Keys IS PRIVATE;
  TYPE Tables IS ARRAY(Integer RANGE <>) OF
    Elements;
  WITH FUNCTION Get_Key(Element: Elements)
    RETURN Keys;
  WITH FUNCTION "<"(Left,Right: Keys)
    RETURN Boolean IS <>;
PACKAGE Efficient_Sort_Package IS
  PROCEDURE Merge_Sort(Table: IN OUT
    Tables);
  PROCEDURE Quick_Sort(Table: IN OUT
    Tables);
END Efficient_Sort_Package;

WITH Exchange;
PACKAGE BODY Quadratic_Sort_Package IS
  PROCEDURE Swap IS NEW Exchange(Elements);
  PROCEDURE Rotate(Table: IN OUT Tables) IS
    Temp: Elements := Table(Table'Last);
  BEGIN
    Table(Table'First+1..Table'Last) :=
      Table(Table'First..Table'Last-1);
    Table(Table'First) := Temp;
  END Rotate;
```

```
  PROCEDURE Bubble_Sort(Table: IN OUT Tables) IS
  BEGIN
    FOR I IN Table'First..Table'Last-1 LOOP
      FOR J IN REVERSE I+1..Table'Last LOOP
        IF Get_Key(Table(J)) <
          Get_Key(Table(J-1)) THEN
          Swap(Table(J),Table(J-1));
        END IF;
      END LOOP;
    END LOOP;
  END Bubble_Sort;

  PROCEDURE Insertion_Sort(Table: IN OUT
    Tables) IS
    J: Integer;
  BEGIN
    FOR I IN Table'First+1..Table'Last LOOP
      J := I - 1;
      WHILE Get_Key(Table(I)) <
        Get_Key(Table(J)) LOOP
        J := J -1;
        EXIT WHEN J < Table'First;
      END LOOP;
      IF I > J + 1 THEN
        Rotate(Table(J+1..I));
      END IF;
    END LOOP;
  END Insertion_Sort;

  PROCEDURE Selection_Sort(Table: IN OUT
      Tables) IS
    Minimum : Integer;
  BEGIN
    FOR I IN Table'First..Table'Last-1 LOOP
      Minimum := I;
      FOR J IN I+1..Table'Last LOOP
        IF Get_Key(Table(J)) <
          Get_Key(Table(Minimum)) THEN
          Minimum := J;
        END IF;
      END LOOP;
      Swap(Table(I),Table(Minimum));
    END LOOP;
  END Selection_Sort;
END Quadratic_Sort_Package;
```

# APPENDIX B EVALUATIONS

## B.1 Spring 1997 evaluations

### Evaluator Information

Web Use:

- Daily Users 8
- Weekly Users 6
- Infrequent User 1

Visualization Use:

- More than 5 hours 4
- More than 1 hour 10
- Less than 1 hour 1

Browser:

- Netscape 15

Operating System:

- Windows 95 5
- Windows NT 3
- Macintosh 6
- X Windows 1

### Evaluation Question Averages

Effectiveness As Learning Tool

| | |
|---|---|
| 4.6 | Visualizations and animations are very important to my ability to understand subjects like data structures. |
| 4.4 | I would use learning tools like this one frequently if they were available. |
| 3.3 | I would prefer to spend more time learning with such tools and less time in classroom lectures. |
| 4.7 | I would like to see more such learning tools in the future. |

Depth of Use

| | |
|---|---|
| 3.9 | I used the "I'll Try" feature of the visualizer frequently. |
| 3.9 | I tried most of the eleven different visualizations. |

Ease of Use

| | |
|---|---|
| 3.9 | Instructions for use are clear, complete and concise. |
| 3.9 | This visualizer requires no additional assistance to use. |
| 4.1 | The method for navigating through the visualizer is clear and simple. |
| 3.8 | The purpose of all buttons and other controls was clear. |

Platform Effectiveness

| | |
|---|---|
| 4.1 | The World Wide Web is an effective platform for this visualization tool. |
| 3.4 | The visualizer worked properly, there were no apparent errors and it did not malfunction during use. |
| 3.5 | The response time of the visualizer is fast enough. |
| 3.0 | It was easy to find a machine on which the visualizer would run. |

**Frequency of Textual Responses**

Things They Liked

| | |
|---|---|
| 7 | ease of use |
| 6 | visualization, visual learner |
| 5 | helpful for understanding data structures |
| 4 | easy access on WWW |
| 2 | learn faster |
| 1 | use of color |
| 1 | good use of technology |
| 1 | interactivity |
| 1 | self-directed learning |

Things They Disliked

| | |
|---|---|
| 10 | too slow |
| 6 | requires powerful machine, hard to find one |
| 3 | too small, not centered, not expandable |
| 2 | graphs look weird |
| 1 | needs more work on interface |
| 1 | bug in insertion sort I'll Try |
| 1 | using I'll Try is confusing |

Things They Would Change

| | |
|---|---|
| 4 | add music, sounds or narration |
| 2 | require assignments to be done on the system |
| 2 | add test or quiz |
| 2 | add textual explanations |
| 2 | add pause button |
| 2 | add stop button |
| 2 | add more material for other parts of course |
| 1 | include a progress indicator for slow PCs |
| 1 | use numbers instead of colors |

## B.2 Fall 1997 evaluations

### Evaluator Information

Browser:

- Netscape 18
- Internet Explorer 4

Operating System:

- Windows 95 17
- Windows NT 4
- Macintosh 1

### Evaluation Question Averages

Effectiveness As Learning Tool (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 4.6 | Completing the on-line homeworks helped me understand the material on graphs, trees and sorting. |
| 4.4 | The animations were one of the most important aspects of the on-line homework assignments. |
| 4.5 | The I'll Try feature was an essential feature that helped me understand the material. |
| 4.1 | The randomly generated data was a helpful aspect of the on-line homeworks. |

Confidence (1=Not Confident, 2=Somewhat Confident, 3=Very Confident)

| | |
|---|---|
| 2.7 | Rate your confidence in the above responses. (3.0 is very confident) |

Ease of Use (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 3.1 | Instructions for use are clear, complete and concise. |
| 3.7 | The method for navigating through the on-line homeworks is clear and simple. |
| 3.4 | The purpose of all buttons and other controls was clear. |

Platform Effectiveness (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 4.0 | The World Wide Web is an effective platform for the on-line homeworks. |
| 3.6 | The software worked properly, there were no apparent errors and it did not malfunction during use. |
| 3.3 | It was easy to find a machine on which the software would run. |

Use of Controls (1=Never, 2=Occassionally, 3 = Frequently)

| | |
|---|---|
| 2.2 | How often did you use the I'll Pick feature? |
| 1.9 | How often did you use the single-step feature or the pause and resume buttons? |
| 2.3 | How often did you use the abort button? |
| 1.5 | How often did you use the speed control? |

Other Issues (1=Too Few, 2=About Right, 3=Too Many)

| | |
|---|---|
| 2.2 | The number of nodes in the graphs, the binary trees are the sorting arrays was |
| 2.0 | The number of correct responses required for each homework assignment was |

The Animations (1=Disagree, 2=Don't Recall, 3=Agree)

| | |
|---|---|
| 2.2 | The meaning of the checkmarks is the binary tree traversal was clear |
| 2.5 | The meaning of the arrow in the graph search was clear |
| 2.4 | The meaning of the node colors in the heap sort was clear |

**Frequency of Textual Responses**

Benefits of On-line Homework

| | |
|---|---|
| 5 | Interactivity was beneficial |
| 8 | Visual aspect was beneficial |

Browser / Platform Issues

| | |
|---|---|
| 5 | Should work on all browsers and platforms or those required should be indicated |
| 3 | It was difficult to find a machine that had the necessary browser |

User Interface Issues

| | |
|---|---|
| 5 | Instructions need to be made clearer |
| 2 | Main page should be organized by homework assignments |
| 4 | Clicking on objects must be too exact |
| 1 | Clarify format, make more user-friendly |

Controls

| | |
|---|---|
| 2 | Didn't recognize speed control |
| 1 | Single step did not work |
| 1 | Buttons need clearer names |

Understandability of Animations

| | |
|---|---|
| 3 | Explain sorts better |

Suggested Additions

| | |
|---|---|
| 4 | Make all, or at-least more, homeworks on-line |

## B.3 Spring 1998 evaluations

### Evaluation Question Averages

Effectiveness As Learning Tool (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 4.3 | Completing the on-line assignments helped me understand the material on graphs, trees and sorting. |
| 4.5 | The animations were one of the most important aspects of the on-line assignment assignments. |
| 4.6 | The I'll Try feature was an essential feature that helped me understand the material. |
| 4.3 | The randomly generated data was a helpful aspect of the on-line assignments. |

Confidence (1=Not Confident, 2=Somewhat Confident, 3=Very Confident)

| | |
|---|---|
| 2.7 | Rate your confidence in the above responses. (3.0 is very confident) |

Ease of Use (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 3.5 | Instructions for use are clear, complete and concise. |
| 4.0 | The method for navigating through the on-line homeworks is clear and simple. |
| 3.5 | The purpose of all buttons and other controls was clear. |

Platform Effectiveness (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 4.4 | The World Wide Web is an effective platform for the on-line homeworks. |
| 4.1 | The software worked properly, there were no apparent errors and it did not malfunction during use. |

Use of Controls (1=Never, 2=Occassionally, 3 = Frequently)

| | |
|---|---|
| 2.3 | How often did you use the I'll Pick feature? |
| 1.8 | How often did you use the single-step feature or the pause and resume buttons? |
| 1.9 | How often did you use the abort button? |
| 1.7 | How often did you use the speed control? |

**Frequency of Textual Responses**

Benefits of On-line Homework

| | |
|---|---|
| 2 | Interactivity was beneficial |
| 6 | Visual aspect was beneficial |
| 1 | Learn more quickly |

Logistic Issues

| | |
|---|---|
| 2 | Lectures must preceed using courseware |
| 1 | Should be available outside the lab |
| 1 | Too many repetitions required |

User Interface Issues

| | |
|---|---|
| 2 | Instructions need to be made clearer |
| 3 | Clicking on objects must be too exact |
| 1 | Clarify format, make more user-friendly |

Controls

| | |
|---|---|
| 1 | Too many controls |

Understandability of Animations

| | |
|---|---|
| 1 | Explain sorts better |

Suggested Additions

| | |
|---|---|
| 2 | Add sound or music |

## B.4 Fall 1998 evaluations

### Evaluation Question Averages

Effectiveness As Learning Tool (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 4.0 | Completing the on-line assignments helped me understand the material on graphs, trees and sorting. |
| 4.3 | The animations were one of the most important aspects of the on-line assignment assignments. |
| 4.0 | The I'll Try feature was an essential feature that helped me understand the material. |
| 4.0 | The randomly generated data was a helpful aspect of the on-line assignments. |

Confidence (1=Not Confident, 2=Somewhat Confident, 3=Very Confident)

| | |
|---|---|
| 2.7 | Rate your confidence in the above responses. (3.0 is very confident) |

Ease of Use (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 3.9 | Instructions for use are clear, complete and concise. |
| 4.1 | The method for navigating through the on-line homeworks is clear and simple. |
| 3.9 | The purpose of all buttons and other controls was clear. |

Platform Effectiveness (1=Strongly Disagree, 5 = Strongly Agree)

| | |
|---|---|
| 3.8 | The World Wide Web is an effective platform for the on-line homeworks. |
| 2.6 | The software worked properly, there were no apparent errors and it did not malfunction during use. |

Use of Controls (1=Never, 2=Occassionally, 3 = Frequently)

| | |
|---|---|
| 2.2 | How often did you use the I'll Pick feature? |
| 2.2 | How often did you use the single-step feature or the pause and resume buttons? |
| 2.0 | How often did you use the abort button? |
| 1.7 | How often did you use the speed control? |

**Frequency of Textual Responses**

Benefits of On-line Homework

| | |
|---|---|
| 9 | Interactivity was beneficial |
| 10 | Visual aspect was beneficial |
| 1 | Learn more quickly |

Browser, Platform, Performance Issues

| | |
|---|---|
| 8 | Should work on all browsers and platforms |
| 11 | Java was a problem |
| 17 | Download speed was a problem |
| 7 | Unreliable |

Logistic Issues

| | |
|---|---|
| 5 | Should be available outside the lab |
| 2 | Took away from time on projects |

User Interface Issues

| | |
|---|---|
| 3 | Instructions need to be made clearer or more needed |
| 12 | Add concurrent explanations |
| 1 | Needs better user interface |

Controls

| | |
|---|---|
| 7 | Better speed control |
| 1 | Easier transition between single step and continuous |

Understandability of Animations

| | |
|---|---|
| 2 | Explain sorts better |

Suggested Additions

| | |
|---|---|
| 1 | Add audio |
| 2 | Add more topics |
| 6 | Add quiz or more interaction |
| 1 | Add difficulty levels |

# APPENDIX C THINK-ALOUD TRANSCRIPTIONS

Note: The italicized text is answers supplied by the researcher to the student's questions.

## C.1 Binary tree traversals

OK, Where do I start out? Traversals? *Traversals.* I think now while I'm reading that? *Well no, you can, you can think—.* I mean like I read this and I start saying what I'm thinking when I go to start visualizing? *Well whenever, yeah whenever it's appropriate. You can articulate as much as—.* Oh cool. I mean I just like go. *Well, um.* I select single step or continuous or whatever I want? *Right, you can do that.* Make tree, Preorder, inorder postorder. *Right.* What do I have to figure out. *Uh, huh. I'm going to try not tell you anymore about it.* All right. OK, I understand. *I think I was doing too much additional explanation last time*. OK, preorder. *See if you can figure out, other than questions on how to use it.* So now for 10. Let me try again. Preorder 43, 15. Exhausted all possibilities you get three check marks. It looks like every time you exhaust all possibilities you check three check marks. That's what it looks like. It goes from start and then it goes to one of the branches and from there each time it goes each time it advances it seems to add a check mark and once it's exhausted all possibilities it makes three check marks. Inorder. Same thing seems to be happening for inorder. Let me try making another tree. OK, this one is a bit more distributed. So we'll try 5. It's going along the branch and each jump it makes it seems to mark a check mark, and as soon as its finished all possibilities on the way back it will make the check marks. Or for each possibility, each branch that it makes. Maybe the check mark determines or represents which path it's taking, I mean the possibilities which is probably what it is doing. And its seems to be just going from the left-most. It starts at the top and then takes the left-most branch and then once it's exhausted, once it reaches the end it goes back up until it finds another branch and then from there it goes to the left-most and then it continues along that way until it's all done. Let's try. That would be preorder and let's try inorder with this one. Let me look at it. When it reaches the end, every time it reaches the end it copies the number and then when it goes up. I'll try to make another graph and check it out to see. With inorder it seems to just goes randomly. With every block it touches, it copies. OK, yeah. So with preorder it seems to always take the left branch whenever possible and wherever it goes, it copies the number to the list. And it stop this—abort. Now, if I try inorder we'll see the—. I think this seems to copy the number when it reaches an end and when there is no where else to go and copies the number. The only thing I can't figure out is why the branches it copies it over first. After exhausting the left-most branch when it goes back up, it seems to copy the value and then it goes to the other branches. That what it looks like. Every time it goes, it goes down to the left-most possible and then on the way up if there's a—, it copies it and then it goes to a right—or, a branch

to the right if possible. That's what it looks like. Let me stop this. This seems to be it. Ah, this seems to makes sense. This goes all the way down to the bottom and whatever is at the bottom-most level it writes first from the left to the right, or no, it branches off left at each junction and when it can't go any more it copies the number and then it goes back and then goes to the other branches and then completes the lowest level branches and then it copies all those numbers and then it copies them hierarchically as it came down. It seems to makes sense. Let me try level order now. Yeah. This seems to simply just go the distances from the starting point and then this also seems to go from the left and travel to the right it just goes from the distances, like the number of nodes between the starting point and the current node and then it does all the ones with one, and then two, and then three. I think it's easier to understand it when there's a large tree with a lot of branches. It makes it a lot easier to understand it, than if there's just two branches. OK, I think I'm done. *OK. All right.* Do you know what really if you have a graph like this it is much easier to see than if you have a graph with only two branches. Because this way you can really see all the patterns and how it develops. *You are really good at doing this. It is not an easy thing to do.* OK. *Maybe you can do them all. It is so important to be good at thinking out loud and it is a hard thing to do if you haven't done it. Now, let me show you one—progress window. What you need to do before you can submit these—you'll notice that on this one there is an I'll try mode.* OK. *So, what you have to do is you have to get 20 right before you can submit these.* Oh, OK, all right sure. Preorder. OK, let me try I'll try in preorder. Click the next node. Preorder was OK 53, 26. OK yeah, This just kind of goes, this just kind of goes along the left path and then it goes, and when it can't hits and when it can't go up, when it can't branch off anymore it goes to the previous node and then checks if that has any other branches and if that doesn't have any other branches it goes to the previous node and then. Try and then it tries this and it goes back to the previous node, nothing, previous node, nothing and then it goes over here and then goes back up and branches. OK wait. I'm going to try inorder now. What did inorder do again? It would go down and then on the way up it would, on the way up it would, oh right. It goes all the way down to the bottom oops, and then it goes up and then it everything as it goes up and then when it gets whoops, here. We'll go all the way down to here. What? OK, try one more time. OK, let's see. Every time I say something it records. Um, let's see 53 and after that I'm confused 24, why 24. I don't understand that at all. OK. I didn't get that at all. I didn't, I don't see why it did 24. Hmm. It went, it should have, I don't see why it didn't go to. I don't understand this at all. OK, let me ask it to show me how to do it. Can there be a flaw in this program, or is it not? *I don't think so.* OK. Because just watching, maybe I just did something really wrong. Because. I figured that. Because when it got here it copied 24 than 94 and then, maybe, Oh. Oh. I see. Yeah, I understand why, I understand why, because there's no, there's supposed to, if there's a node on the left then it will do it first but this is a right side node. OK, I understand, so even if there is no node on the left, it'll still, it'll treat it as, OK. Now I see. Now I

understand. OK, that's cool. *OK, so then, the idea is once you understand is just to do 20 of them.* Yeah, OK. Now, I get it. *All right.* Um, let me just do. Oh my God, this so complicated. It will trace all the way down to here, then, here. All the way down. OK. Good. Now let me try level order. That would to be easy. You just start off with—to the right. Let me just do. Let me do another preorder and inorder. Inorder means it goes to the bottom up and it punches the thing on the way up. OK. What! Oh no. Oh yeah, 48 then it goes 16, Preorder thing is so confusing—0. Preorder. I'm done. *Good, do the next one, there's one more for today.* See, if it's big, something like this, it's easier to understand. *OK.* Then just a few things, in some of these other examples it's just one, two branches. *Right.* It's hard to see how this works. *OK. Your are good at verbalizing. It's not easy to do.* All right, cool. Now I stop. *Right, and then do the search trees.*

## C.2 Binary search tree

Try binary search trees now? *Right.* OK. Are you going to check if this thing actually works afterwards. *What's that?* Are you going to check if this thing works. *Well, yes, I'll play it back. It's recording now, right?* Yeah, yeah. *Does it ever go off?* Uh, yeah, but I changed the mic sensitivity to high because I was—. *OK.* It turned off accidentally. OK. Start visualization. Now this is supposed to be doing something? Cool. I'd like to insert 42. OK. Insert. I'd like to insert 50. 89. Oh, I see. This is sort of sorted. It starts off with the first number and if it's greater, it goes to the right and if it's less it goes to the left. And then it just keeps traversing a lot like—if you want to add a higher number it'll go—interesting thing would be to see if it's between like 15 and 89 what would happen. I'll pick. Insert. Enter value to insert. 70, see what would happen. Oh, it inserts after 89, but to the left. This is really pretty weird. So after 50 if I try to insert between 42 and 50, like 45. Oh, OK. So what does is if it's greater than the number it'll go—insert to the right, and if it's less it'll go to the left. Try. If insert the same number, what happens? Value already in tree. OK, that's neat. Um. Let's try 5 means—it should probably put it—it should probably put it to the right, if I'm not mistaken. Yeah, OK, cool. I understand this stuff. Not bad at all. So if I put it like 4 then it'll go to the left and underneath 5. Yeah, OK, I understand. If I put like 1, it goes to the left of 3, and if I put 2, it'll go to the right of 1. OK, this is much easier than allowing it to do this—to enter numbers yourself—to see how it puts it together. OK, now I'm going to try a high number like 100 or 99. It should put it to the right of 89—just to make sure I know what I am doing. Good, OK, now fine. For delete, good, let me try delete. Whoa, whoa, it moved 70 into it's position. Now, how on earth did it do that? I have absolutely no idea how it did that. Let me try deleting something else. 5. What did it do? Moves 4 into—OK, it collapses the tree. What I don't see is, let's see. Um. Let me insert another number between 78 and 89. OK, now will it collapse 89 into that? 75?

Now if I delete 75. Delete deletes the next—moves the next greater number into its position. Let's try to delete 42. What happens? 45? 89. Why does it do that? I don't understand this at all. Let's try this—OK add a number, 43. So when you add a number, it explores to the edges of the tree and if it's less than the edge of the tree, then it'll add a number to the left and if it's greater it'll add the number to the right. So, let's see, now, why is it when I delete, when I delete 89, huh 15 moved to the right of 45. OK, now if I delete 89, what would I want to go there, 99? Oh, it has to preserve the integrity of the table, so everything to the right of it has to be, has to be in order, so, so basically—. OK, I see that. So, I think what happens is that it has to preserve the integrity of the table. So, if I delete 99. OK, that just goes away. Then I add one more and one to the left. OK, now it has to preserve the integrity of the table, so if I delete 89 it'll do something such that the one to the right of it has to be greater and the one of the left of it has to be less. So, it would probably move—all right. Yes, OK, I see, I understand this now. So—. Let's try adding 0. 0 will go there, ah. Um—already in the tree, OK. Delete. If I delete the 42. What would it move there? It looks like it would move 90 or 45, but probably not 45, 43? 43. Why did it move 43? I have no idea. This is neat stuff, but really weird. Why did it do that? If it preserves the integrity why didn't it do—what it probably does is follows the path and it finds the first one that follows the path and finds the first one that could go there and it follows the path to the left, then. I'll try adding 89 and then I'll delete 90 and see if it was 89 there. So if it's 89, no if it's 91. I don't understand how it works. I absolutely don't get it. How does it do it? How does it—. Options. Let me trying deleting another number now 91 and see what happens. Abort the whole thing. Is there a way I can like clear this thing and start all over? *Clear?* Clear everything off the screen and start all over, or do I have to—. *You have to delete them.* This is really confusing with delete algorithm. Very, like I have no idea. I can see it to the end, I see how it works, but if there is a branch, I just—. *There is code that goes with it.* Oh, I can look at the code? *Sure.* Oh, I can look at the code. OK. *Right down there.* The code I'll be able to deal with. Delete. OK. Remove node dot right equals null. Node dot right is null, then remove is a node. Node is the node of—. OK. If node dot right is null so that means if there's nothing to the right of it, then node is node dot left. OK, that makes sense. If node dot left is null, then it goes to the right. OK, that makes sense. Now if it has both children, par is node dot left—par gets node dot left and max gets par dot right. Max equals null. Max equals null. That means that it—node dot left. What the? Node dot left gets par dot left. Oh, oh. Am I allowed to draw something out? *Sure.* I don't have any paper. So, basically what happens is I have some node that I want to remove. OK, I got a node I got to remove and then node may have some branches—whatever. So, what I do is if the node dot right is null, so if the thing I try to remove doesn't have anything on the right of it then I move the node along to node dot left. OK, that makes sense. If there's nothing on the left, I move to the right. OK, but if I have two children, I make two, one is— I start from the node and then par called node dot left then and this is called par and max I call par dot right. Now if max is equal

to null so there is no max, then node dot left, node dot left gets par dot left, par dot left, which could be null. OK, this goes up here. And node dot data. That makes sense and you remove par. Remove gets par, and then, and then what? Oh, I see it's transporting the data. So, if it goes down there and it's null to the right. I start up here and I have a tree, all right, par. And then if from par. If par doesn't have anything on the right of it. If there's nothing here and it just has something on the left. This data goes here. This dot left gets this. This gets dot—OK, that makes sense. So now it shifting the tree up if there's nothing on the left. Now, if it's not, while, while not—. OK, now I made myself a nice table. 52, 52, add 53 and 51. All right, good, now I got a nice table. So what I'm going to do is look at the algorithm and I'm going to see what happens. OK, 55. I can print this out? *You can try.* I mean can I use it. I can't print it out. It is giving me errors. *You can not?* No. OK. I can do something to figure this out myself. 50, 55. So if, if the node dot right is null, so if there is nothing on the right, it just moves the one on the left up. If it has children on both sides what does it do? If children on both, if it has two children, it looks at the one on the left. If the one of the left's right is null then it moves the one on the left up. If one on the left's right is not null, then it moves what? It moves the right-most one of that all the way up. So, for example, if I do 50. This has one children. OK. It'll probably move 49. Let me try. Oh. What it's trying to do is—. That makes sense. I understand this now. I understand. It has to keep everything in order so it moves the one, because of the arrangement of the table. Oh my God, I can't believe it took me so long to do that. 49, right, if it says too sure it'll go to the right. And then what? Two children, it goes to the right. Par gets node dot left. Node gets node dot right. Two children. Par gets node dot left. OK, let me try. What the heck? *It's not supposed to be confusing, it's just the most difficult part of the algorithm.* OK, wait. If I look at the code. I guess if I had it printed out and I could look at how it works and tell what it's doing. Oh, you know what I could do. Oh, oh what happened? In order to delete 52, 52 is not— Data Structures. Capital D space capital S. d-a-t-a-s-t-r-u-c-t-u-r-e-s, space and capitals. Oh now I'm going to try insert sort in my I'll try method. Um. All right, OK, now, let me try inserting another number. It's going to tell me, 6, goes to the right. Another number 80, that goes to the right. Insert another number, 0, it goes to the left. —go the right of 0. 56 is less than 80. All right, 23, it's greater than 23, so moves to 56. Number to delete, 52. Click the other one? I don't understand what's going on. What, I am so confused. All right. Enter 26. It can't go anywhere. Here. OK. 31 goes to the right. 56. Good. 99 goes here, insert 67, go the left, goes here. Yeah, that's where I wanted to put it initially. Ah, insert 67. It's really confusing whether the branch is full or not because you don't know all the possibilities. *It's what?* It's confusing to say whether or not to say the branch is full because you don't know whether you can squeeze another number in there, for example. *Well.* Maybe what should happen is when you move the mouse down there it should show all the possibilities. *OK, all right, well keep going.* Anyway. 40 goes here. This branch is full. OK, now I thought it would be full, but it was—. Like I was saying it would be

better if when you move your mouse on top of possible candidates, it would automatically add that thing, yeah. 96, branch is full. I'm currently 44, 23, 56, 44. 15. One more to insert, 88 is full. OK, good, now let me try the finds. Click the next node. Now what? This is basically the procedure for adding. Very good, that's correct. Wow, this is pretty cool. Number to find, 15. Less than 22. Now I have the code and I can trace it properly. OK. Remove gets node. There's my node. Here's where I remove 30. So, if node dot right is not null, if par dot left not null, OK out 2 children. Par is node dot left. OK. Max is par dot right. If max is null then node dot left—then node dot left is par. How could par get node dot left, if left is there? Why did it go to the right? This is extremely confusing. OK, that's easier. 2, 23, now. What I going to do now is I'm going to follow the code. Remove the node, 23. Node dot right is null? No. Node dot is left is null? No. Two children. Par gets node dot left. Max gets par dot right, 21. Max equal to null? 21 is null? 21 is not null. Max dot right—. Max dot right initially is equal null. Par dot right. Par which is 0. Par dot right gets max dot left. Max dot left. OK, I absolutely do not understand. If it's at the ends I understand, but if it's like 26, I have no idea how it does it. I tried following through the code, too. It's just so—. I just don't get it. *Well, OK.* Can you explain it? Yes, I guess I can. I shouldn't but—. *What it does is it takes the inorder predecessor, which is it goes.* The inorder predecessor would mean that it, inorder would mean. *To find the inorder predecessor, you go one right, and then all the way left. So this is the inorder predecessor.* How come it doesn't do that here? *It does. Right here This part here that says.* OK, look, Two children. Par gets node dot left. Par would be node dot left. And, max gets par dot right, so that would be here. So, you're going left and then right. *No, it goes one right and then it keeps going left, left, left, left.* No, no, look. Par is node dot left. So node dot left would become par. And then, par dot right. *No, I don't think that's quite what it is.* OK. *What it does is goes one right and then all the way left. Takes that number and puts it up there.* OK, all right. *Then it, then it treats this as though it were one removed and does the delete.* It moves this up there and then what? *And then, then this is just a one, one with one child. It will always be at most one child. So it would then it would just move that up here.* OK, now I understand. I understand completely. So, look, I'll pick. OK, now, this is what I am going to delete. We'll see, one right and then all the way left, that's this. *Right.* OK, all right I understand, now I understand but I still don't think this code represents that properly. Or maybe I—, because look I remove this node, OK, node is what I want to remove. *It is written in a way that is a little bit obscure because it is a hard one to see.* Yeah, I didn't get it at all, but everything else I understood. This is just—. *What would have helped you?* I don't, let's see. *My explaining it?* Yeah, exactly, seriously, that's just—maybe if it was written there exactly how it did it. I guess if it was written there how it does it. *Did you read—* I tried it. I don't know.

## C.3 Priority queue with heap

OK, hello, all right. Um. OK, I started with the enqueue. And it added an element, 57, another element was added, 91. Add another element, 38. Now let me pick the element that it going to add, add a 1, see how that works. First right-handed available spot. Then, now if I add a 2, it should do it bottom there. Oh, why did it stick it on the left? Now, if I add 2, dequeue 38. OK, um, let me try it again. See what happens. Add a new number, enqueue. Whoa, way too fast. Let me slow it down, 2, 13. OK. 49, added there. It will probably switch with 32, that's all. I like doing it by single step better, I think it's easier to deal with it that way. Added 69. Why did it put it on the left? Oh, it goes to the right-most possible, the right-most possible. OK, all right. Now I'll switch it back. I'll do it again. Nothing will happen. Now if I enqueue let's say 99, what it should probably do is it stick it, um, stick it to the right of 57. Yeah, and then it'll switch there and it'll move itself all the way up. Ah, there we are. 90, for example. It stuck it here. Ah, it tries to—. Oh, what it tries to do is make sure everything is—. And then this'll step it up until the lowest element. So, if I want to seek something, for example, I just have to go and search for a number that's greater than 99, or, start off at 99 and then you look between 84 and 90—. OK, all right, now let's try dequeueing some stuff. This is very interesting to figure how it dequeues, let me pick. Um, Professor Jarc, this is slightly broken, I think. *What's wrong?* Even if I tell it I'll pick, and I say dequeue it just kind of automatically picks everything. *Well, the only place that you can pick is on enqueue.* Oh, really. *Right.* Why can't I tell it to dequeue 59? *Well, because there is only one thing that can be dequeued.* Oh. *That's why.* OK. So, all right, then I'll pick. *Just one thing as a clue, if you.* Oh. That's neat. I didn't know how that worked. *It will tell you some of the details of what applies where.* Ah, OK, cool. OK. That's very useful. I think that thing should be on whether or not you select it. *Do you?* Yeah. *It's kind of irritating, though.* Or if you click with the right button or something. I don't know. Bottom value is moved to the root. Let's try something. Bottom value is 1 now. OK, then, it'll transfer left and then heap to the left. It'll move 1 to 69. 69 then 57 moves up. Again, 3. 32 beautiful—. It's going to move everything, if I add a number like 53, it'll move everything up I think until it gets to the point, it'll be along the top and then it—that's cool. That's it. I have to show 4 more dequeues, 3, 2, 1. OK. This is pretty simple. I get this pretty easily. That helped. Now, submit it? For the priority queue thing, we don't have to try anything ourselves? *No.* OK. *Just as long as you—.* Understand it. *Yeah, and you have to, did you look at the progress button, you have to—.* Yeah, I submitted it. *OK, all right.*

## C.4 Height-balanced trees

Postorder traversal of the tree. First left, then right, then data. Height-balanced trees. Make tree. Oh, it's the upside root. OK, that's easy, all the leaves are one then everything else is defined by their furthest distance away from the leaves. That's easy to understand. Um. OK.

Now, I'll just submit that. I'll try, where's I'll try. Balance factors, whoa, whoa, balance factors, how does this work? 3 - 3 = 0, 2 - 0 = 2, 0 - 1 = -1, 0 - 0 = 0. Hmm. Ah, the factor difference between the height of its left and right subtree. Left minus right. Ah, OK. Yeah, OK, makes sense. Is AVL. If the tree on the screen is an AVL tree, otherwise the nodes of the tree with the improper balance factors. Oh, easy, yeah. If the node isn't 0, it displays it. What does this do? Ah. OK. It is height-balanced 4. What is HB-n? If the tree on the screen is height-balanced-n, otherwise the nodes of the tree with improper balance factors. Let me try 1. OK. I don't know about this AVL business, it's kind of confusing. Um. A height-balancing-n tree increases the most—. n and the height balance? Oh, I could be maybe the number of—OK. 2 is wrong. Let me try some, let me try—. It would be nice if we could maybe create our own tree there, and see, that'd be cool, and then see whether it's balanced or not. Oh, I see now, once looking at the Ada code I understand what the—the idea is behind this height-balanced trees business. Professor Jarc, is there a reason why we can't enter 1, for height-balanced, should we be able—? *Yes, well because AVL is 1.* Oh, yeah. OK, I see. *That's why.* All right. Ah, so this means that it should not be height-balanced-2. Hey. This will not be height-balanced-2, true. Yeah. OK. OK, this is simple enough. OK, I completely understand this. OK, I completely understand so I do it until I get 20 and then I'll submit it. OK.

### C.5 Categorized trees

Now, I'll try categorizing. Professor Jarc, you taught CS 131 last semester, didn't you? *Yes.* You taught Ada? *Yes.* Good. *I taught for the past 3 semesters.* Really, you were supposed teach it this semester? *I was supposed to teach it this semester.* How come you didn't. *Because the real-time class of that Professor Feldman was going teach didn't get enough students. It worked out better anyway. It was better for me to be in the lab.* Who was doing the lab last semester. *Jin-Song and Iddo.* We don't have to come if we don't want to. Right? *The project is actually due the last class.* Yeah the last Tuesday. Is almost? *Is-almost complete.* I didn't use help. I need help again. A tree is almost complete. Almost complete is AVL? *No.* OK. It is almost complete. Is it complete. No, no. OK, it's not complete. Almost complete means that there's—oh, I understand that, that's means that you have one or less rows only missing, I mean less than one row missing. That makes sense. A tree is a binary search tree. Yellow subtrees wrong. Yellow subtrees are wrong. OK, why is the yellow subtree wrong? That's a heap, isn't it? This is AVL. OK. Now, all I need to do is remember what a binary search tree is. Then I'll be set. OK. All right, now. What I need to do is determine what, remember what a binary search tree is. Can I look at the previous animations or previous text or whatever. It won't stop this visualization, will it? *Ah, well if you go to a different one?* Yeah. *It will, yeah, actually it will.* Oh, I just try to open it in another one, and see. Here's a binary search tree. OK, let me try opening it in a new window and see what happens. What the heck. This computer

is so ridiculous. Binary search trees. Oh. See, changing segments will stop visualization, so do I want to proceed or not? *Not unless you start the numbers over.* OK, I'll just look at it—. *Actually, I'm not sure whether it will reinitialize the numbers or not. Maybe it won't.* OK, we'll see. What was a binary search tree again? Ah. Professor Jarc, look at—. Now I'm going to return and deal with the binary search tree thing. Good for you. What is going on? Why is this taking so long? *Are you recording your thoughts?* Yeah, I'm trying to figure out what the difference between—. *Between?* Binary search tree and um. Well, I mean I know what the difference is. The only thing I'm trying—. Oh, oh. The only thing I'm trying to figure out, um. I know binary search tree is not in any particular order. Well, I mean it's not in the same type of order that the other type is, right? That the—I accidentally clicked on security and it's bringing up all—. I'll figure it out. One branch of tree. There are three fundamental operations, find, insert, and delete. The insert operation first performs the find operation. Oh, you know what? *What?* I think, if it's not a heap, it's a binary search tree. *I don't think that's true.* Well, I think it's true. *Well, if it's not a heap, it is a binary search tree?* Well in this example, anyway. *Maybe in that example, but not in general.* I have this whole discrete structures book full of nothing but that and I don't know what's going on in that class. *This should help your discrete structures.* Maybe. This is almost complete. Yeah, see I was right, it is almost complete. Ah, what else is this? Let's see is everything greater properly. This is a heap. Yes. This is not a binary search tree. I got this down cold. It's not complete and it is AVL. I got this down cold. OK, now let me try something else. Ah, OK. This is not almost complete. Oh, it is. Oh, yeah, it is, that makes sense. It is almost complete. It's not complete. Right. Since you only have to add less than one row, it's OK. It's a binary—let me see, is it? No, it's a heap. It's a heap. Is it an AVL? Yes, it is. Yes, it is. What does AVL stand for? *The names of two Russians or three Russians.* Oh, OK, I thought it had some designations. *I forgot exactly what their names are.* Make tree, Oh cool, this is big. This is not almost complete, yes, it's not complete. This is not an AVL tree. What on earth? It's not a—it is a binary search tree, no it isn't. Oh, of course, it can't be a binary because, because then it would have to have both sides, right? No. I don't understand this binary thing, I have to go back and do that after. But, yeah, it's not AVL. OK, make a new tree. Oh, this is complete, it's almost complete, obviously. Is it a binary search tree. No. It is a heap. It is AVL. Whoa, maybe, yeah. OK. It still have to do sixteen? Do I have to all sixteen of these things? *What do you mean?* Like do I have to do sixteen more of everything? *Yes.* Oh. *You don't have to.* The only thing I don't know about binary tree. I'm not sure how that works exactly. *Video games?* Shoot down the bad tree. Shoot down the bad node. *I'm leaving that to you.* Ah, hah, you know what? I just found a bug in my program, but that's not what I'm supposed to be doing. *Your mind is working in background mode.* Everything—nothing is a binary search tree. *There are some.* What—I don't remember how that works. How does that work? *Look at the definition.* Where is the definition? *In the text.* Textbook? *No, no, if you*

*look at the text, and you click on binary search tree.* Oh, cool. Maybe that should be the same thing in the visualizations. *Look at the definitions from there? That's possible. Remember, I'll give you an opportunity.* I'm saying them. OK, a binary search tree is a binary tree each of whose nodes contain a value greater than all the nodes in the left subtree and less than all the values in the right subtree. OK, that makes sense. Now, I can tell. OK, I understand that. Let me try start visualization again. Extremely, extremely slow. So slow. *It's better than Tompkins 410.* Slower than this? It's probably because they don't allow caching. They only have one meg free. *Why is that?* They only have one meg free on the computer. It's so ridiculous. Because when you use than computer it creates hundreds of temporary files and all the users have no permission to go and delete the temporary files, so like 300 meg of temporary files and everything and only the system operators can delete them, and there are like hundreds of lost clusters, it's just a big mess. I know, it's so slow. A binary search tree is a binary tree each of whose nodes contain a value greater than all the values in—. Ah, this is wrong because node 58 is wrong. It's not a binary search tree. I understand this now. Bye.

### C.6 Heap sort

OK. Let's try. Turn everything green, I mean red. The bottom row is turning red. That makes no sense. Keeps translating this stuff up, until everything—makes sense. OK, yeah, this makes sense. This is extremely weird. Let's try again. Um, let me see. Make a new tree. Sort. Blue value is swapped the red—OK. Abort. Abort. I'll try. Make tree. OK, now I'm trying. Sort. No, I'm not. I'm holding it close to me. OK. Click nodes to swap.. Very good, that's correct. Oh, OK. All right, it starts from the right, I guess. Now, it goes back from here. All right, OK, it sweep from the right, it looks like. All right, there, switch with that, then 85, no, no, I mean 99 switches with that 58. 99 will switch with 68, no, oops, 99 will switch with—. Hmm. All right, this one will go there. Oh, whoops, OK, now, no. OK, I'll run it again. OK, um, ah, now, let's see we continue along. This is kind of weird. Oh, now I got to switch 99 with 56 which is right-most one on the lowest row, and then what, it does it all over again? Let me look at my progress. Not correct, try again. 52, 68. OK, what happens now? I don't know. What happens? That switches with that. Why was that? I don't see the pattern at all. OK, click the other one. It switches with 82, good, now this switches with this. No. Crap. This switches with that. Oh, it switches with that one. Oh. Oh. I see. That switches over there. OK, I think I generally got the idea here. OK. All right. Now, let me try click nodes to swap. 82 with the one down there. Oh, that's weird, I wonder why it's doing that. A weird way of sorting a tree. OK. It's moving everything around everywhere. This is so weird. It's so weird. OK, I think it's done. Oh, oh I see. I understand this. Let me try a few more. I'm done.

## C.7 Quadratic sorts

OK, the text for quadratic sorts is extremely useful and easy to understand. OK, now I'm going to try to make one of these arrays. It made one. Now, I'm going to try bubble sort and see what happens. Nothing seems to be happening. Oh, OK. I see how it works. Oh, it just swaps if it's higher, it swaps it if lower it doesn't—. This is way too fast. Oh, there, I slowed it down. What happens, I suppose, is that—, oh yeah, if what you're comparing is lower it pushes it to left or else it leaves it alone. That seems to make sense. OK. Stop. Abort. Progress. Now let me try, um, doing it myself in the I'll try. Make array. Bubble. OK, all right, I understand it. If it's lower, you swap, and if it's higher you leave it alone. So, this is very easy and intuitive. I have to do 12 more. I have to do 3 more of them and then I'm done. Yeah, this is very simplistic. I like the bubble sort. It is an easy concept to understand. So, good, now, let me try, abort. Show me. Now I'll try insertion sort. Oh, I see. So what does is each time it finds something, it tries and put it in the place of some, of everything already been sorted. OK. Makes sense. Yeah. All right, that's easy. Let me try a selection sort, or do I need to see more. I'll try a selection sort now. Ah, that's dumb. I better make a new array first. Do the selection sort. I'd be nice if it slowed down, it's way too fast. It's way, way, way too fast, OK, Stop. OK, let me try. OK, yeah, so selection just basically, um, picks each one that hasn't been done yet and swaps it with the lowest one that not there yet. OK, easy enough. 9, 31, 29, 36. There's something between 31 and 36. These—I think maybe these numbers should be, um, like the graph should be easier to distinguish because even if you understand it sometimes it's confusing because they're almost the same size. That would make it much, much easier to deal with. 36. What? Ah. OK, this is easily understood. It's just kind of tiresome that we have to do 30 of them, but—. Yeah, especially the insertion sort, it just takes forever. 96 will be replaced with the smallest one here. 72. 84. Oh 84, the thing with like 84 and 85 and stuff, it's extremely difficult to see what's what. OK, good, I'm done. OK, now them me try the other one. Abort. Selection sort is—yeah, it just searches them out and does it. Yeah, OK, that's easy to do. I don't want to see, whoops, how to insertion. Oh crap. Oh. OK, what does insertion sort do? Oh, it just kind of sticks it in. Oh, I've seen everything. Whoops, stop, stop, stop, stop, stop. OK. Make array. I'll try. Insertion. 13, 82. What? Oh, I see, it just does that between—oh, OK, that's pretty simple, it just puts the each bar between two other bars, but the next bar is moved to wherever it's supposed to go. OK. That's easy enough. That's what I told it—. Ah, OK, it quibbles whether you put it after it or before, it seems to have a problem with that.

## C.8 Efficient sorts

OK, um, I'm going try merge sort. What the hell? What on earth? Whoa. What on earth? Oh, I see what it seems to be doing in merge is it does the first 2, then 4, and then 8 and then 16. Ah, interesting, interesting, interesting. OK, now I understand it. Have I seen enough to go for

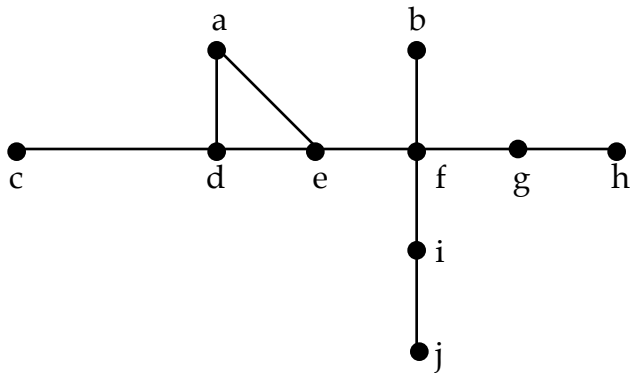
it? 48 Stop. Make array. I'll try. Merge. Oh, oh, oh. I see. I see. Oh, I understand. This, this, the way it works, it takes a few clicks to figure out how this works. But, I still have no clue how it works. Um, oh. It takes a while to figure how this thing works, but afterwards, it's working! Did you do this thing yet? 10 more of these things. I keep screwing this up, it's so complicated. Yeah, I know. Oh, good. How man? What, what? Why is it putting those things up? Oh, oh, I see. It does the last one, then all the eight, which is pretty stupid. It should be able to store more than one click in the memory. *What?* It should be able to store more than one click in the memory. Yeah, I just keep clicking, and then it—. *Are you done, or are you going to*— I'm like almost done. *You're almost done?* Oh good, I don't need any more, I just need to do quicksort. Well, I mean yeah, I have 0 needed. Whoa, cool. What do mean click bars to swap. I don't even know how it works. Oh, this idea with the line thing is pretty cool. The line on quick sort makes it much easier to understand. *Good, good, I added that.* Yeah, because the other ones didn't have it and it was very—. What the? Wait a minute. What the? It goes way too fast! Single step. Click. Oh, step. Oh, I see what it does. Everything greater than the first thing it pushes somewhere else or something. I don't, I have—. Let me just—. I see. I understand how this works. It swaps the—it swaps anything that's above the line with the first one from the right that's below the line. So, everything from the left that's above the line, it swaps with something from the right that's below the line. Now what? Oh, OK fine. OK. What did it just do? *Are you almost done?* I'm almost done. I have like 10 more to go. Now, what do I do? Click the other one. Oh, it moves it to the next higher one and—. OK, this is easy to do. This I can deal with. I have absolutely no idea how that works. OK, I only need 5 more. Oh my God. I was right. OK, yeah, oh, this seems to make complete sense to me all of a sudden.

# APPENDIX D POSTTESTS

## D.1 Posttest for first experiment

The posttest for the first experiment was the final examination. Only the first three problems of the examination that were used for ther study are included.

PROBLEM 1: (13 pts) Consider the following undirected graph.



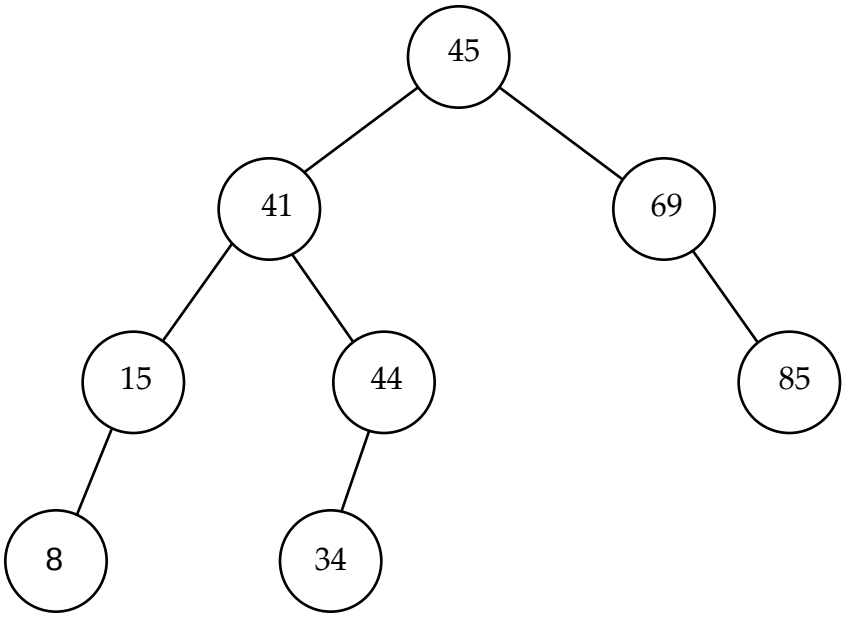

(A) (1 pt each) Indicate whether the graph has each of the following properties (Circle Yes or No):

| | | | | | |
|---|---|---|---|---|---|
| i. | Is it connected? | Yes | _____ | No | _____ |
| ii. | Does it contain cycles? | Yes | _____ | No | _____ |
| iii. | Is it an undirected tree? | Yes | _____ | No | _____ |

(B) (2 pts each) Consider the above graph and indicate whether each of the following lists of nodes is a breadth-first search of that graph, a depth-first search, neither or both. Node c is the assumed starting node.

i. c d a e f b i g j h
ii. c d a e f g h b i j
iii. c d e a f i g b j h
iv. c d e a f g i b h j
v. c a d e f b g h i j

PROBLEM 2: (25 pts) For the following binary tree, answer (A) and (B):




(A) (4 pts each) List the vertices of this tree in the order resulting from each of the given traversals.

i. NLR order: _____
ii. LNR order:: _____
iii. RLN order: _____
iv. Level order: _____

(B) (1 pt each) Indicate whether the tree has the following propertires (circle Yes or No):

| | | | | | |
|---|---|---|---|---|---|
| i. | Is the binary tree complete? | Yes | _____ | No | _____ |
| ii. | Is it almost complete? | Yes | _____ | No | _____ |
| iii. | Is it a binary search tree? | Yes | _____ | No | _____ |
| iv. | Is it an AVL tree? | Yes | _____ | No | _____ |
| v. | Is it a heap? | Yes | _____ | No | _____ |

PROBLEM 3: (20 pts) Consider the following array of integers:

```
49  84  10  70  36  19  41  93
```

(A). (10) Show a trace of each step of the selection sort At each step, explicitly show which values are being swapped by circling those two values.

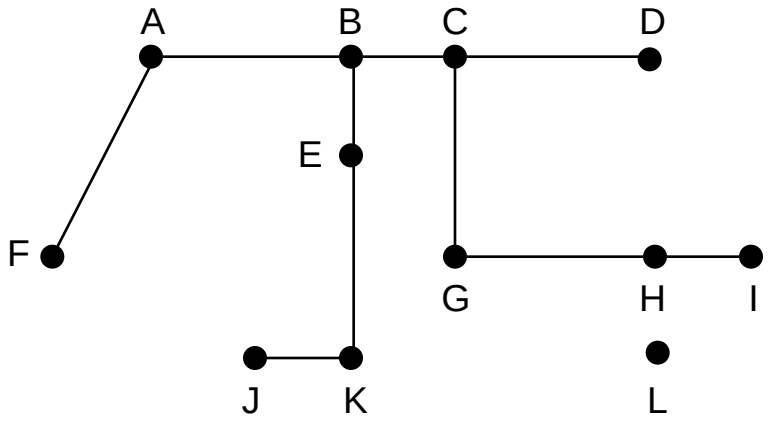
(B). (10) Show a trace of the quicksort to the point which the value `49` is placed into its proper position. At each step, explicitly show which values are being swapped by circling those two values.

## D.2 Posttest for second experiment

**CSci 131 - Data Structures**
Quiz 1

Name: _____

1. (3 pts) Consider the following undirected graph.




For the above graph, circle the correct answer for each of the following questions:

| | | | |
|---|---|---|---|
| a. | Is it connected? | Yes | No |
| b. | Does it contain cycles? | Yes | No |
| c. | Is it an undirected tree? | Yes | No |

2. (4 pts) For each of the following lists of the nodes of the graph from the previous problem, circle either *Depth* or *Breadth* depending on which search would generate it, or select *None*, if neither search would produce that list, or select *Both* if both searches would produce it.

a. A F B C D G H I E K J Depth Breadth None Both

b. A F B C D E K J G H I Depth Breadth None Both

c. A B F C E D G K J H I Depth Breadth None Both

d. A F B C D G H I E K J L Depth Breadth None Both

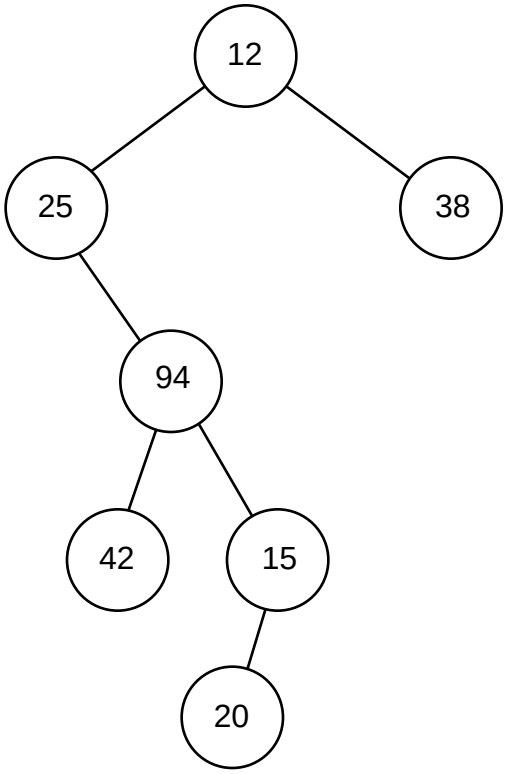
3. (3 pts) For the graph of the previous problem, construct the adjacency matrix that represents it.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | | | | | | | | | | | | |
| B | | | | | | | | | | | | |
| C | | | | | | | | | | | | |
| D | | | | | | | | | | | | |
| E | | | | | | | | | | | | |
| F | | | | | | | | | | | | |
| G | | | | | | | | | | | | |
| H | | | | | | | | | | | | |
| I | | | | | | | | | | | | |
| J | | | | | | | | | | | | |
| K | | | | | | | | | | | | |
| L | | | | | | | | | | | | |

**CSci 131 - Data Structures**
Quiz 2

Name: _____

1. (2 pts) Given the following binary tree:

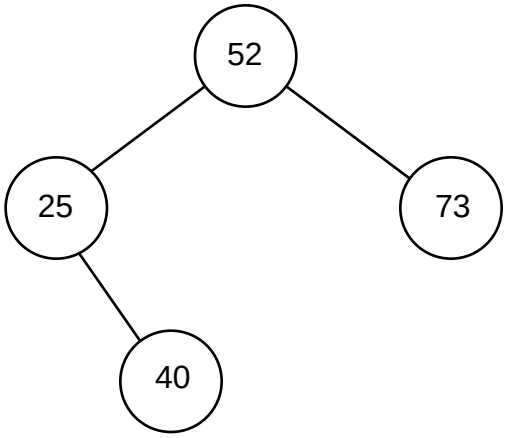



For each of the following, circle one of the four tree traversals that would generate the enumeration, or circle *None*, if none of the traversals would.

a. 20 15 42 94 25 38 12 Pre In Post Level None

b. 25 42 94 15 20 12 38 Pre In Post Level None

2. (2 pts) Given the following binary search tree:

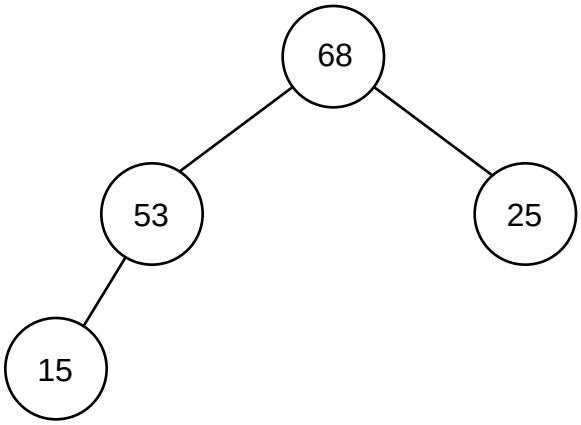



For each of the following values, circle the position where it would be inserted in to the above binary tree:

a. 30 Left of 25 Left of 40 Right of 40 Left of 73 Right of 73

b. 18 Left of 25 Left of 40 Right of 40 Left of 73 Right of 73

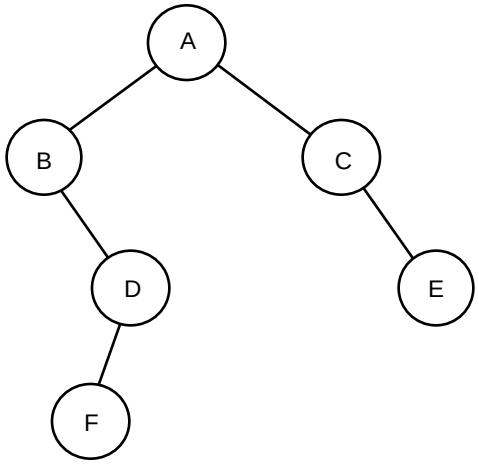
3. (1 pts) Given the following binary tree representation of a priority queue:




Circle the correct choice for the following:

The second value dequeued would be: 68 15 25 53

4. (2 pts) Given the following binary tree:

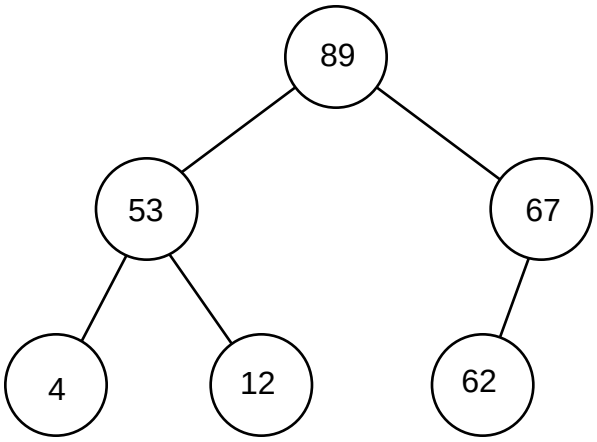



Circle the correct choice for each of the following:

a. The balance factor of node B is -1 3 -2 1 0

b. Is it height-balanced-2? Yes No

5. (3 pts) Given the following binary tree:




Circle the correct choice for each of the following:

a. Is the binary tree complete? Yes No

b. Is it a binary search tree? Yes No

c. Is it a heap? Yes No

**CSci 131 - Data Structures**
Quiz 3

Name: _____

1. (3 pts) Given the following array of integers:

47 29 61 53 63 76 37 77 44 45 49 13 24 35 85 18 79 54 83 10

For each of the following pairs of values circle *First*, if it is first pair of values that would be swapped by the selection sort algorithm, *Second*, if it is the second pair and so on. Circle *None*, if that pair would not be among the first four pairs swapped.

| | | | | | | |
|---|---|---|---|---|---|---|
| a. | 61 18 | First | Second | Third | Fourth | None |
| b. | 29 13 | First | Second | Third | Fourth | None |
| c. | 63 35 | First | Second | Third | Fourth | None |

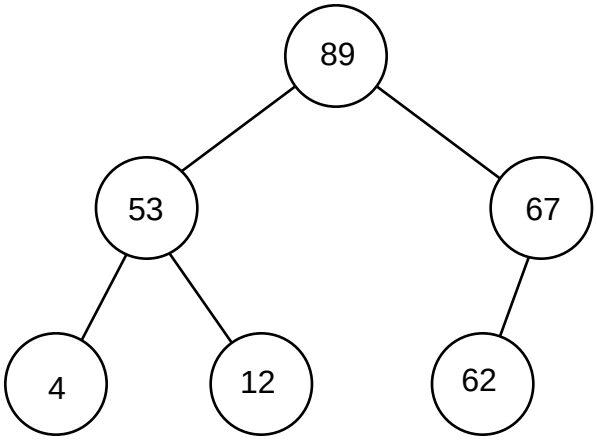
2. (4 pts) Given the following array of integers:

47 29 61 53 63 76 37 77 44 45 49 13 24 35 85 18 79 54 83 10

For each of the following pairs of values circle *First*, if it is first pair of values that would be swapped by the quick sort algorithm, *Second*, if it is the second pair and so on. Circle *None*, if that pair would not be among the first four pairs swapped.

| | | | | | | |
|---|---|---|---|---|---|---|
| a. | 53 18 | First | Second | Third | Fourth | None |
| b. | 76 24 | First | Second | Third | Fourth | None |
| c. | 61 10 | First | Second | Third | Fourth | None |
| d. | 63 35 | First | Second | Third | Fourth | None |

3. (3 pts) Given the following almost complete binary tree that has been converted into a heap during the heap sort algorithm:




For each of the following pairs of values circle *First*, if it is first pair of values that would be swapped first as the next step of the heap sort algorithm, *Second*, if it is the second pair and so on. Circle *None*, if that pair would not be among the next four pairs swapped.

| | | | | | | |
|---|---|---|---|---|---|---|
| a. | 62 67 | First | Second | Third | Fourth | None |
| b. | 89 4 | First | Second | Third | Fourth | None |
| c. | 12 62 | First | Second | Third | Fourth | None |