INVESTIGATING CHARACTERISTICS OF EFFECTIVE PROGRAM VISUALIZATIONS: A TESTING ENVIRONMENT AND THE EFFECT OF COMPARISON CUEING AND EXCHANGE TECHNIQUES ON VIEWER COMPREHENSION IN ALGORITHM ANIMATIONS

by

BINA N. REED

(Under the Direction of Eileen Kraemer)

ABSTRACT

We describe SSEA, a **S**ystem for **S**tudying the **E**ffectiveness of **A**nimations, and an empirical study that evaluates two visualization design attributes. SSEA was created as a testing environment for studying the effects of various attributes in visualization design on viewer comprehension. Researchers create a series of animations in SSEA with a design characteristic in mind. SSEA allows these animations to be viewed while recording the viewer's interactions and responses to questions about the underlying algorithm. At the conclusion of running all experiments, the researchers can examine the log files generated, and perform analysis of the responses and timings with respect to the attribute being examined. The first in a series of studies to be conducted examined the attributes: comparison cueing and exchange techniques using traditional and popup questions, measuring comprehension and perception, respectively. No significant effects were observed in comprehension questions. Significant effects were found in the perceptual questions.

INDEX WORDS: algorithm animation, visualization system, empirical study, visualization design

INVESTIGATING CHARACTERISTICS OF EFFECTIVE PROGRAM VISUALIZATIONS: A TESTING ENVIRONMENT AND THE EFFECT OF COMPARISON CUEING AND EXCHANGE TECHNIQUES ON VIEWER COMPREHENSION IN ALGORITHM ANIMATIONS

by

BINA N. REED

BAE, The Pennsylvania State University, 1993

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2006

INVESTIGATING CHARACTERISTICS OF EFFECTIVE PROGRAM VISUALIZATIONS: A TESTING ENVIRONMENT AND THE EFFECT OF COMPARISON CUEING AND EXCHANGE TECHNIQUES ON VIEWER COMPREHENSION IN ALGORITHM ANIMATIONS

by

BINA N. REED

Major Professor:	Eileen Kraemer
Committee:	Robert W. Robinson
	I. Budak Arpinar

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2006

DEDICATION

To my family.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Eileen Kraemer, for her guidance, wisdom, clarity, and support. I also would like to thank Dr. I. Budak Arpinar and Dr. Robert W. Robinson for their valued input and guidance. Furthermore, I would like to thank the VizEval team, especially Philippa Rhodes for her words of wisdom as well as her friendship.

Additionally, I owe much to my family for their constant support and encouragement. To my parents, I thank you for your love and support whenever I needed it. To my sister and her husband, I am grateful for all of your love and encouragement as we each earned our degrees. And I am especially indebted to my husband and sons. To Rob, who saw me through every high and low, and still thought I was sane. To my son, Corey, who could not wait for the perfect birthday: the day before a final. He patiently sat in my courses, providing an out-of-place giggle every now and then. Finally, to my son Zachary, who grew up knowing my homework needed to be done before a play date. I am ever grateful for being blessed with such people in my life.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 Introduction 1

2 Background and Related Work 5

2.1 A Brief History of Algorithm Animation 5

2.2 The Value of Visualizations 6

2.3 Effective Visualizations: Viewer Comprehension 11

2.4 Summary 18

3 SSEA System Description 19

3.1 System Design Considerations 19

3.2 Implementation 20

3.3 Specifying an Empirical Study 28

4 Experiment Materials and Methods 30

4.1 Experiment Description 30

4.2 Quicksort 33

4.3 Questions 35

5 Experiment Results and Discussion 38

5.1 Traditional Questions 39
5.2 Popup Questions 40
5.3 Correlation between Popup and Traditional Questions 40
5.4 Discussion 41
6 Conclusion 44
REFERENCES 46
APPENDICES 50
A Sample Project Files 50
B Sample Session Log File 52
C Questions for Experiment 54
D Popup Questions for Experiment 59
E Questions in Questionnaire 61
F Experiment Results 63

LIST OF TABLES

Page

Table 5.1: Average time and score per group, traditional questions..... 39

Table 5.2: Average scores per group on popup questions..... 40

Table F.1: Result Summary for groups MC and MX..... 63

Table F.2: Result Summary for groups GC and GX..... 64

LIST OF FIGURES

Page

Figure 3.1: SSEA architecture 21

Figure 3.2: SSEA screen shot 22

Figure 3.3: Animation state diagram 23

Figure 3.4: SSEA screen shot of questionnaire 24

Figure 3.5: Architecture of SKA module 26

Figure 3.6: Animation Engine Details 27

Figure 4.1: 2x2 Factorial Design for Experiment 31

Figure 4.2: Screenshot of quicksort visualization 35

Figure 5.3: Correlation between Popup and Traditional Scores 41

Figure A.1: A sample SSEA.xml file 50

Figure A.2: Sample xml entries for questions 50

Figure A.3: Sample xml entry for a popup question 51

Figure A.4: Sample visualization input xml entry 51

Figure B.1: Sample session log file 52

Figure B.2: Sample session log file 53

Chapter 1

Introduction

Understanding, developing, and debugging computer programs is a complex human activity. Software visualization (the visualization and animation of data structures, programs, algorithms, and processes) has the potential to be useful for helping students learn how programs work, for assisting professional software engineers in debugging and understanding their code, and in providing researchers with insights to analyze and improve algorithms. These visualizations fall into two categories: *program visualizations* and *algorithm animations*. *Program visualizations* are representations of the actual program itself [Price98]. *Algorithm animations* depict the higher-level concept of what the program is doing [Price98] by using movie-like animated graphics.

In the field of Computer Science, it was thought, algorithm visualizations could be used as a pedagogical tool to help students understand algorithms. These algorithms can be complex and difficult to understand merely by looking at pseudocode and textual descriptions. Many studies focusing on whether the additional use of algorithm animations is beneficial have yielded mixed results [Lawrence94, Lawrence93, Hansen00]. Naturally, further research has been conducted to determine the causes for this [Naps03, Hundhausen02]. However, one factor that has been overlooked in these studies and that needs examination is the effect of the design of the animations themselves on the degree to which viewers benefit.

When creating visualizations, consideration should be given to the user's perception, the user's learning style, and to graphic design standards. For animations to be beneficial, these factors need to be studied to determine how they interact so that animation creators may better

design displays that take good advantage of the viewer's perceptual, attentional, and cognitive abilities.

To understand why certain attributes may be more effective than others in helping the viewer to understand the animation and the underlying program or algorithm, we must turn to the study of cognitive psychology. Cognitive psychology studies how mental processes are received from the senses and then are "transformed, reduced, elaborated, stored, recovered and used [Neisser67]." From this and related fields, we can gain insight into how dynamic images on a screen are processed by a person viewing them.

As part of a larger ongoing research project funded by the National Science Foundation and entitled *Program Visualization: Using Perceptual and Cognitive Concepts to Quantify Quality, Support Instruction and Improve Interactions,*[1] this thesis focuses on developing design guidelines for program visualizations. In this work we examine the effects of various attributes of visualizations. Through empirical studies we seek to determine the effects of various attributes on a viewer's comprehension of the visualization. A compilation of these results may then be used by educators, researchers and developers in creating more effective algorithm and program visualizations.

To examine the effects of a particular attribute, we isolate the feature in question in a series of visualizations and evaluate the effects of this attribute in an empirical study. Here we look at a sorting algorithm (in which rectangular bars represent values in an array) and examine the effects of two particular attributes of many such algorithm animations: the use of flashing to

[1] This material is based upon work supported by the National Science Foundation under Grant No. 0308063. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

cue a comparison of values and the use of two different animation styles to depict the exchange of elements.

We label the first attribute as “cueing” and “no-cueing”. The cueing depiction flashes the bars being compared three times when a comparison event occurs. The no-cueing animations simply do not have any flashing for a comparison event.

The second attribute’s depictions are labeled as “grow” and “move”. During an exchange of values, the grow animation shows the smaller element growing to the height of the larger element while the larger element shrinks simultaneously to the height of the smaller element. In the move depiction of an exchange of values, the two bars representing the data elements being exchanged are seen to move in an arcing motion, each one moving to the location of the other.

Four animations were created: cueing with swap, cueing with grow, no cueing with swap, and no cueing with grow. Each of these animations are the same except for the attributes we are investigating.

An interesting aspect of this study is that we seek to simultaneously evaluate the perceptual effects of these animation techniques, and the effects of the animation techniques on comprehension of the algorithm depicted in the animation.

By perceptual effect we refer to the viewer’s ability to see and understand what is animated (i.e., did users detect that two bars just flashed, do they understand that the flashing means that the two bars are being compared to one another?, etc.) Perceptual effects are evaluated through responses to popup questions (questions that appear during playback of the animation, where the animation pauses its run until an answer is provided).

By effects on comprehension we refer to the ability of viewers to understand an algorithm, as measured by the number of correct answers to questions about properties of the

algorithm. The users' responses to "traditional" questions measure the comprehension effects of the algorithm animation.

Our goal in simultaneously collecting data about both perception and comprehension is to help understand and explain the results of the comprehension studies. In prior studies we, and other researchers, have often been left to wonder whether users had not perceived the low-level details of the algorithm or whether the low-level details had been perceived but had failed to translate into comprehension of the higher-level algorithm behavior and properties. By collecting such data we hope to be able to begin to answer this type of question.

A sorting algorithm was chosen not only for its historic value (one of the first algorithm animations is the video, *Sorting Out Sorting* [Baecker98]), but also for its simplicity. The idea of sorting is very elementary. However, quicksort uses recursion for its divide-and-conquer strategy, which is somewhat more complex than other sorting algorithms, and thus may benefit more from visualization. Further, this algorithm provides an appropriate level of challenge to the students participating in our study.

In Chapter 2 we discuss related work in the study of how viewers understand algorithm animations. The experimental method and setup are discussed in Chapter 3. In Chapters 4 and 5, we present and analyze the results. A discussion and conclusion are presented in Chapter 6.

Chapter 2

Background and Related Work

Here we examine the history of algorithm animations and the research around the effectiveness of animations as pedagogical tools. We highlight guidelines presented on how to conduct empirical studies in this area. Following this, we focus on related research that draws out issues in our investigation. All areas of visualization, including engineering, medicine and the sciences, draw from the fields of information graphics, computer graphics, human-computer interaction (HCI), and cognitive science. Here we address research done on viewer comprehension and how design of visualizations should consider viewer cognition and attention. Finally, some researchers in algorithm visualizations have taken into consideration these factors; their work is discussed here.

2.1 A Brief History of Algorithm Animation

Consider a class of undergraduate computer science students studying algorithms and data structures in the early 1980s. The educational technology available at the time consisted of chalkboards and overhead projectors. On occasion, a video was employed. For example in the late 1970s, a landmark algorithm animation video on sorting algorithms, *Sorting Out Sorting*, was created [Baecker81]. Instructors would typically have to create still pictures on overheads or the chalkboard showing the step-by-step process of a particular algorithm. For the most part, these tools were not very conducive to allow an instructor to demonstrate the dynamic workings of the programs being discussed.

However, at about this time computer workstations became more available to universities. So naturally, it seemed inevitable that the medium would shift from low fidelity static displays to the high fidelity dynamic displays of the computer. Initially videos were created [Baecker81]. Then in the mid 1980s, an animation system called BALSA was created at Brown University and integrated into the coursework [Brown98a]. BALSA provided a framework through which instructors could create animations. All students in a class or lab section could then simultaneously view the animation of an algorithm.

In the late 1980s, systems like TANGO allowed students to create their own animations [Stasko90]. In the following years, with each technological advance in graphics and computational power, more could be expressed in the animations, adding color, three-dimensional graphics, and audio [Price98]. Instructors in recent years rely, in addition, on PowerPoint slides, Java applets, Macromedia Flash movies, and web pages with graphics as pedagogical tools.

2.2 The Value of Visualizations

A survey of instructors to determine the use of visualizations as pedagogical tools found they were not as widely used as expected [Naps03]. The question then arose of how valuable visualizations are as a learning aid. Naps *et al.* [Naps03] point out that two main obstacles prevent wide usage. First, it must be determined whether viewers benefit educationally from visualizations. Second, instructors must find it easy to incorporate visualizations as a learning tool.

Much research has focused on the viewer's perspective, determining the effectiveness of visualizations. A good portion of this research compared the benefits of learning algorithms with

visualizations to learning algorithms without visualizations [Stasko93, Hansen02, Grissom03]. Fewer studies examined the quality of the attributes that made up the visualizations [Lawrence93, Bartram01]. As Khuri [Khuri01] notes, "a successful algorithm visualization design should consider effective representation and presentation of information." He states that designers should combine instructional knowledge with key aesthetics that will grab viewer's attention. Accordingly, the designers should have a graphical vocabulary allowing viewers to easily understand the information being conveyed.

Graphical vocabulary [Price98] typically refers to the representation of data structures. Each graphic can have various properties associated with it, such as dimensions, color, position, texture, and shape, just to name a few. However, less mentioned is a graphical vocabulary for important events in animations. Bartram agreed, indicating the lack of research on the effectiveness of motion used as a visual coding of events [Bartram01]. Byrne et al. [Byrne99] call for systematic studies to find quality animations stating that there are "no clear guidelines for the construction of algorithm animations, not just from a psychological perspective but from an implementational one."

Gurka and Citrin [Gurka96] present guidelines on how to perform studies testing the effectiveness of algorithm animations. They list factors that could lead to false negative results, in hopes that future studies might avoid these problems so that "significant results can be reliably obtained and replicated." Three issues that apply to experimentation in general are given. First, qualitative and quantitative data must be collected. An example of qualitative data gathered in experiments are participant surveys. Tests administered to subjects are a form of quantitative data. Second, independent variables (*e.g.*, demographics) and dependent variables (*e.g.*, style of material) should be identified. Then the data should be analyzed with respect to these variables,

being careful not to claim one factor produced the results. Gurka and Citrin state that while some variables can be controlled, other factors cannot. This means that results need to be combined with the qualitative data in order to get an overall understanding of the algorithm animation's effectiveness. The last issue dealt with the comparisons done between the experimental group and control group, which often compared "something" to "nothing". That is, many of the visualization studies compared participants viewing animations to those who did not [Lawrence94, Stasko93]. Gurka and Citrin point out that conclusions from these studies need to consider the extra time and attention the visualization group has as compared to the control group.

Gurka and Citrin also detail seven design factors, most specific to animations. The factors are: usability, animation quality, systems training, system availability, animation type, algorithm difficulty, and subjects' individuality. One of these, animation quality, they describe as "probably the most difficult issue to tackle." This is the factor that is of particular interest to our research. They identify graphic design as a contributor to the quality of animations.

It is believed that the features in a visualization design can impact the viewer's comprehension of the underlying program. Jarc addressed this need with a call for more empirical studies comparing various forms of animating the same algorithm [Jarc99]. This would help identify the graphical representations, which Jarc refers to as visual semantic cues, that have an impact on viewer understanding. He indicates that guidelines can be formed for high-level algorithm visualization design, as well as for low-level graphical features.

In Jarc's doctoral dissertation [Jarc99], an interactive algorithm visualization system he developed was used in two empirical studies. The studies examined how the level of interactivity of the viewers with the system related to their performance on a posttest.

Surprisingly, the results showed marginally better performance for the groups with the more passive viewing and fewer interactions. Jarc claims this result is due to active learning-style students who treated the interactive portion of the visualization as a video game. He claims that these students' focus during the experience was on entertainment and not on learning, and that they thus had to guess on the posttest. Given these results and explanation the question of how interactivity effects viewer comprehension remains open.

2.2.1 Empirical Studies with Graphic Representation Considerations

A number of studies have been conducted examining the effects of various design considerations in graphical displays [Davison01, Bartram01, Lawrence93]. We concentrate here on a few studies to draw out some general issues related to our investigations.

Davison and Wickens [Davison01] studied graphical displays used by pilots and tested the effects of cueing. Though outside of the computer science field, the results are relevant in studying the value of various characteristics in graphic displays. They examined the effects of two types of salient cues, flashing and intensifying brightness, in a helicopter flight simulator that signaled a hazard. One measure of performance was based on the pilots' subsequent flight path, examining if the path would make contact with the hazard. If the path was within 50 feet of the hazard, it was considered a hazard contact. The other performance gauge was the anticipation time for when the hazard was avoided. This measured the time the pilot began maneuvers to avoid the hazard. They found marginally better anticipation times for the group with hazard cues than for the group with no hazard cues. There was no difference in hazard contact performance when comparing these groups. Additionally, contrary to their hypothesis that flashing is a more salient cue, they found a significantly greater performance for the

intensity cue. They explain that this may be because flashing is more distracting and thus impedes performance.

Bartram [Bartram01] states that "motion and movement is considered one of the most powerful visual mechanisms for communicating information." In a series of experiments, viewers were instructed to work on a main task that required users to focus on a small area. To one side of the display area were icons. When viewers saw a change in one of these icons they were to stop working on the main task. Various graphical changes were tested on groups using either motion, color, or shape. The results showed the statistical significance of simple motion attracting a viewer's attention as compared to color and shape changes. Results were based on detection times and error rates of the moving icons versus either icon shape or color changes.

Lawrence [Lawrence93] performed a number of experiments studying the effects of various design elements in algorithm visualizations. In a survey of student preferences, she found the preferred representations of algorithm data and the preferred usage of labels with data. Follow up experiments examining these preferences and data set size found no significant improvement in performance. An additional study found that labeling data elements had no effects on viewer comprehension. Lawrence mentions that though labeling the data has no significant results, there is no negative impact to their inclusion in the visualization and was a preference on the student survey. In another formal investigation of a student preference, the labeling of algorithm steps, it was found that this labeling resulted in a statistically significant increase in post-test scores [Lawrence93].

2.3 Effective Visualizations: Viewer Comprehension

In a study dealing with animations, multimedia, and virtual reality, i.e. "graphical representations that were born from advances in technology", Scaife and Rogers point out the following: "Many of the presumed benefits of good-old fashioned graphical representations (i.e. static diagrams) were considered to be due to years of practice of perceptual processing of visual stimuli and the learning of graphical conventions. This may help us understand why advanced graphical technologies (e.g. animations and virtual reality) have not, as yet, been able to demonstrate comparable performance or learning benefits" [Scaife96].

Algorithm visualizations are comprised of graphical representations of the input to the algorithm, the events and procedures conducted by the algorithm, and the final results. Therefore an animation consists of a series of dynamic graphical events. Animations have also been defined as "a series of rapidly changing static displays, giving the illusion of temporal and spatial movement" [Scaife96]. These graphical events include motion-based events such as object flashing, object movement, and changing object dimensions. Additionally, attention attracting events include highlighting, pop-ups, and the use of sound. Our focus is on the extent to which these low-level graphical features help or hinder the viewer in comprehending the algorithm depicted. The viewer's overall understanding of the animation relies on these features since a viewer's attention can be focused using graphical events indicating that something important is happening. A viewer can then shape and form a mental model through cognitive processing.

To understand a presentation, a viewer "requires a series of cognitive processes composed of visual and auditory attention, comprehension into a proposition, and integration into a mental model." [Faraday96] Here Faraday and Sutcliffe refer to a proposition as a "unit

of meaning," which can be one of three types: object, action, or procedure propositions. Object propositions describe objects in the presentation. Action propositions then describe object changes of state, role or path. Finally, procedure propositions define the cause and effect of a sequence of action propositions. They claim this is the basis upon which the viewer forms a mental model. The combination of many propositions in the final presentation represents the meaning of information being presented.

Thus, perceptual, attentional and cognitive properties in the visualization impact the creation of mental models and call for designs that do not place undue burden on the viewer's cognitive load. Being aware of which attributes are less demanding when the viewer is learning about an algorithm can contribute to the creation of displays from which the viewer may more easily learn. For example, when an adult learns a new language, more thought often goes into translation from his/her native language to the foreign language than on the thought being expressed. This is a form of cognitive load. Therefore, displays should be created to make use of attributes that are more efficient for viewers to perceive and process using the visual attention system rather than using cognition. Additionally, we need to be sure we eliminate any visual noise that can distract from either the visual attention system or cognitive processing.

In order for us to create guidelines for graphical representations, we must determine the effects of what is being viewed. Scaife and Rogers emphasize the "paucity of work on determining how graphical representations are themselves represented and how this interacts with the kinds of high-level cognitive processes" [Scaife96]. Similarly, Faraday and Sutcliffe define good design to be a transferring of knowledge intended by the designer and understood by the viewer [Faraday96]. They further point out there is a danger in designing without regard to attention and comprehension.

2.3.1 Cognition

Information in animations should be designed to be processed sequentially and in distinct state changes [Scaife96]. This notion is similar to Brown's "interesting events" [Brown84]. Though the latter uses the term to describe code annotations that serve as input to visualizations, it can be seen that if this aspect of the algorithm's behavior is important enough for an algorithm visualization designer to deem as an important event, then it too should be graphically represented in accordance with guidelines presented in a cognitive processing model.

Researchers in cognitive science believe that the cognitive value of a graphical representation is not well understood. Some argue that a more solid foundation to determine the way people interact with graphical representations is needed [Scaife96]. Scaife and Rogers point out that research should account for the cognitive processing of these interactions, "that analyses the role played by external representations in relation to internal mental ones" [Scaife96]. They mention previous work that deemed advanced technology-driven graphics (e.g. virtual reality, animations, and multimedia) as better by intuition without proof. Without this evidence there is neither a way to make conclusions from the large number of studies conducted nor determine the value of these contemporary graphical representations.

Studies of this nature look to answer questions regarding how external representations and internal cognitive processing interact. Termed "external cognition" by Scaife and Rogers, they focused on the processing involved when interacting with graphical representations and the cognitive benefits of the various graphics, including static diagrams, animations and virtual reality [Scaife96].

The external cognition analysis of how viewers process different graphics describe three properties of forming a mental model: *computational offloading*, *re-representation* and *graphical*

constraining. Computational offloading refers to the differences external representations allow in external cognition. A particular representation can reduce the amount of cognitive effort in understanding. The second factor, *re-representation*, is the representing of the same abstract concept in different external graphical ways. Some graphical representations lead to more difficulty in problem-solving, while others make it easier, and still others can have equivalent effects. The last factor, *graphical constraining*, refers to the ability of graphics to enforce by restriction the type of interpretations that can be made. For example, charts and diagrams created in solving logic problems aid in finding impossible situations. This aids viewers in making inferences about the problem. These three factors, though similar sounding in definition, complement one another. Computational offloading refers to cognitive benefits, re-representation refers to structural properties, and constraining allows for computational offloading by restriction.

Scaife and Rogers point out that research on the cognitive processing of static graphics has been verified to apply to dynamic graphics to some degree [Scaife96]. When compared with sentential representations, i.e. written explanations, they note that graphics allow for more computational offloading and for information to be formulated less explicitly. Similar research finds that graphics support the role as external memory, and as an aid in limiting abstraction in the wrong direction [Larkin87, Scaife96, Zhang94]. However, none of the different representations can claim an advantage when it comes to making inferences.

2.3.2. Attention

Even if a graphical representation is chosen for its cognitive processing advantages, a designer must also consider the attentional properties of the graphic. There are two types of

forces that can attract attention according to Pashler *et al.* [Pashler01]. One force is bottom-up influences, which refer to reacting to some stimulus reflexively. For example, touching something hot will cause a person to quickly move their hand away from the item. Top-down reactions occur due to some particular intentional stimulus, *e.g.*, a person when fearful may jump at a small noise. Studies by Faraday and Sutcliffe provide evidence that varying techniques of the design can have effects on the viewer's fixation and attention [Faraday96]. Furthermore, they propose guidelines regarding attention including how to integrate motion, transition between events, pace the events, use and place labels, and incorporate audio.

In another study [Faraday98], Faraday and Sutcliffe created a tool for novice presentation designers that evaluated the presentation created by the designer. They discuss the guidelines for creating a presentation and processes of critiquing a presentation, embodied by the tool created for their research. The design issues mentioned include focusing the viewer's "thread of attention", timing effects to complement other features, and designing effects to make important information salient. The last item considers the various low-level properties of a graphical representation.

Cueing a graphical object focuses the viewer's visual attention to the object. Psychologists know and have shown that this can be done independent of eye movements [Sears00]. However, it is not so obvious what happens when multiple items may vie for attention. Research has explored the capacity and limitations of visual attention. Sears and Pylyshyn discuss the notion of visual indexing, which refers to providing "a means of setting attentional priorities when multiple stimuli compete for attention, as indexed objects can be accessed and attended before other objects in a visual field" [Sears00]. Visual indexing provides a means of quickly accessing the objects indexed without additional attentional scanning. In

terms of cueing, this means an object cue will not cause undue cognitive loading to follow the object or otherwise reference it. Cueing can focus the attention on the objects and then follow with other events that can be processed. As Sears and Pylyshyn [Sears00] point out, "to selectively attend to a non-indexed object, its position must first be ascertained through attentional scanning." Thus, a visually indexed object can be attended to more rapidly. Additionally, visual indexes are not tied to a position in the visual field of view. Indexes are to objects or features, not to locations, and objects are continually referenced even as they move. A limitation to the visual index is its small capacity to hold more than 4 to 5 objects. However, the visual index can track each object independently and in parallel with other indexed objects. Finally, through experimentation, Sears and Pylyshyn found that objects that are indexed confer a priority in attention and accessibility.

2.3.3 Empirical Studies with Psychological Considerations

Some research studying the pedagogical value of visualizations provided a thoughtfully designed visualization, which gave consideration to the cognitive and attentional factors of viewer comprehension [Holmquist00, Narayanan02, Tudoreanu03, Tudoreanu02, Hansen02].

Holmquist and Narayanan [Holmquist00] created tools to create visualizations derived from a theoretical cognitive process model. This model of comprehension is based on stages that build a mental model of the information being presented. This affected the design of the system more than the design of the visualizations presented. The authoring tool, Hypermedia Authoring Support System (HASS), creates Hypermedia Education Manuals (HEM) visualizations. While the HEMs are viewed by users, their interactions are recorded. An evaluation tool, Hypermedia Evaluation System (HES), then can be used to analyze the data and suggests visualization design

improvements. Empirical evidence confirmed that students using the system with the improved design significantly outperformed those with a system without better information flow and navigational design.

Later, Narayanan and Hegarty [Narayanan02] conducted a few experiments to test the use of algorithm animations using a cognitive process model-based animation versus a "conventional" animation. Their cognitively designed animations differed from the conventional animation by having several levels of presentations allowing for comprehension to occur in stages. Results from the studies indicated that students with the cognitively designed material significantly outperformed students with traditional algorithm material.

Tudoreanu [Tudoreanu03] found that visualizations that employed factors such as the presence of legends and navigation buttons that reduced cognitive effort were more effective than visualizations that lacked such devices. In a comparison of perfomance between two different visualizations of the same algorithm, two graphical representations of a parent-child relationship were employed in the studies. One used arrows while the other used different spatial placement in a 3-dimensional view to represent the relationship. The visualization with arrows had a significantly higher performance rate. However, this was not the primary focus of the experiment and it was not the only difference between the two animations. Thus any conclusions would have to be substantiated with further studies.

Hansen *et al*. [Hansen02] created a system, HalVis, that supports animations that occur in stages allowing for the viewer to progressively learn about the underlying algorithm. It also used "probes and questions that stimulate thinking" while the animation was running. One of the experiments involved comparing performance using the HalVis system versus viewing another

visualization system. Overall, the results showed the students using the hypermedia visualization outperformed students using other teaching methods.

Experiments conducted by Tudoreanu *et al*. [Tudoreanu02] used a testing environment and visualizations designed to reduce the cognitive load. The testing environment was designed to aid viewers by adding interactive print statements and automatically positioning and sizing windows on the monitor. The visualization design contained legends. They concluded that viewers of these visualizations performed significantly better than the participants who did not view a visualization.

2.4 Summary

From these areas of research, we begin to understand the impact of viewer comprehension has on good animation design. We seek to investigate how the design of visualizations can be enhanced, beginning with empirical studies as outlined by Gurka and Citrin. In the past, findings across studies were inconsistent and findings were not duplicated since the parameters of the studies varied widely. The SSEA system, described in Chapter 3, will allow our studies, examining the effects of attributes in algorithm animations on viewer comprehension, to remain consistent. This is done by providing a uniform environment and having participants perform the same task across these studies.

Chapter 3

SSEA System Description

The System to Study Effectiveness of Animations (SSEA) application allows researchers to investigate various characteristics of algorithm visualizations in a single environment. The system allows subjects to view one of a variety of animations while answering questions to determine their understanding of the underlying algorithm. There are three main users of SSEA: the viewer, the animator, and the experiment designer (referred to as the experimenter or researcher). The animator creates the algorithm visualization to be viewed based on the researchers' design and goals. The animator and researcher may be the same individual or, at the least, will be working closely together.

3.1 System Design Considerations

Many factors contributed to SSEA's design, some based on our previous work and others determined by our future needs. First to be considered was the visualization component itself, and determining whether one of many existing algorithm visualization systems could be used for our studies. In our related research we are running ongoing experiments using the VizEval suite [Rhodes04, Ross04]. Similar to SSEA, VizEval is a framework designed for researchers to design, create, deploy, and organize resulting data for experiments designed to study cognitive, perceptual and attentional features of visualization. The animation module used in the VizEval suite is called the Support Kit for Animation (SKA) [HTaylor02]. However, VizEval evaluates the effects of features in a short animation that is not based on an underlying algorithm. The features we have tested in the VizEval experiments included determining the effects of cueing on

a viewer's ability to detect property changes of objects and whether viewers could correctly identify multiple object changes. This required all subjects to view a very short animation, lasting less than one second, followed by a set of questions. Subjects were required to watch a series of these animations and questions during a single sitting.

The SSEA studies differ from these in that they focus on comprehension of a visualized algorithm rather than perception of individual elements of a visualization. User interactions along the lines of traditional algorithm visualization systems were thus required for SSEA. Therefore a new environment was created using the same animation module, SKA, which provided the graphics and animation support. The SSEA environment provides the ability to easily create graphical representations of the specific features we wished to investigate.

Another design consideration for SSEA was the intended deployment and target operating systems. Our studies would be conducted locally on students at the University of Georgia campus. Therefore, a stand-alone system was feasible to be installed in campus computer labs. For the sake of portability for future tests, the system was implemented in platform-independent Java. A final factor was that the task of creating new visualizations for further studies should require only minimal effort from researchers.

3.2 Implementation

SSEA integrates a visualization interface with a question panel, pop-up questions and a monitor. The monitor automates data collection of user interactions and events. The architecture for SSEA follows the Model-View-Controller design pattern, separating data from views of the data. Here data consists of project and animation information. Figure 3.1 depicts the overall architecture of SSEA with the various components. Two main modules control SSEA, a

controller and SKA. The controller manages the allocation of data, the piping of user interactions to appropriate components, and the message passing between components. SKA manages the display of the animation and adjusts the visualization to any user interactions.

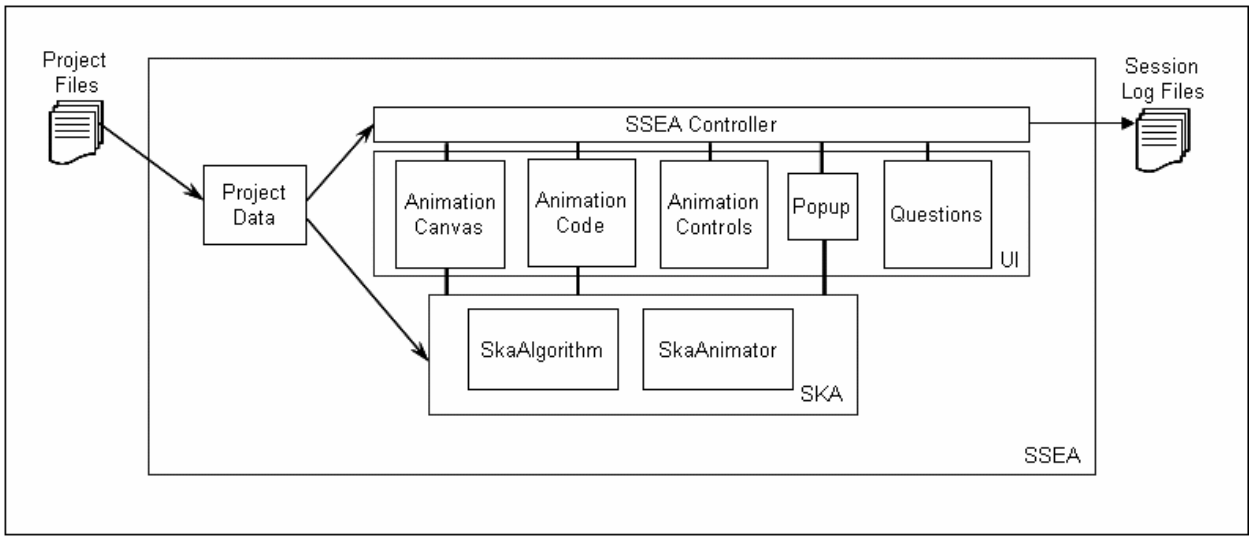

Figure 3.1. SSEA architecture.

3.2.1 User Interface

Separate user interface modules exist for each display area in SSEA. These areas, seen in Figure 3.2, include an animation area (A), a pseudocode display (B), animation controls (C), and a question area (D). The animation area and code area display the graphical representation of the underlying algorithm, and are synchronized by SKA. A high resolution monitor is required to view the multiple areas of SSEA in their entirety. Below we discuss details of the interface; many of these features are listed in a report by Naps, *et al.* [Naps03] as good design items for visualization systems.

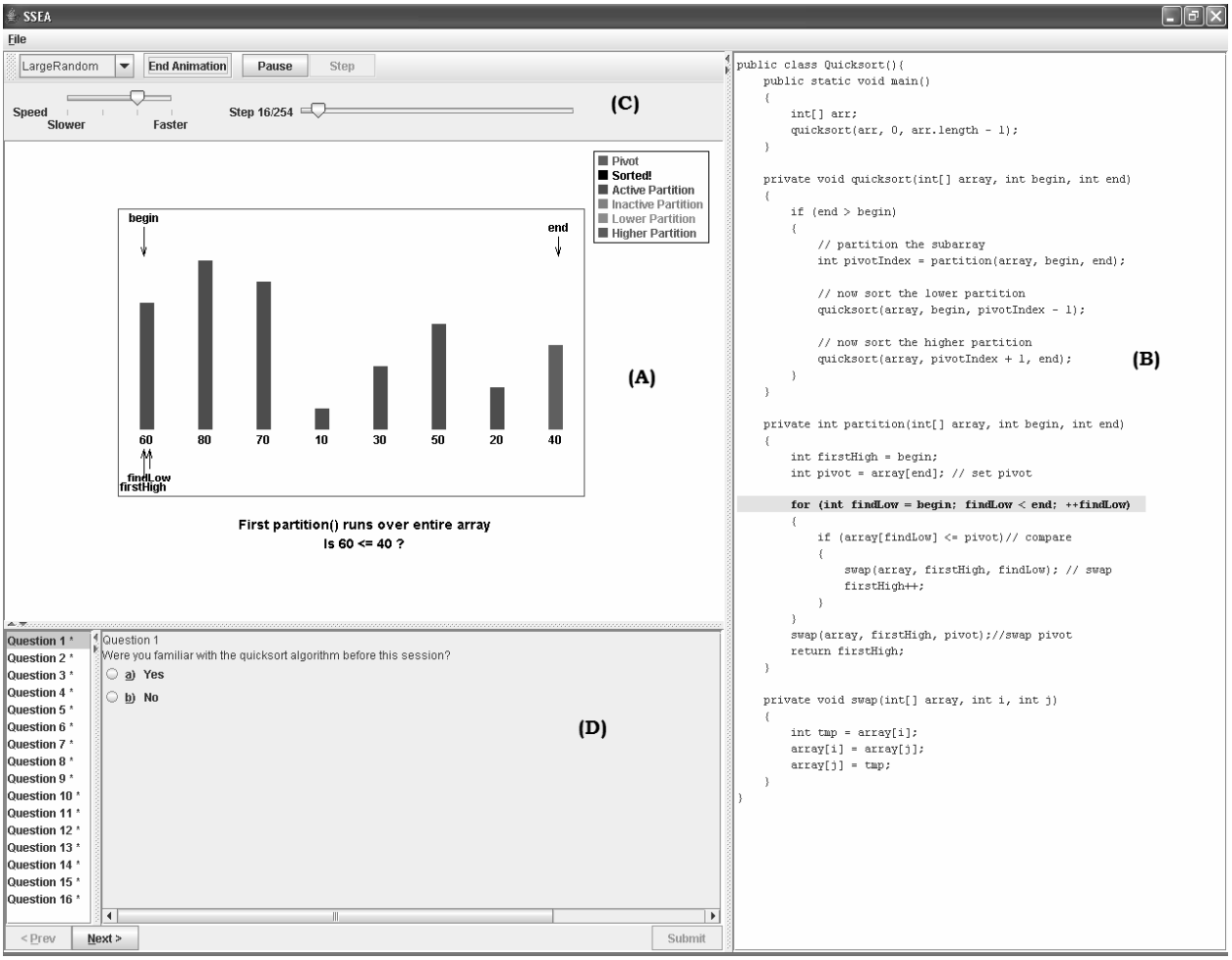

Figure 3.2. SSEA screen shot. (A) Animation area where visualization of underlying algorithm is displayed. **(B)** Pseudo code display: highlights code being executed by underlying algorithm, which is synchronized with animation area. **(C)** Animation controls: viewer controls playback of algorithm animation. **(D)** Question listing: viewer responds to questions designed by the researchers to evaluate the viewer's comprehension of the algorithm.

The playback of animations can be controlled by the viewer via the animation control area (Figure 3.2C). One feature allows the user to select from a collection of data sets as the input for the algorithm. Another control sets the speed of the animation. The animation can be paused, ended, and then begun again from the start. Stepping through the animation, which can only occur if the animation is paused, causes the next step of the animation to execute and then the animation is paused again. A slider indicates the progress of the animation. Moving the slider to the left will allow users to select a point at which to restart the animation. The state of

the animation is tracked by the controller. Figure 3.3 is a state diagram of the control of the animation playback.

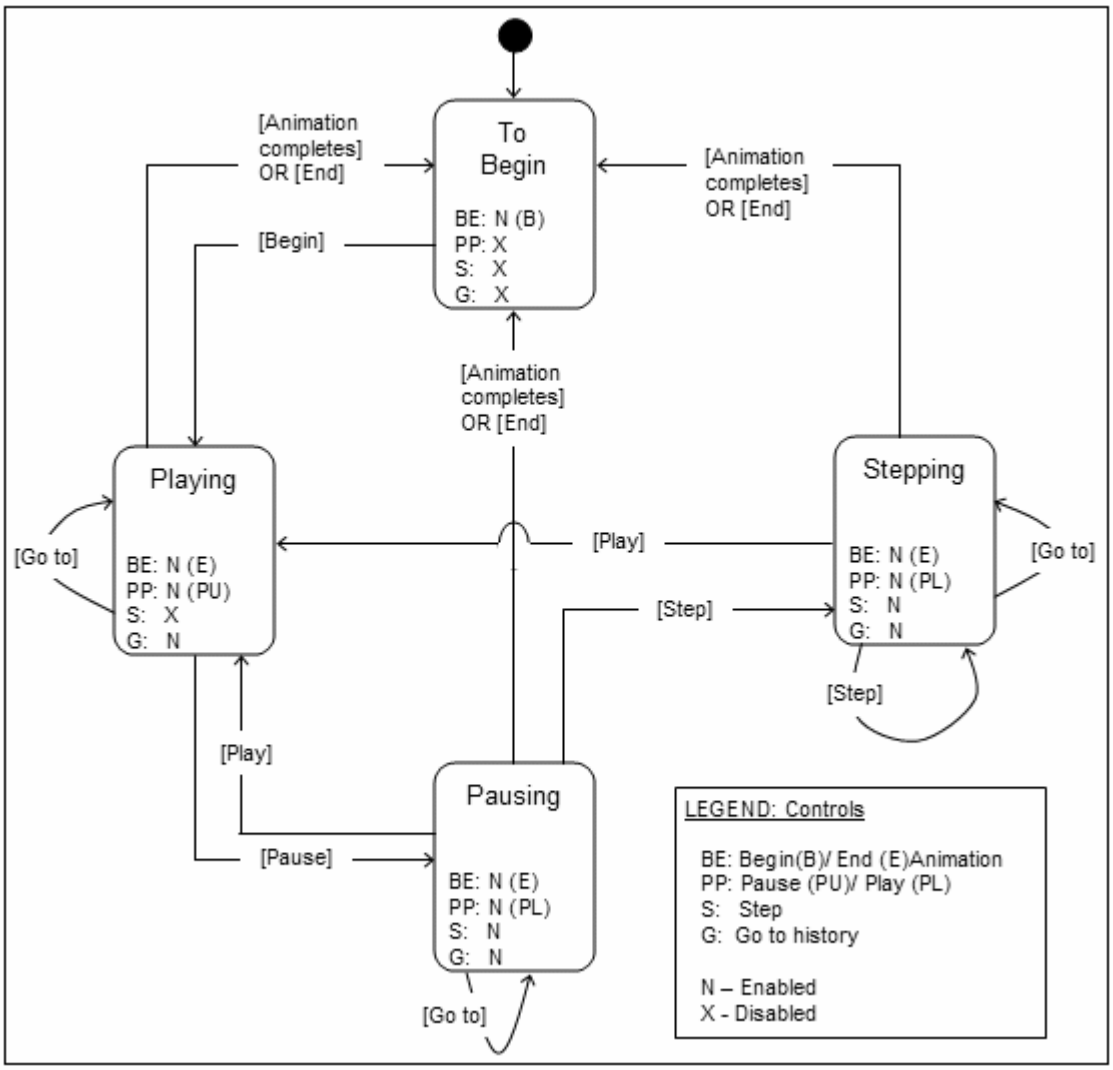

Figure 3.3. Animation state diagram. Based on user interactions with the animation controls, various states of animation will be entered. The states of animation are starting, playing, stepping, and pausing. Possible interactions include begin or end animation, pause or play animation, step animation, or go to a previous step in the history of the animation.

While watching the animation, the viewer may see a question pop up. The animation pauses its run until an answer is provided. These "popup" questions are a design feature available to the experimenter and implemented by the animator. These questions can be

associated with the animation of the algorithm on a particular data set. The popup can appear only during the initial run of the animation of the algorithm with the associated data set. To prevent users from using the code to supply the correct answer to the popup questions, the popup window is positioned over the pseudo code area and is not movable. Optional to the researcher to display after the user submits an answer to the question is the correct answer.

Below the animation area are the questions designed by the researchers to evaluate the viewers' comprehension of the algorithm. These "traditional" questions are displayed individually, with a listing of all questions off to the left side. An asterisk next to a question number in the list indicates that the question is still unanswered. Users may return to a question and change their response. When all questions are answered, a "submit" button becomes enabled. When the user clicks the submit button the session terminates.

Optional to the researcher is a questionnaire that can be displayed initially, requiring viewers to answer all questions before proceeding. Figure 3.4 depicts a sample questionnaire; all responses are recorded in the session log file.

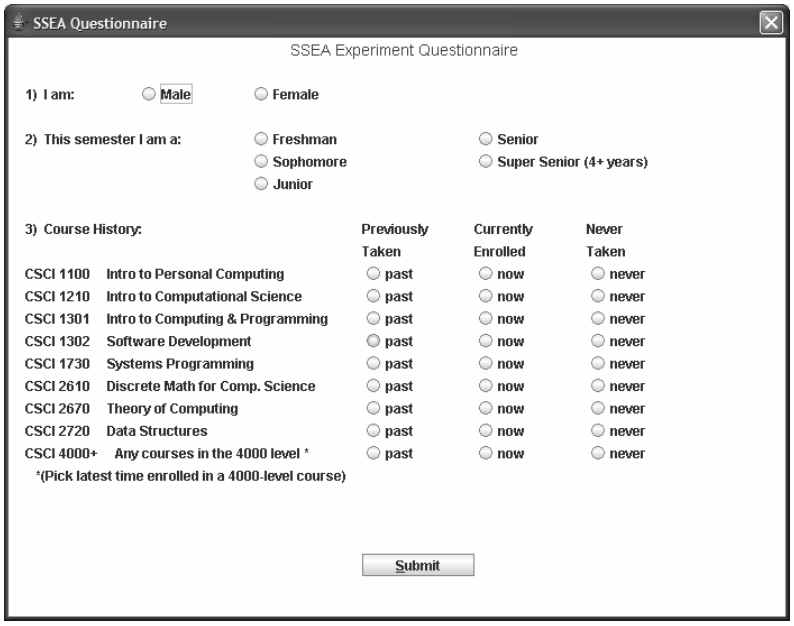

Figure 3.4. SSEA screen shot of questionnaire.

3.2.2 Visualization Module

The graphical visualization generated by the SKA module results from an algorithm and an animator working somewhat independently. A threaded architecture with a buffer is used, following the Producer-Consumer design pattern. An algorithm thread (an instance of the SkaAlgorithm class) is the producer of shared data, while the animator thread (an instance of the SkaAnimator class) consumes the data. The shared buffer (an instance of the ActionBuffer class) contains the information needed to produce the algorithm visualization. The algorithm thread is created when a new input set is selected. A new animator thread is created for each new run of the visualization.

The graphical representations consist of graphical objects and actions on one or more of these objects. Graphical objects consist of lines, rectangles, text labels, circles and composite graphics. Each object has numerous properties that can include color, fill, visibility, font, position, and labels. The canvas references a list of graphics. As graphics are updated by the animator module, the canvas repaints the graphics causing an animation. Refer to Figure 3.5 for an overview of the animation engine architecture.

The visualization programmer creates a visualization by defining a subclass of the algorithm SKA class, SkaAlgorithm, which will run the underlying algorithm being visualized. The code is annotated with additional calls that form the animation. When the algorithm thread runs, it produces a list of AnimObjects, animation objects, that is shared with the animation module. There are two types of animation objects, as depicted in Figure 3.6B, actions and markers.

AnimActions represents actions on graphics, which include but are not limited to: moving an object, changing an object's property, hiding or showing an object, and changing the

object's textual label. Specialized actions are available to add new graphics not previously present on the display. Actions can also be combined into a "bigger" action. For example, a series of actions that hide, show, and then hide an object will create a flashing effect. The execution of an action constitutes a step in the running animation.

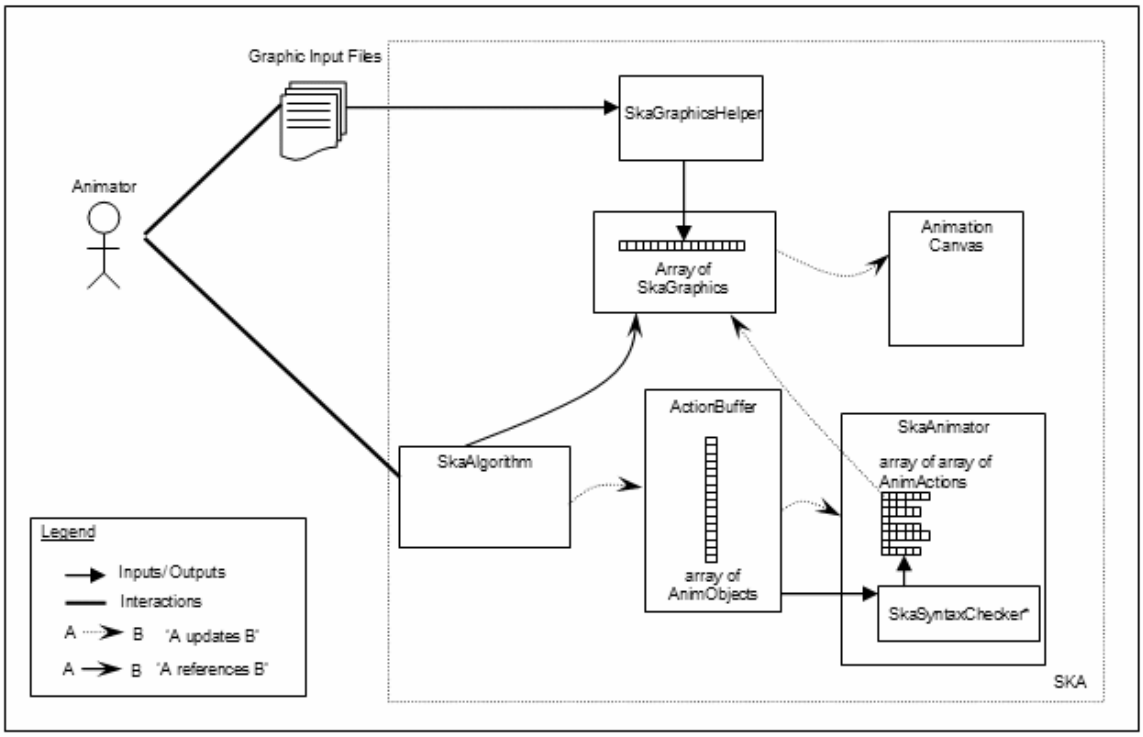

Figure 3.5. Architecture of SKA module. The architecture follows a Producer-Consumer design. An instance of the SkaAlgorithm class is the producer of a sequential list of AnimObjects that is stored in the buffer, ActionBuffer. The list is created by the programmer's subclass implementation of SkaAlgorithm. The consumer, an instance of SkaAnimator, waits for the buffer to be sufficiently full of data before it transforms the sequential list into a list of lists, a two-dimensional array. A two-dimensional array is created since each step of the animation may require more than one action to occur simultaneously. In other words, an array of steps is created where each step may consist of an array of actions. The conversion of the one-dimensional array into a two-dimensional one occurs via a special helper class, SkaSyntaxChecker. If there are any errors in conversion, the helper class will flag an error to the programmer in the design mode of SSEA.

AnimMarkers indicate how the actions are to occur relative to one another. For example, actions can be run in parallel, or they can be grouped so that the next step of the animation will

occur only after all the actions in the group are complete. Additionally, markers indicate to the controller which component should being updated with a new action. Special markers cause a popup question to appear during the animations or highlight a line in the pseudocode display. The programmer can use the markers to ensure that the relative position of the actions will not compromise the intended visualization.

The animator module, SkaAnimator, the consumer, waits for the buffer to be sufficiently full of data before beginning to produce the animation. SkaAnimator will replay all the actions step by step during a call to go back to a previous step. To accomplish this, each graphic has a history of all its properties (see Figure 3.6C). Thus the graphics can be redrawn as they were at the previous step and the actions will act on them as they had in the original playback. This supports the ability of future versions of SSEA to permit user driven visualizations, where interactions are specified by the animator. For example, the visualization could prompt the user to select an object as the next step in the algorithm. Based on the user response to these prompts, the resulting new visualization would be shown.

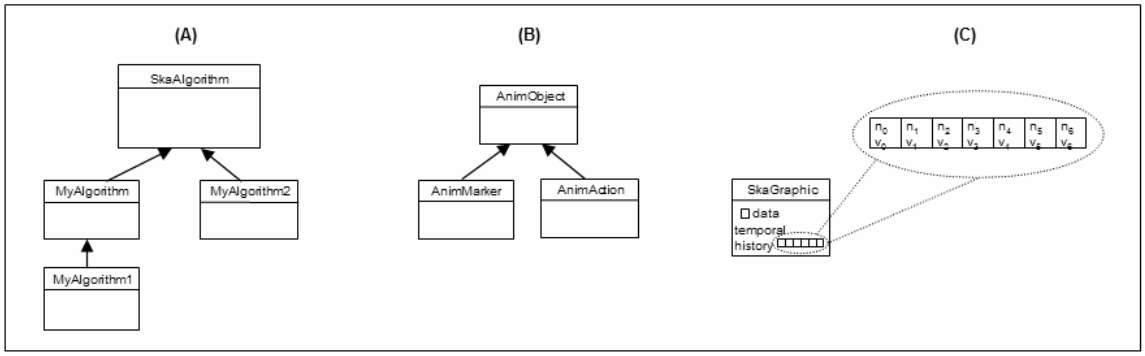

Figure 3.6 Animation Engine Details. (A) Possible subclasses of SkaAlgorithm that a programmer can create. MyAlgorithm1 is an example of a way to create varied animations for the same underlying algorithm. **(B)** Architecture of creating animation using actions and markers. Markers assist the SkaSyntaxChecker to create the 2D array of actions. Actions change graphics, creating a new scene in the animation. **(C)** Temporal properties in all graphic objects allow for going back in history. A history is kept for each property of a graphic. Here n_i represents the step where new value, v_i ,became effective.

Finally, the list of animation objects described above is sequential. This is done for the programmer's sake, enabling changes and debugging to be done with relative ease. The programmer can run SSEA in a design mode that allows him to check for any problems with the visualization. Determining where errors are located would be a complex and difficult task if it were the programmer's responsibility to create a two-dimensional array of animation objects. Instead, SkaAnimator uses a special helper class, SkaSyntaxChecker, to accomplish the transformation and error checking. A two-dimensional array is created since each step of the animation may require more than one action to occur simultaneously. In other words, an array of steps is created where each step may consist of an array of actions.

3.3 Specifying an Empirical Study

In the process of specifying an empirical study, the researcher first designs an experiment with some aspect of algorithm animation in mind. The researcher creates a specification for the visualization that includes the graphics of the visualization, the desired input sets, and the algorithm(s). The researcher should also determine the substance of pre-test questions, popup questions and traditional questions. At the conclusion of running all experiments, the researcher will be able to examine the log files for each session conducted.

Log files contain all viewer interactions with SSEA. For each session, the log records the timings and type of user interactions to control the animation, responses to questions including any selections made that differ from the final choice, and all responses to popup and regular questions. For an example of a hypothetical log file, see Appendix B.

Once the above requirements have been determined, the programmer then takes the specifications from the researcher and creates the corresponding project files to run the

experiment. These include the list of questions, the list of popup questions, files to be used as the input data sets (called graphic files), annotated pseudocode to be displayed and highlighted during the animation, and a path for storing each session's log file. Sample entries for the project file, the project question file, the project popup question file, and the input sets can be found in Appendix A. We can see how these project files fit into the overall SSEA architecture in Figure 3.1.

The programmer will specify graphic files and create algorithm subclasses to form the visualizations in SSEA. The graphic files form the different input sets available to the viewers. Graphic files are text files with each line representing a graphic object that will be displayed on the canvas at the start of the animation. Figure A.5 in Appendix A lists the tags needed to create these objects.

The programmer can create a single algorithm by subclassing the algorithm module, SkaAlgorithm to create a class which we will call MyAlgorithm. MyAlgorithm will contain the underlying code of the algorithm to be animated, with important events calling predefined or customized animation methods. The programmer then can create additional algorithms by either subclassing SkaAlgorithm again or by subclassing his own MyAlgorithm (see Figure 3.6A). The latter case is beneficial when the underlying algorithm is the same, but the researcher wishes to change the animations. This facilitates creating varied graphical representations for the same event in an algorithm, by overriding a customized animation method in MyAlgorithm.

Chapter 4

Experiment Materials and Methods

In this chapter, we describe an empirical study that evaluates two visualization design attributes. This is the first in a series of studies to be conducted examining the effects of attributes in the comprehension of program visualizations. By conducting empirical studies we can determine which attributes are more effective in communicating information about a program. The experiment tests how cueing of comparisons and techniques for illustrating the exchange of objects affect the viewer's comprehension of the underlying algorithm.

The empirical study we conducted used an algorithm animation framework called SSEA, a System to Study Effectiveness of Animations. Our participants were undergraduates in computer science classes, mostly computer science majors. Participants viewed one of four animations of the quicksort algorithm in SSEA, and answered questions about the algorithm. The four animations differed only in the graphical representations of the comparison events and of the swapping of two values.

4.1 Experiment Description

The hypotheses being tested are:

1. The performance rate, based on the number of correct answers for a given set of questions, is significantly higher for participants viewing algorithm animations that use flashing as a means to cue a comparison event than for participants viewing animations in which bars are compared without such cueing.

2. The performance rate, based on the number of correct answers for a given set of questions, is significantly higher for participants viewing visualizations of sorting algorithms in which objects are seen to move to their new locations when swapped than for participants viewing visualizations of sorting algorithms in which the heights of the objects being swapped grow or shrink in place.

We are examining these hypotheses on a fixed set of conditions. The data set size is small, and a single view of the animation and pseudocode is presented to viewers.

The experiment was conducted on a voluntary group of undergraduate computer science students, registered at the University of Georgia in the fall semester of 2005 and spring semester 2006. Students were recruited from various computer science classes during these semesters. Participants received a monetary payment of five-dollars in return for their participation.

The participants were randomly assigned to a group corresponding to the version of the algorithm animation they would view. Figure 4.1 depicts the four possible groups. A total of 59 students participated. Each participant was allowed as much time as they needed to complete the study at their own pace, but most finished in about an hour. Studies were conducted on Dell Dimension desktop computers with high-resolution 17-inch LCD flat-panel color monitors.

		Exchange Technique	
		Move	Grow
Cue Presence	Yes	MC	GC
	No	MX	GX

Figure 4.1. 2x2 Factorial Design for Experiment. Letters indicate the algorithm animation code: (M) refers to an exchange technique where bars move to new locations; (G) refers to an exchange technique where simultaneously bars grow and shrinking to new heights; (C) represents comparison events that are cued; (X) refers to when no cueing occurs for comparison events.

An instruction sheet led each participant step-by-step through the study. Each participant initially used the SSEA system to view a simple algorithm that finds the maximum value of a data set. This allowed them to learn how to interact with the animations using SSEA. The maximum value animations, which included two data input sets, were the same for all participants. Additionally, one popup question occurred in the animation in order to prepare students for these kinds of questions in the quicksort animation. The instructions led each participant to use each one of the animation controls. The instructions also asked participants to answer and submit the set of four traditional questions. In addition, a sheet explaining the animation controls of SSEA was attached. Participants were required to complete these steps before continuing on to the main section of the experiment.

Once the training portion of the experiment was completed, the participant was instructed to start the quicksort SSEA program. Participants were unaware that others were viewing different animations. The first step in the quicksort portion required participants to complete a questionnaire form about their gender, year in college, and the computer science courses they had completed or were currently taking. See Appendix E for a listing of questions posed to participants in the questionnaire.

Participants then viewed the quicksort animation for the group to which they had been randomly assigned and answered questions. The questions (listed in Appendix C) were the same for all groups and were either multiple choice or true-false. The instructions for the quicksort visualizations directed students to view all of the ‘LargeRandom’ input set. This ensured that participants would be asked all eight popup questions. A list of the popup questions can be found in Appendix D.

When participants completed the quicksort visualization questions, they were given the opportunity to comment on the animations they viewed, the SSEA system, or give any general feedback through a paper survey form. After feedback forms were collected, students received payment for their participation.

4.2 Quicksort Visualization

The quicksort algorithm implemented selected the right most value in the active partition as the pivot, and did not include any performance improvements that have been seen in other versions of the algorithm.

A screenshot of the beginning of the quicksort visualization can be seen in Figure 4.2. In the animation, the input data is represented as filled rectangular bars, each bar being an element of an array. The value of the element is indicated by both its height and a label appearing below the bar. Labeled arrows are used to symbolize the four index variables of the array in the algorithm, pointing to the respective bar in the array.

The quicksort algorithm is recursive and involves dividing and sub-dividing the array into partitions. An outlining rectangle indicates the current partition, enclosing the bars in the partition. In addition, labeled arrows marked “begin” and “end” indicate the boundaries of the current partition. Two additional labeled arrows marked “firstHigh” and “findLow” represent the variables that move within the current partition looking for values that are higher or lower than the pivot.

Bars are colored to indicate the state of the algorithm. A legend is provided in the upper right-hand corner. The current pivot is colored green. Inactive partitions are colored gray. The active partition is initially colored red. As the algorithm proceeds the bars are sorted into a lower

partition (values less than or equal to the pivot) which is colored light blue, and a higher partition (values greater than the pivot) which is colored purple. The pivot value is then swapped with the rightmost value in the lower partition, which is its final location. It is then colored black to indicate that is in its sorted position.

Two lines of text below the array serve as captions, providing a description of the current step being performed. Lines in the pseudocode display are highlighted as the animation is executing the corresponding event.

Animations of two events are varied in the four cases that we are studying, a comparison event and an exchange event. In two of the cases (MC and GC) labeled “cueing”, the comparison of the element at the “findLow” arrow with the pivot element is cued. That is, the compared bars flash three times when the elements are compared. In the other two cases (MX and CX) labeled “no-cueing”, the bars are not flashed. In two of the four cases (GC and GX) labeled “grow”, we study the exchange of values that is depicted through in-place changes of bar heights; one bar appears to become taller while the other bar in the exchange simultaneously becomes smaller. In the other two cases (MC and MX) labeled “move”, the exchanged bars are seen moving in an arcing fashion from their original positions to their new positions.

There were four predefined input data sets students could choose for the quicksort algorithm to animate. The default was a set of 8 randomly ordered values. Additionally, a smaller set of random values, having a size of 5, was available. Finally, eight ascending or descending values were also offered for participants to view.

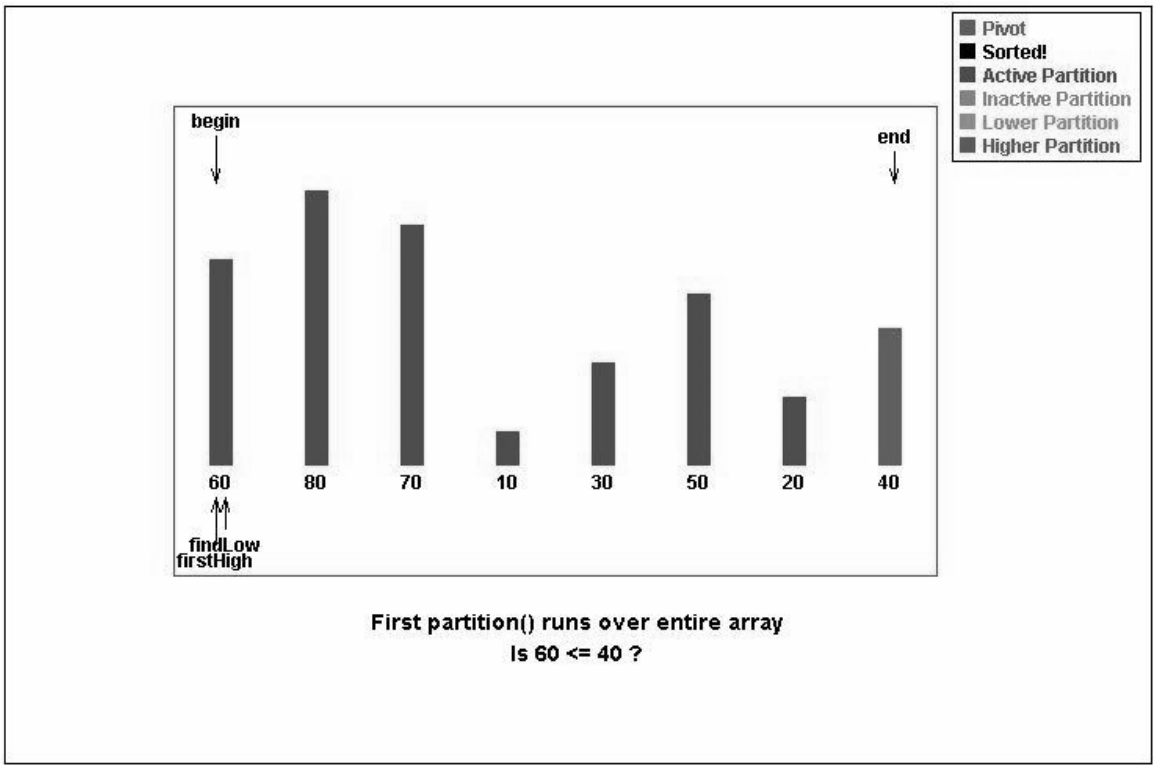

Figure 4.2. Screenshot of quicksort visualization.

Many low-level features of the animations are based on studies done in a collaborative effort between the Davis group at Georgia Tech and the Kraemer group at University of Georgia [HFES]. In a pilot study the Davis group determined that a height change of 22 pixels was the "just noticeable difference" for bars. Therefore, the display height of each bar was made proportional to its value by a factor of 22. Additionally, the number of elements in the input arrays, either 5 or 8, is within the capacity limits of short-term memory (7±2) [Sperling86].

4.3 Questions

The participants' responses to the "traditional" questions measure their comprehension of the underlying algorithm. The problems presented to each student were classified as a level from Bloom's taxonomy [Bloom56], reflecting a range of concepts. The questions are classified as knowledge, comprehension, application or analysis level. These four levels are the lower levels

of a six-level classification. Knowledge level demonstrates the ability to recall facts, terminology, or other previously learned information. Comprehension level demonstrates an understanding of main ideas. Application level demonstrates the ability to solve new problems from previously acquired knowledge. Analysis level demonstrates an ability to break down knowledge to make generalizations and inferences. By classifying the questions in this taxonomy, we can examine the results with regard to different levels of expertise of the participants.

Each question was designed to have only one answer, using multiple-choice or true-false formats. The majority of questions, sixteen, were presented to students so they could answer them with the use of the visualization. Three of theses questions can be classified at the application level, requiring students to use the information they just learned and apply it to make calculations on a different data set. Four questions at the knowledge level can be easily answered from the pseudocode. Six comprehension level questions need further understanding of how the algorithm works. Two questions require "the ability to break a complex problem into parts" [Howard96] of the analysis level. Finally, the first question simply asked for prior knowledge of quicksort. A list of the questions can be found in Appendix C. These questions could be answered in any order and at any time during the experiment, not simply as a post-test. The visualizations were available for students to use as a reference while answering these sixteen traditional questions. Both the intermediate choices and final answers for all questions are recorded in the log.

There were eight popup questions the students could answer. At a specific event during the animation, a question regarding that particular event would pop up. Questions were placed in a modal window, requiring a response before the student could proceed. Once a question was

answered it would not appear again if the viewer would replay the animation for the same input set.

In contrast to the traditional questions, the responses to the popup question address the participants' perception of the animation. These questions are used to solicit answers about what the user just saw, categorized as either cueing-specific ("which two bars were just compared?") or exchange-specific ("which two bars just exchanged values?"). All responses to the popup questions are recorded in the session log.

Chapter 5

Experiment Results and Discussion

During the fall semester 2005 and spring semester 2006 at the University of Georgia, 59 computer science undergraduate students participated in our study. A random selection of one of four quicksort visualizations, where objects either were: moved when a swap occurred and comparisons cued (MC), moved for a swap and comparisons were not cued (MX), objects changed heights growing and shrinking to their new values when a swap occurred while comparisons were cued (GC), or objects changed height without comparisons cued (GX). There were 14 students in the MC group, 12 students in the MX group, 16 in the GC group, and 17 in the GX group. A summary of results can be found in Appendix F.

Most of the participants were male, with only 8 females participating. The majority of students were juniors (42%), followed by seniors (28%), sophomores (22%), and 8% were freshman. The quicksort algorithm is formally taught in the CSCI 2720 Data Structures. None of the participants had completed the course prior to taking part in the study. For students currently enrolled in the class, the instructor for the Fall 2005 course agreed to hold off the quicksort lecture until after the experiments were concluded. Students enrolled in the Spring 2006 Data Structures course partook in the experiment on their first day of class, thus ensuring no formal lecture. Despite these careful measures, a surprising majority of students claimed to be familiar with the algorithm with 40 of the 59 participant responding ‘yes’ to question 1. A listing of the questions can be found in Appendix C.

Feedback from participants can be summarized as two general remarks. First, many commented that the overall experience of using the visualization was positive, describing the

study as enjoyable. The second common remark indicated that more of an introduction to the algorithm be given.

5.1 Traditional Questions

Participants overall performed reasonably well on the traditional questions, some of which required detailed knowledge of the procedure or time complexity analysis, with an average score of 60.74%. Average time to complete the study was 22.9 minutes. Table 5.1 shows the per-animation-group averages for score and time.

Group	Average Score (%)	Average Time (minutes)
MC	61.57	26.00
GX	60.41	21.24
MX	59.08	25.25
GC	61.63	20.25

Table 5.1 Average Time and Score per Group, Traditional Questions

An ANOVA (Analysis of Variance) analysis was performed. No statistically significant effects were found for performance on overall traditional questions for cueing, exchange type, or interaction effects.

The traditional questions were further sub-divided into groups based on the type of knowledge the question tested: Knowledge, Comprehension, and Application. See the classification labels on the question listing in Appendix A. ANOVA analyses were performed on these subsets. Again, no significant difference was found among the four animation groups.

5.2 Popup Questions

Overall performance on the popup questions was 66.22%. Table 5.2 shows the per-animation group scores for the popup questions. Times are as reported in the traditional question section.

Group	Average Score (%)
MC	72.86
GX	58.41
MX	62.33
GC	71.63

Table 5.2. Average scores per group on popup questions.

An ANOVA analysis was performed. Cueing was found to have a statistically significant effect $F(1,57) = 4.44$, $p < 0.04$ on participant performance on popup questions. Further, cueing was also found to have a statistically significant in a subset of the popup questions classified as cueing-specific (questions 1, 2, 3, and 8), $F(1,57) = 10.39$, $p<0.002$.

Exchange type was not found to have a statistically significant effect on performance on the overall set of popup questions. However, in a subset of questions classified as exchange-specific (questions 4-7), a statistically significant benefit to "move" over "grow" was found, $F(1,57)=5.74$, $p< 0.02$.

5.3 Correlations between Popup and Traditional Questions

The correlation co-efficient between performance on popup questions was calculated for the overall group (0.466). Figure 5.3 plots both individual popup scores against traditional scores (diamonds) and popup score against mean traditional score for those with that popup score value (squares).

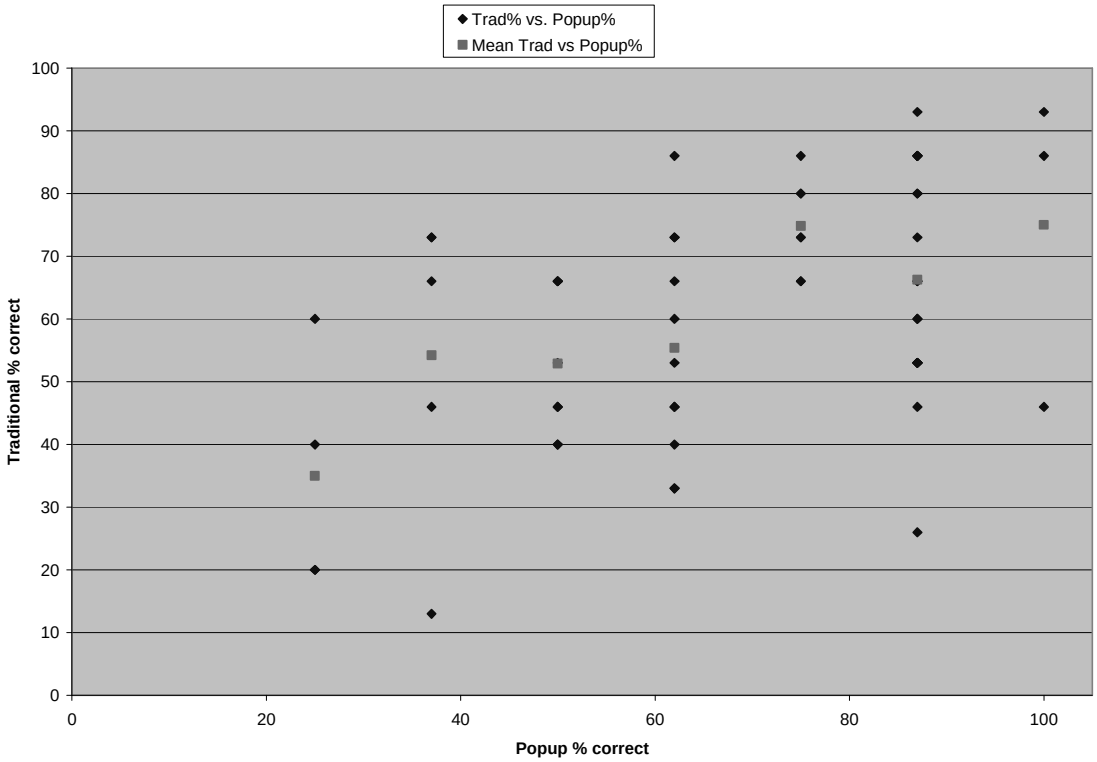

Figure 5.3 Correlation between Popup and Traditional Scores

5.4 Discussion

The lack of a significant effect on viewer comprehension for the type of flash cueing evaluated in this study may, at first, seem a surprising result. However, a viewing of the animation (or a review of the animation description in Chapter 4) reveals that the animation employs several types of cueing to indicate that two bars are being compared. In particular, color, labeled arrows, and location within the current partition all also cue the identity of the bars to be compared. Thus, we do not conclude that such flash cueing is not valuable in promoting comprehension of animations. Rather, we conclude only that the use of flash cueing as a redundant cue in this animation did not significantly benefit comprehension.

It is interesting to note that flash cueing was found to have a statistically significant benefit both in overall performance on the popup questions and especially on performance on the cueing-specific subset of the questions. The popup questions focused on perception and recall of animation events that had just occurred. The cueing-specific questions took the form of "Which two bars were just compared?". Thus, we can conclude that flash cueing does help the viewer to better note the low-level behavior of the animation. Whether this low-level benefit carries over to the higher-level comprehension of the depicted algorithm appears to depend on the presence of other, redundant cues. In the presence of multiple other cues, as in our study, the flash cueing was not shown to be of significant benefit to comprehension of the algorithm. In the case that flash cueing were the *only* cue that the values of two bars are about to be exchanged, we speculate that such cueing would quite likely have a much greater impact.

Again in a similar fashion to cueing, the type of animation used to depict an exchange was not found to have an effect on performance on the traditional questions. The lack of a significant effect of exchange type on viewer comprehension of the depicted algorithm, while less surprising, may have a similar explanation to that of cueing: the exchange of bars is cued redundantly. Color, labeled arrows, and position within the current partition all serve to indicate the identity of the bars that have been exchanged.

Exchange type did not have a significant effect on overall performance on the popup questions. However, on the exchange-specific subset of the popup questions a significant benefit was found for the "move" animation versus the "grow" animation. These questions took the form "Which two bars were just exchanged?". We controlled the time aspect of this portion of the animation to ensure that the "grow" and "move" animations required the same amount of time, and can thus eliminate differences in time-on-screen as a possible explanation for this

effect. Perhaps a more likely explanation is that while the bars remain in place in the "grow" animation, they move across the screen in the "move" animation. If the user's gaze has fixated on a portion of the display that does not contain the bars that are exchanging values, then the "move" animation has the potential to move the bars across the user's current area of focus, while the "grow" motion does not. Further, the greater overall motion associated with "move" animation has the ability to attract the user's attention, even in the periphery of the user's view [Bartram01]. Further studies in which the distance from the user's current focus and between the exchanged bars is varied could be performed to help sort out the components of this effect.

The moderate strength of the correlation between performance on the popup questions and performance on the traditional questions suggests that the presence of such popup questions may help to focus the user's attention on the details of the algorithm, and help the user to form a better understanding of the depicted algorithm. Another possible explanation for the correlation is that test subjects who perform well on one type of question are "good students" who are likely to perform well on other types of questions. We are investigating this question in a between-subjects study, in which we compare groups who are presented with popup questions against those who do not see popup questions. In addition, we study the effects of the presence/absence of feedback (providing the correct answer to the popup after the user has submitted their answer) and the effect of "predictive" questions (What is about to happen?) versus "reactive" questions (What just happened?). In this work we must also consider the possibility that over-attention to low-level detail may prevent the user from "stepping back" to gain a higher-level, conceptual view of the algorithm's behavior.

Chapter 6

Conclusion

We have focused on creating a framework for examining the effects of various attributes that can be incorporated into visualizations. The SSEA system enabled us to begin empirical studies on various graphical representations of algorithm events. Our experiments tested performance on cueing and using different methods for exchanging objects in a quicksort animation.

In the context of low-level, perceptual considerations, flashing is an effective technique for cueing the comparison of values in algorithm animations of the type studied in this thesis. Similarly, in the context of low-level, perceptual considerations, the arcing "move" depiction of an exchange is superior to the "grow" depiction of the same event.

Whether the perceptual benefits carry over to comprehension of the depicted algorithm is not clearly shown. In practice, the degree to which such perceptually beneficial techniques aid in comprehension may rely on the extent to which the cued behaviors are redundantly encoded in the depicted algorithm.

The use of popup questions that ask the user to answer questions about what they have just seen may encourage users to pay closer attention to an animation, and to thus develop a better understanding of the depicted algorithm. A danger exists, however, that over-attention to detail may prevent the user from "stepping back" and forming a higher-level view and understanding of the depicted algorithm. We seek to investigate this question in a follow-up study.

Future empirical studies will focus on other design features of algorithm visualizations including the effect of labeling, the effect of incorporating audio cues, effective use of popup questions, and pacing.

The latter item presents a number of possibilities for creating effective visualizations. As Faraday and Sutcliffe point out, timing of salient effects are key to a good thread of attention [Faraday98]. For example, they note that labels must be given 200ms focus per word. Similarly the guidelines they presented require minimum viewing times.

It is hoped that our research will help contribute to the design of pedagogical tools by promoting understanding of which graphical representations have an impact on performance. As is true for traditional pedagogical tools and graphical representations, a person needs to learn how to "read" charts or diagrams and understand the information being presented in relation to their current knowledge of the subject. If animations become a common tool, then the graphical representations will become a graphical vocabulary.

The framework, SSEA, has successfully been used in two empirical studies to date, including the one described in this thesis. It is also being used by the VizEval group at the University of Georgia for additional new studies, which follow from the findings of the initial studies. All finding and results achieve the goal of providing credibility to the conclusions drawn due to having a uniform environment and having participants perform the same tasks throughout the studies.

References

[Baecker98] Baecker, R. "Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science." In *Software Visualization: Programming as a Multimedia Experience*, 369-381. Cambridge, MA: The MIT Press, 1998.

[Baecker81] Baecker, R. and D. Sherman. Sorting Out Sorting. 30 minute color sound film. *Dynamic Graphics Project*, University of Toronto, 1981.

[Bartram01] Bartram, L. R. "Enhancing Information Visualization with Motion." Doctoral, Simon Fraser University, 2001.

[Bloom56] Bloom, B. S., Ed. *Taxonomy of Educational Objectives; The Classification of Educational Goals by a Committee of College and University Examiners*. New York, Longmans, Green, 1956.

[Brown98a] Brown, M. and M. A. Najork. "Algorithm Animation Using Interactive 3D Graphics." In *Software Visualization: Programming as a Multimedia Experience*, 119-135. Cambridge, MA: The MIT Press, 1998.

[Brown84] Brown, M. and R. Sedgewick. "A System for Algorithm Animation." *Computer Graphics*, 1984, **18**(3): 117-186.

[Byrne99] Byrne, M. D., R. Catrambone and J. T. Stasko. "Evaluating Animations as Student Aids in Learning Computer Algorithms." *Computers & Education*, 1999, **33**(4): 253-278.

[Davison01] Davison, H. J. and C. D. Wickens. *Rotocraft Hazard Cueing: The Effects on Attention and Trust*. In Proceedings of 11th International Symposium on Aviation Psychology, Columbus, Ohio, 2001.

[Faraday96] Faraday, P. and A. Sutcliffe. *An Empirical study of Attending and Comprehending Multimedia Presentations*. In Proceedings of ACM Multimedia, Boston, MA, USA, 1996.

[Faraday98] Faraday, P. and A. Sutcliffe. *Providing Advice for Multimedia Designers*. In Proceedings of Proceedings of the SIGCHI conference on Human factors in computing systems Los Angeles, CA, USA, ACM Press/Addison-Wesley Publishing Co., 1998.

[Grissom03] Grissom, S., M. F. McNally and T. Naps. *Algorithm Visualization in CS Education: Comparing Levels of Student Engagement*. In Proceedings of Proceedings of the 2003 ACM symposium on Software visualization San Diego, California ACM Press, 2003.

[Gurka96] Gurka, J. S. and W. Citrin. *Testing Effectiveness of Algorithm Animation*. Proceedings of the 1996 IEEE Symposium on Visual Languages, IEEE Computer Society, 1996.

[HTaylor02] Hamilton-Taylor, A. and E. Kraemer. "SKA: Supporting Algorithm and Data Structure Discussion." *ACM's 33rd SIGCSE Technical Symposium on Computer Science Education*, 2002, **34**(1): 58-63.

[Hansen02] Hansen, S., N. H. Narayanan and M. Hegarty. "Designing Educationally Effective Algorithm Visualizations." *Journal of Visual Languages and Computing*, 2002, **13**(2): 291 - 317.

[Hansen00] Hansen, S., N. H. Narayanan and D. Schrimpsher. "Helping Learners Visualize and Comprehend Algorithms." *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 2000, **2**(1).

[HFES] Davis, E. T., K. Hailston, E. Kraemer and A. Hamilton-Taylor. *Examining Perceptual Processing of Program Visualization Displays to Enhance Effectiveness*. In Proceedings of Annual Meeting of the Human Factors and Ergonomics Society, (In submission) 2006.

[Holmquist00] Holmquist, S. and N. H. Narayanan. *Tightly Coupling Authoring and Evaluation in an Integrated Tool to Support Iterative Design of Interactive Hypermedia educational manuals*. Proceedings of the conference on Designing Interactive Systems, New York City, New York, United States, ACM Press, 2000.

[Howard96] Howard, R. A., C. A. Carver and W. D. Lane. *Felder's Learning Styles, Bloom's Taxonomy, and the Kolb Learning Cycle: Tying it All Together in the CS2 Course*. Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, Philadelphia, PA, U.S.A., ACM Press, 1996.

[Hundhausen02] Hundhausen, C. D., S. A. Douglas and J. T. Stasko. "A Meta-Study of Algorithm Visualization Effectiveness." *Journal of Visual Languages and Computing*, 2002, **13**(3): 259-290.

[Jarc99] Jarc, D. J. "Assessing the benefits of interactivity and the influence of learning styles on the effectiveness of algorithm animation using web-based data structures courseware." Doctoral, The George Washington University, 1999.

[Khuri01] Khuri, S. *Designing Effective Algorithm Visualizations*. In Proceedings of First Program Visualization Workshop, Porvoo, Finland, 2001.

[Larkin87] Larkin, J. H. and H. A. Simon. "Why a Diagram is (Sometimes) Worth Ten Thousand Words." *Cognitive Science*, 1987, **11**: 65-99.

[Lawrence93] Lawrence, A. "Empirical Studies of the Value of Algorithm Animation in Algorithm Understanding." PhD, Georgia Institute of Technology, 1993.

[Lawrence94] Lawrence, A., A. Badre and J. Stasko. Empirically Evaluating the Use of Animations to Teach Algorithms. *Technical Report GIT-GVU-94-07*, Georgia Institute of Technology, 1994.

[Naps03] Naps, T., G. Robling, J. Anderson, S. Cooper, et al. *Evaluating the educational impact of visualization*. Working group reports from ITiCSE on Innovation and technology in computer science education, Thessaloniki, Greece, ACM Press, 2003.

[Narayanan02] Narayanan, N. H. and M. Hegarty. "Multimedia design for communication of dynamic information." *International Journal of Human-Computer Studies*, 2002, **57**(4): 279-315.

[Neisser67] Neisser, U. *Cognitive Psychology*. New York, Appleton-Century-Crofts, 1967.

[Pashler01] Pashler, H., J. C. Johnston and E. Ruthruff. "Attention and Performance." *Annual Review of Psychology*, 2001, **52**: 629-651.

[Price98] Price, B., R. Baecker and I. Small. "An Introduction to Software Visualization." In *Software Visualization: Programming as a Multimedia Experience*, 3-27. Cambridge, MA: The MIT Press, 1998.

[Rhodes04] Rhodes, P., S. Thomas, M. Ross, E. Kraemer, et al. VizEval - An Experimental Systemm for the Study of Program Visualization Quality. *Technical Report for The University of Georgia*, 2004.

[Ross04] Ross, M. "A Testing Environment for the Evaluation of Program Visualization Quality." Master of Science Thesis, University of Georgia, 2004.

[Scaife96] Scaife, M. and Y. Rogers. "External cognition: how do graphical representations work?" *International Journal of Human-Computer Studies*, 1996, **45**: 185-213.

[Sears00] Sears, C. R. and Z. W. Pylyshyn. "Multiple Object Tracking and Attentional Processing." *Canadian Journal of Experimental Psychology*, 2000, **54**(1): 1 - 14.

[Sperling86] Sperling, G. and B. A. Dosher. "Strategy and Optimization in Human Information Processing." In *Handbook of Human Perception and Performance, Vol. 1: Sensory Processes and Perception*, edited by K. R. Boff, L. Kaufman and J. P. Thomas. New York: Wiley, 1986.

[Stasko93] Stasko, J., A. Badre and C. Lewis. *Do algorithm animations assist learning?: an empirical study and analysis*. Proceedings of the SIGCHI conference on Human factors in computing systems, Amsterdam, The Netherlands, ACM Press, 1993.

[Stasko90] Stasko, J. T. "Tango: A Framework and System for Algorithm Animation." *Computer*, 1990, **23**(9): 27-39.

[Tudoreanu03] Tudoreanu, M. E. *Designing Effective Program Visualization Tools for Reducing User's Cognitive Effort*. In Proceedings of Proceedings of the 2003 ACM symposium on Software visualization San Diego, California ACM Press, 2003.

[Tudoreanu02] Tudoreanu, M. E., R. Wu, A. Hamilton-Taylor and E. Kraemer. *Empirical Evidence that Algorithm Animation Promotes Understanding of Distributed Algorithms*. In Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC '02), IEEE Computer Society, 2002.

[Zhang94] Zhang, J. "Representations in Distributed Cognitive Tasks." *Cognitive Science*, 1994, **18**: 87-122.

Appendix A

Sample Project Files

```
<SSEA>

<VizInput>
  <path>VizInputs.xml</path>
</VizInput>

<Questions>
  <path>Questions.xml</path>
</Questions>

<QuestionPopups>
  <path>QuestionPopups.xml</path>
</QuestionPopups>

<Code>
  <path>code.java</path>
</Code>

<LogFilesPath>
  <path>c:\experiment\logfiles\</path>
</LogFilesPath>

</SSEA>
```

Fig. A.1. A Sample SSEA.xml file. This file is needed to start SSEA and specifies where to find input files, questions, popup questions, the pseudocode, and where log files for sessions are to be saved.

```
<question type="TF">
  <text>This text would be a for a true or false answer.</text>
  <answer>a</answer>
</question>

<question type="M5">
  <text>What text would be a question?</text>
  <choices>
    <li>text for choice a</li>
    <li>text for choice b</li>
    <li>text for choice c</li>
    <li>text for choice d</li>
    <li>none of the above</li>
  </choices>
  <answer>d</answer>
</question>
```

Fig. A.2. Sample xml entries for questions. The ‘type’ attribute can have TF, YN, M4, or M5 values to specify True/False, Yes/No, multiple choice with 4 choices, and multiple choice with 5 choices, respectively.

```
<question type="M4" name="qName" input="all">
  <text>What question should popup?</text>
  <choices>
    <li>the answer</li>
    <li>another possible answer</li>
    <li>another option</li>
    <li>yet another answer</li>
  </choices>
  <answer>a</answer>
</question>
```

Fig. A.3. Sample xml entry for a popup question. The attribute 'name' specifies the string used to create the question action in the algorithm. The 'input' string can be 'all' to popup this question for all input data sets, or it can have the name of a specific input data set. The latter will cause the question to appear only if that input set is being animated.

```
<VizInput>
  <display>InputA</display>
  <file>InputA_GraphicFile.txt</file>
</VizInput>
```

Fig. A.4. Sample visualization input xml entry. This specifies a graphic file available to the user as input data set 'InputA'.

Appendix B

Sample Session Log File

```
<?xml version="1.0" encoding="UTF-8"?>

<SSEA_Session>
  <event time="1126987856588" action="startSSEA">
    <description>ComputerID</description>
    <description>AlgorithmID</description>
  </event>

  <event time="1126987856618" action="Animation">
    <description>StateChange</description>
    <optionalValue>To Begin</optionalValue>
    <step>0</step>
  </event>

  <event time="1126987860694" action="Animation">
    <description>Change Input</description>
    <optionalValue>From InputA to InputB</optionalValue>
    <step>0</step>
  </event>

  <event time="1126987861956" action="Animation">
    <description>StateChange</description>
    <optionalValue>Playing</optionalValue>
    <step>0</step>
  </event>

  <event time="1126987881804" action="Animation">
    <description>Speed</description>
    <optionalValue>164</optionalValue>
    <step>18</step>
  </event>
```

Fig. B.1. Sample session log file. Initial session startup information and animation interactions. This example shows the viewer changing the input set from InputA to InputB, press play and then adjusting the speed.

```
          <event time="1126987896736" action="QuestionPopup">
            <description>Selection</description>
            <optionalValue>c</optionalValue>
            <Name> exAllInputs</Name>
            <Input>InputC</Input>
           </event>

           <event time="1127087407353" action="Questions">
             <description>Change Question</description>
             <Number>2</Number>
           </event>
           <event time="1127087413261" action="Questions">
             <description>Selection</description>
             <optionalValue>b</optionalValue>
             <Number>2</Number>
           </event>
           <event time="1127087413261" action="Questions">
              <description>Selection</description>
              <optionalValue>a</optionalValue>
              <Number>2</Number>
           </event>

            <event time="1127087427912" action="endSSEA" />
            <answersSelected>
              <Q1>b</Q1>
              <Q2>b</Q2>
              <Q3>a</Q3>
            </answersSelected>
            <correctAnswers>
              <Q1>b</Q1>
              <Q2>c</Q2>
              <Q3>a</Q3>
            </correctAnswers>
          <CorrectPercentage>67</CorrectPercentage>

        <PopupAnswersSelected>
         <Question QP_NAME="exAllInputs" Input="InputA">
          unanswered </Question>
         <Question QP_NAME="exAllInputs" Input="InputB">
          unanswered</Question>
          <Question QP_NAME="exAllInputs" Input="InputC">c</Question>
          <Question QP_NAME="exOneInput" Input="InputA">d</Question>
        </PopupAnswersSelected>
        <PopupCorrectAnswers>
          <Question QP_NAME="exAllInputs" Input="InputA">a</Question>
          <Question QP_NAME="exAllInputs" Input="InputB">a</Question>
          <Question QP_NAME="exAllInputs" Input="InputC">a</Question>
          <Question QP_NAME="exOneInput" Input="InputA">d</Question>
        </PopupCorrectAnswers>
        <CorrectPercentage>50</CorrectPercentage>
```

Fig. **B.2. Sample sesion log file.** Displayed are entries recording a user answering a popup question, answering question number 2 and then changing number 2's answer. Final tallies for both sets of questions are summarized at the end of the log file.

Appendix C

Questions for Experiment

1. Were you familiar with the quicksort algorithm before this session?

   a. Yes
   b. No

   Answer: na
   Classification: None

2. Which best describes how quicksort works?

   a. By partitioning the array into two subarrays, using the median value of the array as the separator.
   b. For each element in the array, x, move x to the index equal to the number of elements that are less than x.
   c. By partitioning the array into two subarrays, and then sorting the subarrays independently.
   d. By swapping adjacent items that are out of order.

   Answer: c
   Classification: Comprehension

3. Which best describes the correct order of events in Quicksort.
   I. Call quicksort on the higher partition.
   II. Compare elements to the pivot, if less than or equal in value swap element into the lower section of the partition.
   III. Select a pivot.
   IV. Call quicksort on the lower partition.
   V. Swap the pivot into the position between the lower and higher partitions.

   a. I, II, III, IV, V
   b. III, II, V, IV, I
   c. III, II, I, IV, V
   d. none of the above

   Answer: b
   Classification: Comprehension

4. In this version of quicksort, which element is chosen as the pivot in the active partition?

a. The leftmost element
b. A Random element
c. The middle element
d. The rightmost element

Answer: d
Classification: Comprehension

5. When is the pivot swapped?

a. When a value less than or equal to the pivot value is found.
b. When a value greater than the pivot value is found.
c. At the end of partitioning a subset of the array.
d. none of the above

Answer: c
Classification: Comprehension

6. Which element is pivot swapped with?

a. an element that is less than or equal to the pivot value.
b. the last element in the lower partition.
c. the element at firstHigh
d. the element at end

Answer: c
Classification: Knowledge

7. For the input set 'SmallRandom', the number of calls to quicksort() is ___.

a. 2
b. 3
c. 5
d. 7
e. it cannot be determined

Answer: d
Classification: Comprehension

8. In quicksort, the comparison of an element with the pivot is done in which method(s)?

a. quicksort()
b. partition()
c. swap()
d. a and b
e. a and c

Answer: b
Classification: Knowledge

9. What two objects are being compared in the partitioning step?

a. the element at firstHigh and the element at pivot
b. the element at begin and the element at pivot
c. the element at firstHigh and the element at findLow
d. the element at findLow and the element at pivot

Answer: d
Classification: Knowledge

10. Swaps can occur between _____.

a. the element at begin+1 and the element at firstHigh
b. the element at firstHigh+1 and the element at pivot
c. the element at firstHigh and the element at findLow
d. the element at findLow and the element at firstHigh+1

Answer: c
Classification: Knowledge

11. Sort the following array using one iteration of quicksort. What does the higher partition look like? 5 7 8 2 9 6

a. 7 9 8
b. 9 8 7
c. 7 8 9
d. 6 7 9 8
e. 9 8 7 6

Answer: a
Classification: Application

12. Using the same array, what will the array be after the first call to partition()? 5 7 8 2 9 6

a. 2 5 6 7 8 9
b. 5 2 6 7 8 9
c. 5 2 6 7 9 8
d. none of the above

Answer: c
Classification: Application

13. What will the pivot value be at the start of the second call to quicksort()? 5 7 8 2 9 6

a. 2
b. 5
c. 6
d. 7

Answer: a
Classification: Application

14. If this algorithm takes x seconds to run on an array of n elements, about how long would you expect it to take to run on an array of 8n elements? (Assume values are in random order).

a. 8x
b. 24x
c. 64x
d. 64n
e. xn

Answer: b
Classification: Analysis

15. This algorithm uses recursion. (Definition of recursion: to divide a problem into subproblems of the same type).

a. True
b. False

Answer: a
Classification: Comprehension

16. When does the worst-case time for quicksort occur for an array of n elements?

a. When the pivot is always the largest or smallest element in the active partition.
b. When the input size is a power of b.
c. When the partition splits the array into 2 subarrays of equal lengths.
d. There is no predictor for worst-case time.

Answer: a
Classification: Analysis

Appendix D

Popup Questions for Experiment

1. What elements were just compared?

    a. 60 and 70
    b. 60 and 40
    c. 70 and 40
    d. 70 and 80
    e. 80 and 40

  Answer: c

2. What elements were just compared?

    a. 10 and 20
    b. 20 and 30
    c. 10 and 30
    d. none of the above

  Answer: b

3. What was the last comparison?

    a. Is 20 less than or equal to 70?
    b. Is 70 less than or equal to 20?
    c. Is 40 less than or equal to 70?
    d. Is 40 less than or equal to 20?
    e. Is 20 less than or equal to 40?

  Answer: e

4. What elements were just swapped?

    a. 20 and 10
    b. 70 and 40
    c. 20 and 70
    d. 10 and 40

  Answer: c

5. What elements were just swapped?

a. 50 and 70
b. 60 and 70
c. 60 and 80
d. 70 and 80

Answer: c

6. What element did 70 swap with?

a. 60
b. 70
c. 80
d. it did not swap

Answer: b

7. What variables were swapped?

a. firstHigh and findLow
b. end and firstHigh
c. begin and findLow
d. findLow and end

Answer: b

8. What variables were compared?

a. firstHigh and findLow
b. end and firstHigh
c. begin and findLow
d. findLow and end

Answer: d

Appendix E

Questions in Questionnaire

All questions were presented in a single screen with responses as radio buttons. A screenshot of the questionnaire can be seen in Figure 3.4.

1. I am:

a) Male
b) Female

2. This semester I am a :

a) Freshman
b) Sophomore
c) Junior
d) Senior
e) Super Senior (4+ years)

Course history section:
3a. CSCI 1100 Intro to Personal Computing

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3b. CSCI 1210 Intro to Computational Science

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3c. CSCI 1301 Intro to Computing & Programming

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3d. CSCI 1302 Software Development

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3e. CSCI 1730 Systems Programming

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3f. CSCI 2610 Discrete Math for Comp. Science

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3g. CSCI 2670 Theory of Computing

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3h. CSCI 2720 Data Structures

a) Previously Taken
b) Currently Enrolled
c) Never Taken

3i. CSCI 4000+ Any courses in the 4000 level (Pick the latest time enrolled in a 4000-level course)

a) Previously Taken
b) Currently Enrolled
c) Never Taken

Appendix F

Experiment Results

AlgoID	Tradional Results (%)	Popup Results (%)	Total time (min.)	Year	M/F
MC	93	100	39	Freshman	M
MC	80	75	44	Freshman	M
MC	73	37	28	Freshman	M
MC	66	50	20	Sophomore	M
MC	33	62	16	Sophomore	M
MC	53	87	31	Junior	M
MC	53	87	27	Junior	M
MC	53	87	16	Junior	M
MC	66	87	30	Junior	M
MC	60	62	14	Junior	M
MC	40	50	18	Junior	M
MC	66	87	13	Senior	F
MC	66	62	44	Senior	M
MC	60	87	24	Super Senior	M
MX	66	75	40	Freshman	F
MX	66	50	25	Sophomore	M
MX	20	25	14	Sophomore	M
MX	93	87	22	Junior	M
MX	86	87	41	Junior	M
MX	53	87	22	Junior	M
MX	33	62	15	Junior	M
MX	86	100	21	Senior	M
MX	46	50	28	Senior	F
MX	60	25	29	Super Senior	M
MX	80	75	27	Super Senior	M
MX	20	25	19	Super Senior	M

Table F.1 Result Summary for groups MC and MX.

AlgoID	Tradional Results (%)	Popup Results (%)	Total time (min.)	Year	M/F
GC	93	62	22	Freshman	M
GC	86	62	21	Sophomore	F
GC	73	75	26	Sophomore	M
GC	66	50	23	Sophomore	M
GC	80	87	17	Sophomore	F
GC	46	50	20	Sophomore	M
GC	46	62	14	Sophomore	M
GC	60	87	20	Junior	M
GC	46	87	15	Junior	M
GC	73	75	22	Junior	M
GC	53	50	26	Junior	M
GC	40	62	12	Junior	M
GC	46	100	17	Junior	M
GC	66	75	21	Senior	M
GC	86	75	33	Super Senior	F
GC	26	87	15	Super Senior	M
GX	46	37	11	Sophomore	M
GX	40	25	13	Sophomore	M
GX	73	62	14	Sophomore	M
GX	53	87	24	Junior	M
GX	73	37	28	Junior	M
GX	86	87	25	Junior	M
GX	86	87	14	Junior	M
GX	80	87	14	Junior	M
GX	73	62	23	Junior	M
GX	13	37	11	Junior	M
GX	66	37	14	Junior	M
GX	40	50	26	Junior	M
GX	40	0	19	Senior	M
GX	46	62	26	Senior	M
GX	53	62	19	Senior	F
GX	86	87	21	Super Senior	M
GX	73	87	59	Super Senior	F

Table F.2 Result Summary for groups GC and GX.