Empirical Studies of Animation-embedded Hypermedia Algorithm Visualizations

Steven Hansen, Daniel Schrimpsher, N. Hari Narayanan
and Mary Hegarty

Technical Report CSE98-06

November 4, 1998

Visual Information, Intelligence & Interaction Research Group
Department of Computer Science & Engineering
Auburn University
Auburn, AL 36849-5347

www.eng.auburn.edu/departments/cse/research/vi3rg/vi3rg.html
Empirical Studies of Animation-embedded Hypermedia Algorithm Visualizations

Steven Hansen*, N. Hari Narayanan, and Daniel Schrimpscher  
Visual Information, Intelligence & Interaction Research Group  
Department of Computer Science & Engineering  
Auburn University, AL 36849-5347

Mary Hegarty  
Department of Psychology  
University of California  
Santa Barbara, CA 93106

Abstract: If a “picture is worth a thousand words,” then why have attempts over the past decade to use pictures and animations to replace or supplement traditional instructional methods for teaching algorithms produced such disappointing results? In an earlier publication [Hansen Schrimpscher & Narayanan 1998] we described a research program based on the premise that a novel approach to algorithm animation design is needed in order to effectively harness the power of animations to enhance learning. In this report, we discuss the architecture of the resulting hypermedia algorithm visualization system, called HalVis, and present details of five empirical studies with HalVis which demonstrated significant advantages of this system when compared to learning by means of traditional instruction.

1. Introduction

Computer science students generally find algorithm analysis and design a hard subject because an algorithm describes a process that is abstract and dynamic, while the methods typically used to teach algorithms are not. For over a decade, researchers and educators have pursued the notion that using computer animations to illustrate the dynamic behavior of an algorithm could be an effective tool to help students overcome the difficult task of learning algorithms. In fact, numerous studies and experiments have been done, attempting to prove that animations can indeed improve the learning of the challenging behavioral aspects of algorithms [Hundhausen 1996]. While the pictures and animations are enthusiastically received by the students [Stasko 1997], most of the studies have not proven conclusively that these visual tools actually improve learning [Badre et al. 1991; Byrne et al. 1996; Stasko, et al. 1993]. Is it because animation is an ineffective teaching medium? Intuition tells us this is not likely.

Ongoing research in the areas of multimedia, usability and cognitive science sheds insight into factors that contribute to the design of effective visualization systems [Petre et al. 1998; Narayanan & Hegarty 1998], and suggests that previous attempts at using animation to teach algorithm behavior were unsatisfactory not because of a flaw with animation as a technique, but perhaps because of the approach used to convey the animations. Our research is based on the hypothesis that animations are indeed powerful vehicles for effectively conveying the dynamic behavior of algorithms, but that a rethinking of algorithm animation design is required in order to harness its power to enhance learning. By enhanced learning, we mean a better understanding of the description and behavior of an algorithm resulting in an increased ability to accurately answer questions about the pseudocode and operations of the algorithm. We have developed a framework for designing interactive multimedia presentations of algorithms called Hypermedia Algorithm Visualizations (HalVis)—the term visualization suggests a richer process than merely watching an animation, and the term hypermedia reflects the use of multiple media, semantic links and other cognitive devices to help the student form accurate mental models of algorithms.

In the next section we present the key features and the architecture of HalVis. Section 3 contains detailed descriptions of five empirical studies conducted over a 12 month period and involving over 130 undergraduates, in which we compared HalVis with traditional means of learning about algorithms—lectures, textbooks and problem solving. Results from these experiments indicate that learning by interacting with animation-embedded

* Direct all correspondence to this author. Email: hansen@eng.auburn.edu
hypermedia visualizations of algorithms is more effective than learning by reading a textbook, listening to a
lecture, or interacting with an algorithm animation typical of extant research on this topic. We also found that
learning from HalVis is as effective as learning by reading a textbook-style mixed-mode (containing graphics as
well as textual descriptions) presentation combined with problem solving activities. A general discussion of the
implications of these experiments and the plans for future research appear in Sections 4 and 5.

2. HalVis: Hypermedia Algorithm Visualizations

2.1. Key features of the design framework

Under the design framework we have developed, a hypermedia algorithm visualization is much more than an
algorithm animation. It provides an interactive environment that elicits active student participation using a
carefully orchestrated presentation of information in various media (such as animations, text, static diagrams,
aural narratives, etc.) with appropriate temporal, spatial and hyperlink connections. This framework incorporates
the following key features.

• Embedding animations within a hypermedia visualization that also employs textual descriptions, audio
narratives and static diagrams to provide contextual information. The focus is not on the animation, but on
providing relevant and sufficient information in appropriate media to support achieving the learning objectives.

• Providing three distinct kinds of animations to illustrate qualitatively different views of algorithm behavior.
Initially, the algorithm’s essential behavior is illustrated using animated real world analogies, and bridges
between the analogy and the abstract components of the algorithm as well as the concrete graphical
representations used to depict the algorithm in later animations are provided. The motivation behind this feature
is two-fold. First, it has already been observed that students tend to employ analogies in describing how
algorithms operate [Douglas et al. 1995; Stasko 1997]. Second, analogies can serve to provide a form of
scaffolding [Hmelo & Guzdial 1996] for subsequent learning. Next, a micro-level animation of specific
algorithmic operations in tandem with pseudocode highlighting and textual explanations is provided. Finally, a
macro-level animation illustrates the algorithm’s aggregate behavior and performance characteristics on large
data sets.

• Presenting algorithm animations in discrete chunks accompanied by explanations of the specific actions
being accomplished. By chunking animations into meaningful “bite-sized” pieces and providing logical pauses
between chunks, the student is able to better digest the abstract, dynamic information being presented. Allowing
the student to adjust the level of chunking tailors the flow of information to meet individual needs. This is in
stark contrast to most current algorithm animation systems which present the detailed dynamics as a one-shot,
stand-alone show that is entertaining to watch but tends to obscure the very details a student needs to learn.

• Encouraging student participation by allowing rich interactions with the animations and using probes or
questions that stimulate thinking and foster self-explanations [Chi et al. 1989]. Students are prompted to input
data sets of their choosing to explore algorithm behavior more thoroughly. The system periodically poses
questions to the student. The simplest form is called a “tickler”, which is a question that pops up in random
order but always in the appropriate context. Tickler questions are open-ended, focus student attention on
specific issues, challenge their understanding and promote self-explanations to improve comprehension. Their
answers are not entered into the computer nor is feedback provided. We also place multiple choice questions
requiring students to enter answers in order to proceed further at “articulation points” between modules of the
visualization. In this case, immediate feedback is provided by the system.

2.2. System Architecture

We developed a system called HalVis to test the components of our design framework. HalVis is implemented
using Asymetrix Toolbook. As of this writing, it contains visualizations of four sorting algorithms (BubbleSort,
SelectionSort, MergeSort and QuickSort) and one graph algorithm (Dijkstra’s Shortest Path). Each algorithm
visualization in HalVis consists of the four modules described below:
• **Fundamentals:** This module contains information about basic building blocks of algorithms common to virtually all algorithms. Examples of concepts explained in this module include Comparing & Swapping Data, Looping Operation and Recursion. This module is not directly accessible to the student. It can be accessed only through hyperlinks from other modules, so that the basic information is presented *on demand* (in response to a learner request in the form of clicking on a hyperlink) and *in context* (of algorithm-specific information within which the hyperlink is embedded).

• **Conceptual View:** This module introduces a specific algorithm in very general terms using a real world analogy. For instance, BubbleSort is introduced using a flask of water with bubbles that rise to the surface according to their size, and the MergeSort algorithm uses animated playing cards to illustrate dividing and merging to create a sorted sequence (Figure 1). This module uses animations, text and audio to provide the student with a general description of the algorithm. The animated and interactive real-world analogy is intended to function as scaffolding, and to provide bridging information to facilitate the learner’s progress from the visual elements in the analogy to the data structures and algorithm operations in later modules.

![Software Visualization System](image)

**Introduction to MergeSort**

**MergeSort** takes its name from the fact that it uses a **merge procedure** to create an ordered sequence. It uses just two simple operations, one that splits a sequence into two parts and another that merges two sequences into a single, ordered one.

Starting with a single dataset, **MergeSort** splits it into two halves, recursively sorts the halves, and merges the halves back into a single dataset.

![Show Me The Split Operation](image)

**Figure 1. Conceptual View of the Merge Sort Algorithm**

• **Detailed View:** This module describes the algorithm at a very detailed level using two presentations. One consists of a detailed textual description of the algorithm alongside a pseudocode representation of it. Embedded in the text are hyperlinks to related information in the Fundamentals module. The second presentation (Figure 2) contains four windows that depict various aspects of the algorithm’s behavior. The Execution Animation window shows how steps of the algorithm modify data structures using smooth animation. The animation is chunked at multiple levels of granularity corresponding to meaningful units of the algorithm’s behavior, with the level of chunking selectable by the learner. At the lowest level, the animation displays the execution of an individual statement, pausing for the learner’s signal to proceed. The next level corresponds to a logical operation, like
completion of a single pass in a loop. At the highest level, the animation proceeds to completion without pausing. The Execution Status Message window provides comments and textual feedback to the student about key events and actions during execution. This is also available as an audio commentary. The Pseudocode window shows the steps involved in the algorithm, which are highlighted synchronously with the animation. Finally, the Execution Variables window displays a scoreboard-like panorama of the variables involved in the algorithm and their changing values. Before launching the animation, students can change the data input to the algorithm as well as the speed and granularity of animation and feedback. Execution of each step of the algorithm affects the display in the four windows simultaneously. Figure 2 shows seven data elements to be sorted using the MergeSort algorithm. When the user presses the ShowMe button, the four windows spring to life, moving the seven data items as needed and pausing between chunks until the algorithm is finished. HalVis intentionally limits the number of data items in the Execution Animation window to focus attention on the micro-behavior of the algorithm.

Figure 2. Detailed View of the MergeSort Algorithm

- **Populated View:** This module provides an animated view of the algorithm on large data sets to make its macro-behavior explicit. In this view (Figure 3), many of the details presented in the Detailed View are elided to enhance the student’s focus on the algorithm’s aggregate performance. Animations embedded in this view correspond closely with algorithm animations found in earlier systems. A novel feature is a facility for the student to make predictions about different parameters (e.g., number of comparisons, swaps, or recursive calls) of algorithm performance and then compare those against the actual performance when the animation is running. When the learner presses the ShowMe button, the system prompts for predictions from the learner, initializes the bars (data) into random order and proceeds to execute the algorithm, animating its progress by highlighting and moving the graphic objects. Color coding is used to convey information such as already processed data and data elements currently being processed.
Questions: This module presents the student with several questions at articulation points between the other modules to facilitate and measure comprehension. A combination of multiple choice, true-false, and algorithm debugging questions are provided. Students get immediate feedback on their answers. HalVis also uses randomly generated, context-sensitive ticklers in other modules to help focus the student's attention on key aspects of the algorithm being studied.

![Figure 3. Populated View of the MergeSort Algorithm](image)

3. Experiments: Learning with Hypermedia Algorithm Visualizations

Common teaching instruments include textbooks, lectures, and laboratory experiences. More recently, instructors have also started using algorithm animations [Stasko 1997]. A survey of computer science instructors [Badre et al. 1991] showed that over 81% of instructors use at least one of these methods to teach algorithms to students. Previous experiments on the effectiveness of algorithm animations as teaching tools compared experimental groups employing a combination of instructional media, as shown in Table 1.
We conducted a series of experiments to evaluate the effectiveness of hypermedia visualizations of algorithms, specifically to validate our hypothesis that the visualizations would prove to be more beneficial than traditional teaching methods. Our hypothesis was that students would learn more effectively using HalVis than from other teaching methods, as indicated by their performance in pre-tests and post-tests. We compared student performance after they interacted with algorithms in HalVis and learned about algorithms using a typical instructional approach. The five individual experiments are summarized in Table 2, showing the student level, the learning media used by the comparison groups, and the algorithm(s) studied. The first two experiments compared learning with HalVis to learning from textbooks alone, using second year computer science undergraduates for one experiment and third year students for the other experiment. Extending this comparison further, experiment III compared learning with HalVis to learning from a compilation of the best descriptions and depictions extracted from a survey of 18 textbooks followed by solving a set of exercises. The fourth experiment compared learning from HalVis to learning from lectures. Finally, the fifth experiment compared learning from HalVis to learning from the combination of a typical algorithm animation and text. These experiments, we felt, would help us determine the comparative effectiveness of HalVis. If HalVis proved more effective than other instructional media, then it could be inferred that HalVis coupled with other media (textbooks, lectures, etc.) would be at least as effective, and possibly even more effective. In these experiments we used pre- and post-tests to measure students’ ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. We did not differentiate between visual and verbal learners since HalVis contains rich textual and visual presentations to support both kinds of learner dispositions.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Level</th>
<th>Comparison Groups</th>
<th>Algorithm(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lawrence et al. 1994</td>
<td>K13</td>
<td>Animation + Lecture + Laboratory Slides</td>
<td>Slides + Lecture + Lab</td>
</tr>
<tr>
<td>Crosby &amp; Stelovsky 1995</td>
<td>K13</td>
<td>Animation + Lecture + Homework Slides</td>
<td>Slides + Lecture + Homework</td>
</tr>
<tr>
<td>Stasko et al. 1993</td>
<td>K17</td>
<td>Animation + Text</td>
<td>Text</td>
</tr>
<tr>
<td>Badre et al. 1991</td>
<td>K14</td>
<td>Animation + Handout</td>
<td>Lecture + Handout</td>
</tr>
<tr>
<td>Byrne et al. 1996</td>
<td>K15</td>
<td>Animation + Video Lecture + Text Diagrams</td>
<td>Diagrams + Video Lecture + Text</td>
</tr>
</tbody>
</table>

Table 1. Prior Experiments on Algorithm Animation

3.1. Experiment I
The first experiment involved students relatively new to algorithmic study, and compared the effectiveness of learning using algorithm visualizations to learning from text. We chose to use the MergeSort algorithm for this experiment because first-year students find its recursive structure somewhat challenging to comprehend.

3.1.1 Subjects
The experiment involved 28 undergraduates enrolled in an introductory data structures and algorithms course at Auburn University. Subjects received course credit for their participation. In the first week of the quarter, subjects completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information to rank students and create a matched pair of groups: one group (called the "Text" group) would learn about the MergeSort algorithm using textbook descriptions, and another group (called the "Algorithm
Visualization” (AV group) would learn the MergeSort algorithm using the HalVis algorithm visualization tool. Twelve students in the Text group and sixteen students in the AV group completed the experiment.

3.1.2 Materials

Text Group: The Text group received a photocopied 6-page extract from a textbook [Dale Lilly & McCormick 1996] that discussed the MergeSort algorithm. The handout included a description and analysis of the algorithm, various diagrams, and program code.

AV Group: The AV group learned about the MergeSort algorithm using the HalVis system with no supplementary materials provided.

Test Questions: A pre-test/post-test combination measured individual learning performance with 9 questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithm and the post-test measured changes resulting from experimental conditions.

3.1.3 Procedure

We timed the experiment to follow class lectures covering basic program design and fundamental data structures, but precede lectures that covered sorting algorithms. Towards the middle of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the MergeSort algorithm. The pre-test results helped us verify that the two groups were evenly balanced, and provided a baseline against which to compare subsequent changes. The pre-test scores indicated that the subjects did not know this algorithm and that the groups were evenly distributed (Average = 27% for the Text group and 28% for the AV group).

During the following week, the AV group met in a public computer laboratory. They were given a 5 minute introduction to HalVis, which oriented them to the various screens they would encounter and provided them with basic navigational tips. The students were then assigned to a computer and instructed to interact with the software until they felt they understood the MergeSort algorithm. The computers were Pentium-class systems with 15 inch color monitors. Subjects were not given any text material to study, nor had they been exposed to the MergeSort algorithm in the class prior to the experiment. There was no time limit, so when each subject indicated he/she was done, he/she was given a post-test that helped measure knowledge improvement. No student in the AV group took more than 60 minutes for the entire experiment.

On the same day the Text group met in a classroom, and was provided with photocopied pages from their textbook describing the MergeSort algorithm. They were not provided any other information, nor had they been exposed to the MergeSort algorithm during class lectures. They were asked to learn the MergeSort algorithm from the materials provided, with no time limit imposed. When they finished studying the explanatory materials provided, they were given a post-test and allowed to leave. No student in the text group took more than 45 minutes for the entire experiment.

3.1.4 Results

The overall results are shown in Figure 1A. The pre-test results indicate that both groups were equally unfamiliar with the MergeSort algorithm. The post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 74% compared to the Text group’s 43%, and the results are significant for both the overall performance (F(1,27)=10.9, p<0.003) and for improvement (F(1,27)=6.7, p<0.015). The statistical summary is shown in Table 1A.
The results are further summarized as box plots in Figure 1B. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group.

Figure 1C shows the individual pre-test and post-test scores of each subject by group. Each participant is indicated by his/her randomly assigned ID number on the vertical axes. Pairs of horizontal bars indicate each participant’s test performance. The light bars represent pre-test performance and the dark bars show post-test performance. No bar is shown when the corresponding score is zero. The tables below provide the percentage grade obtained by each participant in pre- and post-tests. It is interesting that every subject in the AV group improved his/her knowledge, but two subjects in the Text group (T11 and T05) actually did worse.
3.1.5 Discussion

These results suggest that novice students perform better in answering conceptual and procedural questions about the MergeSort algorithm after learning from a hypermedia algorithm visualization than after studying a typical textbook. However, there are several factors that must be mentioned to keep these results in perspective. First, one could argue that a different textbook could have led to different results. There are good textbooks and there are bad textbooks, and an experienced instructor can usually recognize either. But opinions differ on what constitutes a good textbook, otherwise there would only be one accepted textbook and all courses would use it. We believe the material we used was from a well-written book. Experiment III probed this issue further. Second, only novice students participated in this experiment. It is possible that more advanced students may benefit more from a textbook explanation of an algorithm. Experiment II investigated this possibility. Third, difference in student motivation between the groups could have influenced the results. The level of enthusiasm observed in the HalVis group was much higher than in the Text group. The novelty of the visualization and the interactive features of HalVis seemed to engage the students’ interest. In contrast, there was nothing new or uniquely motivating for the Text group. Fourth, familiarity with the learning materials provided could have had an influence. The Text group did not have to acquaint themselves with a new user interface, software system or learning from interactive visualizations. They were all familiar with reading and learning from a textbook. The students in the AV group had to contend with a new interface and a different way of learning. If this factor indeed played a role, the AV group exhibited a higher level of comprehension despite any additional cognitive effort involved in learning the interaction and navigation facilities of HalVis.
3.2. Experiment II
This experiment was similar to Experiment I in that the goal was to compare the effectiveness of learning using HalVis to learning from text. Our aim was to test whether results of Experiment I could be replicated with more sophisticated algorithms and higher level students. We asked students to learn the MergeSort and QuickSort algorithms. Unlike the previous experiment, these participants completed all components of the experiment in one day: a pre-test, learning two algorithms, and a post-test.

3.2.1 Subjects
This experiment involved 22 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Like Experiment I, participants were ranked based on academic ability (based on course performance up through mid-term grades, GPA, and ACT/SAT scores) and assigned to a matched pair of groups: a “Text” group and an “Algorithm Visualization” (AV) group. Students were given extra credit for participating. Eleven students in the Text group and eleven students in the AV group completed the experiment.

3.2.2 Materials
Text Group: The Text group received ten page photocopied extract from their textbook [Weiss, 1993] that discussed the MergeSort and QuickSort algorithms.

AV Group: The AV group learned about the MergeSort and QuickSort algorithms using the HalVis system with no supplementary materials provided.

Test Questions: A pre-test/post-test combination measured individual learning performance with 18 questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test results measured knowledge improvement resulting from the experimental conditions.

3.2.3 Procedure
We timed the experiment to precede the class lectures that dealt with sorting algorithms. Towards the middle of the quarter, on the day of the experiment, all participants met in a classroom and completed the pre-test. Afterwards, members of the AV group were taken to a public computer laboratory, while the Text group remained in the classroom.

In the computer laboratory, the AV group was given a brief navigation-only orientation to the HalVis system, then assigned to individual computers to interact with the software and learn the algorithms. Students were allowed to take as much time as needed. They did not have access to any supplementary materials. As each subject finished interacting with the visualizations, a post-test was given. All subjects completed the experiment in less than 2 hours.

The Text group was given the extract from their course textbook. This contained a typical combination of textual descriptions and explanations, diagrams, pseudocode and program examples. Like the AV group, there was no time constraint. When a subject signaled completion of studying the materials, he/she was given the post-test and allowed to leave. Students in the Text group averaged 75 minutes to take the pre-test, study the textual materials and complete the post-test.

3.2.4 Results
The overall results are shown in Figure 2A. The pre-test results indicate that both groups were equally unfamiliar with both algorithms. The post-test averages show a significant improvement for the AV group over the Text group. The AV post-test average was 63% compared to the Text group’s 44%, and the results are significant for both the overall results ($F(1,21)=4.96$, $p<0.038$) and for improvement ($F(1,21)=9.29$, $p<0.006$). The statistical summary is given in Table 2A, showing both aggregate and algorithm-specific results for each group.
Figure 2A. Experiment II Overall Summary

### Statistical Summary

<table>
<thead>
<tr>
<th></th>
<th>Pre-test</th>
<th>Post-test</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text (MS)</td>
<td>44%</td>
<td>53%</td>
<td>9%</td>
</tr>
<tr>
<td>AV (MS)</td>
<td>48%</td>
<td>71%</td>
<td>23%</td>
</tr>
<tr>
<td>Text (QS)</td>
<td>10%</td>
<td>35%</td>
<td>25%</td>
</tr>
<tr>
<td>AV (QS)</td>
<td>4%</td>
<td>55%</td>
<td>51%</td>
</tr>
<tr>
<td>Text (MS+QS)</td>
<td>27%</td>
<td>44%</td>
<td>17%</td>
</tr>
<tr>
<td>AV (MS+QS)</td>
<td>26%</td>
<td>63%</td>
<td>37%</td>
</tr>
<tr>
<td>F(1,21)</td>
<td>0.02</td>
<td>4.96</td>
<td>9.29</td>
</tr>
<tr>
<td>p</td>
<td>p&lt;0.89</td>
<td>p&lt;0.038</td>
<td>p&lt;0.006</td>
</tr>
</tbody>
</table>

Table 2A. Experiment II Statistical Summary
The results are summarized as box plots in Figure 2B. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group.

3.2.5 Discussion

These results parallel those of Experiment I in suggesting that students perform better in answering conceptual and procedural questions about the MergeSort and QuickSort algorithms after using a hypermedia visualization system to learn than after studying a typical textbook. As shown in Table 2B, the more advanced status of the students led to higher prior knowledge scores for the MergeSort algorithm (44% and 48%), which was much higher than the pre-test levels observed in the novice students of Experiment I (27% and 28%). This is not a surprising result, since it is reasonable to expect the more advanced students could have been exposed to the MergeSort algorithm before, and even if not, could grasp the essential concepts more readily than novice students. These two experiments together suggest that visualization is a more effective learning method than learning from a textbook regardless of the level of students. As Table 2B indicates, AV groups improved their performance by approximately three times compared to the performance improvement of the Text groups in the two experiments. Interestingly, the AV groups in both experiments reached the same level of performance after interacting with HalVis though they started off with different levels of prior knowledge.

<table>
<thead>
<tr>
<th>Experiment I</th>
<th>Experiment II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-test</td>
<td>Post-test</td>
</tr>
<tr>
<td>Text Group</td>
<td>27%</td>
</tr>
<tr>
<td>AV Group</td>
<td>28%</td>
</tr>
</tbody>
</table>

Table 2B. Comparison of Results of Experiments I and II

As with Experiment I, there are several factors to consider to keep these results in perspective, such as textbook quality, student experience and ability, motivation, and familiarity with learning approaches used in the experiment. This experiment added a significant load to the learning task, requiring students to learn two fairly challenging algorithms in one session, as well as complete a pre-test and a post-test in the same session. This cognitive load could have affected the test performance of both groups.
3.3. Experiment III
This experiment involved students relatively new to algorithmic study, and compared the effectiveness of learning from HalVis to learning from text and problem solving. The goal was to provide one group with the best possible descriptive and depictive materials and a set of exercises, in order to investigate the limits of learning from HalVis by comparing it with learning from carefully designed, detailed textual and diagrammatic explanatory materials coupled with problem solving. We chose to use the BubbleSort and SelectionSort algorithms for this experiment. While these are relatively simple algorithms, we felt that asking participants in this experiment who were novice students, not yet exposed in depth to the subject of sorting, to learn both algorithms in one session would represent a reasonable cognitive load.

3.3.1 Subjects
The experiment involved 25 undergraduate computer science students enrolled in an introductory data structures and algorithms course at Auburn University. Subjects received course credit for their participation. In the first week of the quarter, they completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information to rank students and assign them to a matched pair of groups: one group (called the "Text" group) that would learn about the BubbleSort and SelectionSort algorithms using a handout that we created and then completing a series of problem solving exercises, and another group (called the "Algorithm Visualization" (AV) group) that would learn the same algorithms using the HalVis algorithm visualization tool alone. Twelve students in the Text group and thirteen students in the AV group completed the experiment.

3.3.2 Materials
Text Group: The Text group received an 8 page explanation that contained both textual descriptions and graphic depictions of the BubbleSort and SelectionSort algorithms, along with several exercises. We carefully designed this handout to address the issue of the quality of text used in the previous experiments. After reviewing the descriptions, depictions and examples of the BubbleSort and SelectionSort algorithms contained in 19 textbooks published between 1974 and 1997 [Aho Hopcroft & Ullman 1974; 1983; Baase 1988; Dale Lilly & McCormick 1996; Delillo 1993; Hansen 1983; Harel 1992; Hillam 1994; Horowitz & Sahni 1976; Kingston 1990; Korsch 1986; Kozen 1992; Manber 1989; Nance & Naps 1995; Rowe 1997; Sedgewick 1988; Shaffer 1997; Singh & Naps 1985; Weiss 1993], we selected the three best explanations we could find. These explanations were then edited to increase clarity and merged to create a handout containing textual and pictorial explanations of the two algorithms. We also developed and included a set of "end of chapter" style exercises in this handout for students to solve after perusing the explanations. This handout is provided in Appendix A.

AV Group: The AV group learned about the BubbleSort and SelectionSort algorithms using the HalVis system with no supplementary materials provided.

Test Questions: A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. A copy of the post-test is included as Appendix B.

3.3.3 Procedure
As with the previous experiments, we timed the experiment to follow class lectures covering basic program design and fundamental data structures, but precede those that covered sorting algorithms. Towards the middle of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the BubbleSort and SelectionSort algorithms. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the two groups were evenly balanced.

The following week, the AV group met in a computer laboratory on campus. They were given a 5 minute introduction to HalVis, which oriented them to the various screens they would encounter and provided them with navigational tips. The students were then assigned to a computer and instructed to interact with the software until they felt they understood the two algorithms. The computers were Pentium-class systems with 15 inch color monitors. Subjects were not given any text material to study, nor had they been exposed to the algorithms in class lectures. There was no time limit, so when each subject indicated he/she was done, he/she was given a
post-test that helped measure knowledge improvement. No student in the AV group took more than 90 minutes for the entire experiment.

On the same day, the Text group met in a classroom, and was provided with the handout described above. They were asked to read and understand the materials and then to solve the set of exercises at the end. They were not provided with any additional information, nor had they been exposed to these algorithms in class lectures. When they finished studying the descriptive materials and attempting the exercises, they were given a post-test and allowed to leave. No student in the text group took more than 60 minutes for the entire experiment.

### 3.3.4 Results

The overall results are shown in Figure 3A. The pre-test scores indicated that the subjects did not know these algorithms and that the groups were evenly matched (Average = 35% for the Text group and 31% for the AV group). The post-test averages show an improvement of 30% for the AV group to only 22% for the Text group. These results, while indicating better learning for the AV group, are not statistically significant as can be seen in Table 3A.

![Figure 3A. Experiment III Overall Summary](image)

<table>
<thead>
<tr>
<th>Statistical Summary</th>
<th>Pre-test</th>
<th>Post-test</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>35%</td>
<td>57%</td>
<td>22%</td>
</tr>
<tr>
<td>AV</td>
<td>31%</td>
<td>61%</td>
<td>30%</td>
</tr>
<tr>
<td>F(1,24)</td>
<td>0.36</td>
<td>0.32</td>
<td>0.82</td>
</tr>
<tr>
<td>p</td>
<td>p&lt;0.55</td>
<td>p&lt;0.57</td>
<td>p&lt;0.37</td>
</tr>
</tbody>
</table>

Table 3A. Experiment III Statistical Summary

The results are summarized as box plots in Figure 3B. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. Examining the box plot reveals a wider dispersion of scores in the post-test results of the text group, but a much tighter clustering in the post-test results of the AV group.
Figure 3C shows the individual pre-test and post-test scores of each subject by group. Each participant is indicated by his/her randomly assigned ID number on the vertical axes. Pairs of horizontal bars indicate each participant’s pre- and post-test scores. The light bars represent pre-test performance and the dark bars show post-test performance. The tables below provide the percentage grade obtained by each participant in pre- and post-tests. Two subjects in the Text group (T06 and T07) and one subject in the AV group (V10) did worse after the experiment than they did on the pre-test. This is the first time, in the three experiments conducted thus far, where the performance of a member of the AV group decreased.
Figure 3D shows performance by question (see Appendix B for the questions) across the two groups for the post-test. For each question on the vertical axis, the horizontal axis provides the number of subjects who answered it correctly. It can be seen that only in three questions did the text group outperform the AV group: questions 7 and 11 dealing with worst case orderings, and question 10 that examined the number of swap operations the SelectionSort algorithm would require.

Figure 3E shows performance across pre- and post-tests by the AV group. For each question on the vertical axis, the horizontal axis provides the number of subjects who answered it correctly. If none answered a question correctly (Q5b, pre-test), the corresponding bar is not shown. This figure illustrates the substantial increase in the number of students answering correctly the questions dealing with algorithm recognition (Q1, Q3), behavior (Q2, Q6, Q8, Q10, Q12, Q14) and data ordering (Q2, Q4, Q5, Q7, Q9, Q11, Q13) after interacting with the visualizations of the algorithms.

3.3.5 Discussion

The combination of using simpler algorithms, significantly improving the text and asking students to engage in problem solving had a marked impact on the Text group’s performance. It improved to a level on par with that of the AV group. Our conclusion from this experiment is that AV appears to be as effective for novice students to learn about algorithms as learning from carefully crafted textual materials coupled with problem solving exercises. Factors we did not control for, such as motivation and familiarity with textual descriptions and exercises may also have influenced the results.
Figure 3D. Comparison of Post-test Responses, by Group
Figure 3E. Comparison of AV Group Pre-test and Post-test Responses
3.4.  **Experiment IV**  
This experiment was designed to compare HalVis with classroom lectures, and also to investigate how HalVis and lectures can together contribute to learning. Our hypothesis was that HalVis would facilitate learning much more than a lecture. If so, HalVis used in conjunction with lectures should assist learning even more. We wanted to (1) verify our hypothesis that students learning from HalVis would outperform students learning by lecture alone, (2) measure additional learning obtained by combining lecture and HalVis, and (3) investigate whether the order (HalVis before lecture, or vice versa) would make a difference in performance. Participants in this experiment were novice computer science students, and the algorithms used were SelectionSort and MergeSort.

3.4.1  **Subjects**  
The experiment involved 27 undergraduates enrolled in an introductory data structures and programming course at Auburn University. Subjects received extra credit for their participation. In the first week of the quarter, the subjects completed a demographic survey providing information such as GPA, ACT and SAT scores. We used this information and current class standing to rank and assign students to a matched pair of groups, a Lecture-Visualization (LV) group and a Visualization-Lecture (VL) group. The LV group received a class lecture discussing the algorithms, then interacted with the visualizations of the two algorithms in a computer laboratory. The VL group interacted first with HalVis, then attended the class lecture covering the algorithms. There were nine students in the LV group and eleven in the VL group that completed the three components of the experiment; seven students (six in the LV group and one in the VL group) did not complete all three components and their data is not included in the analysis below.

3.4.2  **Materials**  
**Lecture:** All participants attended a lecture on the two sorting algorithms provided in two consecutive 50-minute class sessions conducted by Dr. Dean Hendrix, an Auburn University Computer Science and Engineering Department faculty. His lecture consisted of verbal instruction accompanied by blackboard diagrams, overhead transparencies, and a lecture summary handout. He responded to several questions from students in the class during the lecture.

**Visualization:** Both groups interacted with the same visualizations of both the SelectionSort and MergeSort algorithms, with no supplementary materials provided.

**Test Questions:** A pre-test/mid-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithms. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes.

3.4.3  **Procedure**  
The phases of this experiment were carefully synchronized with the class syllabus. The week before the scheduled lecture about sorting algorithms, students were given a pre-test to measure prior knowledge about the two algorithms and to provide a baseline to measure subsequent changes.

The day before the lecture, the VL group met in a computer laboratory on campus, interacted with HalVis to learn the two algorithms, and completed a mid-test. This test measured changes in knowledge resulting from HalVis interaction. The same lecture was attended by both the groups. We chose to use a regular classroom lecture over a videotaped one to allow student interaction with the professor and to simulate a realistic learning environment. Having both groups attend the same lecture eliminated variations between separate lectures.

The day after the lecture, the LV group met in a computer laboratory on campus and first completed the same mid-test taken by the VL group. This test measured changes in knowledge resulting from the lecture for the LV group. Then the group was assigned to computer terminals and asked to interact with HalVis to learn the two algorithms. When they felt they understood the algorithms, they were asked to complete a post-test and allowed to leave. On this same day, the VL group met in a classroom and completed the same post-test. The post-test measured the final knowledge level of the two groups after both the lecture and the interactive sessions.
We designed the experiment to minimize outside interactions that might affect the results. First, while we did not explicitly instruct students not to read the course textbook or try to learn more about the algorithms from other sources, only one of the algorithms was covered in the course textbook. A question in the mid- and post-tests asked the students whether they had read about the algorithms elsewhere. None of the students indicated that they had read about the algorithms in the mid-test, and four indicated that they had in the post-test. We did however ask students to refrain from discussing any aspect of the experiment during its course.

Figure 4A. Comparison of Group Pre-Test, Mid-Test and Post-Test Responses

3.4.4 Results

Examining the results depicted in Figure 4A, we see that both groups were relatively unfamiliar with the algorithms based on their pre-test averages (7% for the VL group and 19% for the LV group). The mid-test results indicate the VL group learned more than the LV group. Following the session with HalVis, the VL group average score was 70% compared to 44% for the LV group after the lecture. Despite having less prior knowledge about the algorithms, in the mid-test, the VL group after interacting with the algorithm visualizations
significantly outperformed the LV group that received a classroom lecture. The improvement in additional knowledge gained by the VL group from the following lecture session was marginal, whereas the visualization helped the LV group catch up with the VL group by the time of the post-test.

Another view of the results is depicted in Figure 4B, showing the improvement in each group’s performance by test. Again, large increases in knowledge as measured by test performance occurred in both groups as a result of interacting with algorithm visualizations. The VL group experienced a 25% improvement in average score after receiving the lecture, then improved another 28% following the AV interaction. The LV group experienced a 63% improvement after the visualization, while the following lecture provided an improvement of only 2%. Another interesting observation is that both groups eventually reached similar levels (72%) of performance. The LV group showed steady increases following the lecture and then the visualization. The VL group showed a significant increase resulting from visualization alone, to which the lecture did not add considerably. This appears to indicate that interactive hypermedia visualizations are more effective when prior knowledge is limited, and that a conventional teaching method such as a lecture does not appear to provide significant additional learning benefits. On the other hand, students with prior knowledge from conventional instruction also benefited from algorithm visualizations, which significantly increased their knowledge.

Statistical support for these conclusions is provided in Tables 4A, 4B and 4C. Table 4A shows the between-group statistical results. The post-test results show that the order in which lectures and visualizations are presented does not appear to make a difference, as both groups scored approximately the same (72% for the VL group and 72% for the LV group (F(1,19)=0.001, p<0.97). While these post-test results are not significantly in favor of either group, Table 4A shows that the mid-test performance results are significantly in favor of the group that interacted with the visualization first (VL group) compared to the group (LV group) receiving the lecture (F(1,19)=11.87, p<0.033).

Table 4A. Experiment IV Statistical Summary: Between Groups (Overall Performance)

<table>
<thead>
<tr>
<th></th>
<th>Pre-test</th>
<th>Mid-test</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>LV</td>
<td>19%</td>
<td>44%</td>
<td>72%</td>
</tr>
<tr>
<td>VL</td>
<td>7%</td>
<td>70%</td>
<td>72%</td>
</tr>
<tr>
<td>F(1,19)</td>
<td>1.78</td>
<td>5.3</td>
<td>0.001</td>
</tr>
<tr>
<td>p</td>
<td>0.198</td>
<td>0.033</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 4B shows that the score improvement from the pre-test to the mid-test favored the group receiving the visualization, which for this first phase of the experiment was the VL group (F(1,19)=26.89, p<0.0001). Additionally, the score improvement from the mid-test to the post-test also favored the group receiving the visualization, which for this second phase of the experiment was the LV group (F(1,19)=11.87, p<0.003). These results suggest that visualization results in a significant improvement in learning.

Table 4B. Experiment IV Statistical Summary: Between Groups (Improvement)

<table>
<thead>
<tr>
<th></th>
<th>Pre-to-Mid-Test Improvement</th>
<th>Mid-to-Post-Test Improvement</th>
<th>Overall Improvement (Pre-to-Post)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LV</td>
<td>25%</td>
<td>28%</td>
<td>53%</td>
</tr>
<tr>
<td>VL</td>
<td>63%</td>
<td>2%</td>
<td>65%</td>
</tr>
<tr>
<td>F(1,19)</td>
<td>26.89</td>
<td>11.87</td>
<td>1.16</td>
</tr>
<tr>
<td>p</td>
<td>0.00001</td>
<td>0.003</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Specific test-to-test improvements for the two groups are shown in Table 4C. Here, the only result that did not yield statistical significance was the improvement in performance for the VL group, when they went from having interacted with the visualization to receiving the lecture. The lecture added only 2% to the overall performance. But this effect is not statistically significant, indicating that the visualization prior to the lecture must have been the primary cause of overall improvement. In all other cases, both the lecture and the visualization resulted in significant improvements in knowledge.
Table 4C. Experiment IV Statistical Summary: Within Group (Improvement)

<table>
<thead>
<tr>
<th></th>
<th>pre-to-mid</th>
<th>mid-to-post</th>
<th>pre-to-post</th>
</tr>
</thead>
<tbody>
<tr>
<td>LV</td>
<td>F(1,19) 6.35</td>
<td>5.037</td>
<td>21.62</td>
</tr>
<tr>
<td></td>
<td>p &lt; 0.023</td>
<td>p &lt; 0.039</td>
<td>p &lt; 0.0002</td>
</tr>
<tr>
<td>VL</td>
<td>F(1,19) 43.97</td>
<td>0.016</td>
<td>73.44</td>
</tr>
<tr>
<td></td>
<td>p &lt; 0.00002</td>
<td>p &lt; 0.899</td>
<td>p &lt; 0.0000004</td>
</tr>
</tbody>
</table>

These results are summarized as box plots in Figure 4C. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. The distribution of scores is interesting. The distribution in the LV group appears to be similar in each of the three tests, with quartiles that are approximately equidistant from a well-centered mean. The distributions are not as uniform across the tests in the VL group. The pre-test is tightly clustered and positively skewed. The mid-test following the visualization indicates a wider distribution of scores across a larger range, which the lecture condition appears to tighten up. This seems to indicate that individuals may not have uniformly benefited from the visualization to the same extent, and that this dispersion was somewhat remedied by the lecture which presumably benefited those who did not significantly gain from the visualization. These individual differences are masked by the 2% overall improvement of the VL group from the lecture. This argues for hypermedia algorithm visualizations supplementing, rather than replacing, traditional instruction.

Figure 4C. Experiment IV Box Plots

3.4.5 Discussion

This experiment shed interesting insights on the three hypotheses that we set out to investigate. First, the results support the hypothesis that learning by visualization is more effective than learning by lecture alone. The mid-test results captured the improvement caused by visualization in the VL group and lecture in the LV group. The
results of this first phase are statistically significant (F(1,19)=5.3, p<0.03).
Second, these results suggest that the combination of learning by visualization and lecture leads to improvements over learning by visualization or lecture alone. Both groups improved after both phases were completed. The improvement observed in adding the visualization to the LV group was large and statistically significant (F(1,19)=5.037, p<0.039). However, the improvement observed in adding the lecture to the VL group was only 2% and not significant (F(1,19)=0.016, p<0.89).

Third, as to the impact of the presentation order in the final outcome, these results suggest that order does not matter. After completing both phases, the two groups performed at about the same level (~72%, F(1,19)=0.001, p<0.97).

Naturally, there are factors that could have influenced these results. The most obvious one is the quality of the lecturer. To address this issue, we requested the services of a highly rated (by students) professor who had taught introductory computer science courses several times. Dr. Dean Hendrix is known to be an excellent lecturer. We developed the experiment so that all the participants would attend a single lecture, to avoid the possibility that one lecture session could have covered the material in a different way than another. We also chose to have a live lecture instead of a videotaped one, to allow teacher-student interaction more typical of a classroom environment.

Another factor we did not control for was that some students might have read additional textual materials between the phases of the experiment. To reduce this possibility, we intentionally used the SelectionSort algorithm which was not mentioned in the course textbook. MergeSort was however covered in the textbook. We asked the students how much time they spent reading the text, if at all. Only four indicated that they had read the text, and the average time indicated was 10 minutes. We did not detect any significant differences in performance between the algorithm described in the textbook and the algorithm not covered in the textbook.

3.5. Experiment V
The goal of this experiment was to compare the effectiveness of learning from hypermedia algorithm visualizations in the style of HalVis to learning from a typical extant algorithm animation. The algorithm we used was Dijkstra’s Shortest Path algorithm. It is conceptually difficult, and is different in style (a graph algorithm) from all algorithms used in the previous four experiments (sorting algorithms). Participants completed all phases of this experiment in one day: a pre-test, computer interaction, and a post-test.

3.5.1 Subjects
This experiment involved 40 undergraduate computer science students enrolled in a third year algorithm analysis course at Auburn University. Like previous experiments, participants were ranked based on their course performance up through mid-term grades, GPA, and ACT/SAT scores, and assigned to two matched groups: a “Tango” group and a “HalVis” group. Students were given extra credit for participating. Twenty students each in the Tango group and the HalVis group completed the experiment.

3.5.2 Materials
Tango Algorithm Animation: One of the most mature and widely available algorithm animation platforms is the Tango software suite developed by Dr. John Stasko [Stasko 1990, Stasko 1997], publicly available from Georgia Institute of Technology at ftp.cc.gatech.edu:/pub/people/stasko. The Tango software distribution executes on Windows95 systems and includes a library of animated algorithms. Three researchers in our group carefully examined eight animations of the Shortest Path algorithm available in this distribution, and selected one that appeared to be the most complete, easiest to understand, and which most closely matched the features of the HalVis system (i.e., use of multiple representations, contextual descriptions and animated pseudocode).

Hypermedia Algorithm Visualization: A visualization for the Shortest Path algorithm was built and provided in HalVis.

Handout: The Tango group received a supplement to help them learn the Shortest Path algorithm, which consisted of a 5 page extract from their textbook [Weiss 1992]. This was done to simulate the conditions under which Tango-style animations were previously experimentally evaluated [Lawrence et al. 1994], when the visualization groups received textual supplements in addition to the visualization.
Test Questions: A pre-test/post-test combination measured individual learning performance with questions that probed conceptual and procedural knowledge about the algorithm. Students were tested on their ability to recognize and reorder pseudocode descriptions of algorithms, mentally simulate algorithmic operations, and predict resulting data structure changes. The pre-test measured prior knowledge about the algorithms and the post-test results measured changes resulting from the experimental conditions.

3.5.3 Procedure

As with the previous experiments, we timed the experiment to precede the course lectures that covered the subject of graph algorithms. Towards the end of the quarter, participants were asked to complete a pre-test that measured their prior knowledge about the Shortest Path algorithm. In addition to providing a baseline against which to compare subsequent changes, the pre-test results also helped us verify that the two groups were evenly balanced.

In the following week, both groups met in the same public computer laboratory on campus, but at different times. Both groups received a brief, navigation-only orientation to the software they were to use, then were assigned to a computer and instructed to interact with the visualization until they felt they understood the algorithm. The computers were Pentium-class systems with 15 inch color monitors.

Members of the HalVis group were not given any text material to study, nor had they been exposed to the algorithm earlier in class. There was no time limit for either group, so when each subject indicated he/she was done, he/she was given a post-test to measure knowledge improvement. No student in the HalVis group took more than 90 minutes for the entire experiment.

Members of the Tango group received an extract from their textbook describing the Shortest Path algorithm and were assigned to a computer to interact with the animation. They were not provided with any other information, nor had they been exposed to these algorithms during class lectures. When they indicated they understood the material, they were given a post-test and allowed to leave. No student in the text group took more than 60 minutes for the entire experiment.

3.5.4 Results

Examining the results shown in Figure 5A below, we see that both groups were relatively unfamiliar with the algorithm based on the pre-test averages (23% for the Tango group and 22% for the HalVis group). The post-test results show that the HalVis group’s scores improved to 89% while the Tango group improved to 71% (F(1,37)=12.75, p<0.001). The statistical results are summarized in Table 5A.

The results are also summarized as box plots in Figure 5B. The box indicates the range of entries in the 25th through 75th quartile, and the lines extending to the left and right show the range of scores for the entire group. The thick vertical line in the box indicates the mean, and the thin line represents the median value for the group. The distribution of scores is interesting. Generally, the HalVis pre-test score distribution is tight and normal looking, but there are two outliers (shown as black dots) that scored very well, indicating prior knowledge of the Shortest Path algorithm. The distribution of the HalVis group’s post-test scores indicates a tighter clustering, with one outlier at 69% (shown as a black dot), compared to the post-test score distribution of the Tango group. The post-test score distribution of the Tango group also shows (as a black dot) the presence of one outlier who scored extremely poorly.

<table>
<thead>
<tr>
<th>Statistical Summary</th>
<th>Pre-test</th>
<th>Post-test</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tango</td>
<td>23%</td>
<td>71%</td>
<td>48%</td>
</tr>
<tr>
<td>HalVis</td>
<td>22%</td>
<td>89%</td>
<td>68%</td>
</tr>
<tr>
<td>F(1,24)</td>
<td>0.01</td>
<td>12.75</td>
<td>4.79</td>
</tr>
<tr>
<td>p</td>
<td>p&lt;0.91</td>
<td>p&lt;0.001</td>
<td>p&lt;0.035</td>
</tr>
</tbody>
</table>

Table 5A. Experiment V Statistical Summary
Discussion

This experiment compared our HalVis algorithm visualization framework with an animation generated from a popular algorithm animation package. We chose Tango since Tango and its predecessors form a set of algorithm animations that have not only been extensively described in the literature [Stasko 1990, Stasko 1997], but also have been the subjects of significant experimental analyses reported in the literature [Byrne et al. 1996; Kehoe & Stasko 1996; Lawrence et al. 1994; Stasko 1997; Stasko et al. 1993]. The Tango animation was well-paced, showed good use of color to highlight algorithm actions, included a brief textual introduction, contained contextual explanations and provided the student with the algorithm’s pseudocode, whose lines were highlighted synchronously as the animation proceeded. We supplemented this animation with pages describing the algorithm from the course textbook [Weiss 1993] in order to provide the student with as much information about the algorithm as possible in a standalone setting, and to replicate as closely as possible the conditions of algorithm animation experiments reported by other researchers. The results indicate that our framework for hypermedia algorithm visualization design is more effective than an algorithm animation representative of current approaches.

How did our Tango group compare to previous experiments reported in the literature that used Tango under similar circumstances? One experiment reported in [Lawrence et al. 1994] compared groups using the Tango
animation system in conjunction with a lecture and active or passive laboratory assignments to learn Kruskal’s Spanning Tree algorithm. In their study, one group’s conditions closely matched that of our experiment: the group that received a prepared lecture (roughly corresponding with our group that received text) and a Tango/Polka animation that contained contextual descriptions but did not permit data modification (passive laboratory). The comparison is shown in Table 5B. While there are many factors that render an exact comparison impossible, the general results appear to suggest that our experimental group using Tango performed comparably with their corresponding experimental group.

<table>
<thead>
<tr>
<th>Comparison Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Lawrence et al, 1994</td>
</tr>
<tr>
<td>HalVis</td>
</tr>
<tr>
<td>Tango</td>
</tr>
</tbody>
</table>

Table 5B. Comparison of Results with [Lawrence et al. 1994]

As with the other experiments, there are several factors potentially influencing the results. The high scores of the HalVis group in the post-test indicates a possible ceiling effect that might have suppressed a greater separation between the groups. Another factor could be the quality of the Tango animation. We attempted to address this by selecting the best and most comparable animation from the eight supplied with the Tango distribution files. We believe that we chose a representative animation. Nevertheless, it is possible that a different Tango animation might have led to different results for the Tango group. A similar argument could be made for the text (photocopied pages from the course textbook) that was provided to the Tango group. The textbook has been used for several years at Auburn, and it is considered to be a good choice for an algorithms course.

4. General Discussion
Our experiments were designed to test the effectiveness of a novel framework for hypermedia algorithm visualizations, a framework that embeds animations in a context and knowledge providing hypermedia environment, against traditional methods of instruction. Comparisons with learning from a textbook, learning by reading carefully constructed textual explanations and solving problems, learning from lectures, and learning by interacting with an algorithm animation representative of extant research on the topic, all indicated the significant advantages of the HalVis framework from a self-directed learning perspective. Results from four of the five experiments were statistically significant. Significant effects were found for different algorithms and undergraduate students at different levels. The one experiment in which both the experimental and control groups performed at the same level served to illustrate that a hypermedia visualization can be as effective as learning from a well-crafted mixed mode explanation (text + diagrams) combined with problem solving. The following summarizes conclusions from these five experiments:

- Advanced as well as novice students perform better in answering conceptual and procedural questions about fundamental algorithms after interacting with hypermedia algorithm visualizations than after studying explanations found in typical textbooks on algorithms.
- Hypermedia algorithm visualizations appear to be as effective a learning aid for novice students to learn about algorithms as learning from carefully crafted textual and diagrammatic explanations combined with solving a set of problems.
- Novice students gain more knowledge after interacting with hypermedia algorithm visualizations than after hearing a typical classroom lecture. Furthermore, lecture and visualizations supplementing each other provides the best learning scenario, and the order of presentation does not seem to influence extent of learning.
- It appears that interactive hypermedia algorithm visualizations are more effective when prior knowledge is limited. However, students with prior knowledge from conventional instruction also derive a significant learning benefit from algorithm visualizations.
- Individual differences in learning from algorithm visualizations exist. These differences may be compensated by the use of multiple modes of instruction. This argues for hypermedia algorithm visualizations supplementing, rather than replacing, traditional instructional methods.
Finally, the framework for algorithm visualization design that HalVis exemplifies appears to be much more effective than previous algorithm animation designs.

5. Conclusion
The general conclusion is that interactive hypermedia algorithm visualizations modeled after the HalVis framework (a system in which animations are embedded within a knowledge and context providing hypermedia environment) can provide significant benefits to learners as an educational medium for self-directed and self-paced learning, either by itself or even more so in combination with other instructional media.

There are a number of possible reasons for this. First, in comparison with previous animation systems that only presented animations with some textual feedback, HalVis allows the student to learn incrementally by starting from a real world analogy and transitioning to the algorithm itself. Second, the hypermedia structure allows a student access to fundamental building blocks of algorithmic knowledge in-context and on-demand. Third, a learning objective-based design approach and the hypermedia structure surrounding animations have allowed us to divide dynamic information into manageable and meaningful pieces, and present each piece using animation chunks. This makes it easier for students to pause and reflect, repeat, or access other relevant information through hyperlinks while watching animations. Furthermore, animation chunks are presented in synchrony with other representations in other media. These novel features, we believe, result in the dynamic information being conveyed better in context, and therefore in a more comprehensible fashion. Fourth, rather than providing just one view of an animation as has been the typical approach, HalVis presents three kinds of animations (analogical, micro-level and macro-level), so that the macro behavior is seen after the micro behavior is seen and understood, both following an analogical introduction to the algorithm. Fifth, our framework allows students to actively engage themselves in the visualization by changing data inputs, making performance predictions, and reflecting on questions that pop up in context, all contributing to better learning.

Future research plans include a series of ablation experiments designed to measure the differential contributions to learning of these various features and subsequent refinements to the HalVis framework. Latest results from this research program are always available at http://www.eng.auburn.edu/cse/research/vi3rg/vi3rg.html.

Acknowledgements
This research is supported by the National Science Foundation under contracts CDA-9616513 and REC-9815016. The experiments reported herein could not have been carried out without the participation of volunteers from the following courses: CSE 220-Fundamentals of Computer Science and CSE 360-Fundamental Algorithm Design and Analysis.

References
Appendix A

Sorting Text

(Descriptive Materials Created and Used As a Handout for Experiment III)
Sorting Algorithms

Learning Objectives:

- Understand the basic concepts of sorting algorithms
- Learn the construction and behavior of specific sorting algorithms
- Predict how sort algorithms will operate on a given data set
- Be able to modify and find errors in sorting algorithms

5.1. Introduction

Sorting is one of the more interesting topics in computer science and in the study of algorithms, not only because sorting is a common and useful problem but also because there are many different ways one can sort a list. The various approaches are interesting to study and understand. They represent different ways of solving a similar problem. Some approaches are easier to understand than others, some are more take less time, some use less space and some are better in situations where the order of the lists to be sorted is known in advance.

The input to a sorting problem is an unordered list of elements, typically numbers or letters. The task is to produce the list in a particular order, either ascending or descending, based on each element’s lexicographic value. A dictionary and a phone book are examples of alphabetically ordered lists in ascending sequence. Examples of numerical sequences are shown below, depicted as both horizontal and vertical lists of numbers.

<table>
<thead>
<tr>
<th>Unordered</th>
<th>Ascending Order</th>
<th>Descending Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 4 8 10 1</td>
<td>1 4 6 8 10</td>
<td>10 8 6 4 1</td>
</tr>
</tbody>
</table>

6. Unordered

<table>
<thead>
<tr>
<th>Ascending Order</th>
<th>Descending Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 6 8 10</td>
<td>10 8 6 4 1</td>
</tr>
</tbody>
</table>
**Bubble Sort**

The bubble sort algorithm uses a simple scheme. Each iteration puts the smallest unsorted element in its correct place, changing places with other elements in the list. The first iteration puts the smallest element in the first position. Starting with the last element, we compare successive pairs of elements, swapping whenever the bottom element of the pair is smaller than the one above it. In this way, the smallest element “bubbles” up to the top or front of the list. The next iteration puts the smallest element in the unsorted part of the list into the second position, using the same technique.

The figures below walk through sorting a 5-element list. Each row represents a comparison of the items in bold print, and arrows are used to show items that were exchanged or not. Each pass is indicated separately for easier reading. Note that in addition to putting one element in its proper place, each iteration causes some intermediate changes in the order of the other elements also.

The first traversal puts the value 1 into place at the head of the list, making 4 comparisons and 4 swaps in the process. Note that this first traversal does not guarantee the entire list is sorted—it only ensures that the first element is.

**6.1.1 First Pass**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

1\textsuperscript{st} comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1\textsuperscript{st} comparison</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

2\textsuperscript{nd} comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2\textsuperscript{nd} comparison</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

3\textsuperscript{rd} comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3\textsuperscript{rd} comparison</td>
<td>10</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

4\textsuperscript{th} comparison

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4\textsuperscript{th} comparison</td>
<td>1</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
The second traversal will bring the second largest element to its proper resting place. Notice that only 3 comparison operations were needed. A fourth was not required since the first element is already in position. Also notice that only 2 swap operations were needed, since the elements in the second comparison were not out of order as a pair.

### 7.1.1 Second Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8. Before</td>
<td></td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>1st comparison</td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>2nd comparison</td>
<td></td>
<td>1</td>
<td>10</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3rd comparison</td>
<td></td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

The third traversal will bring the third largest element to its proper resting place, requiring 2 comparison operations and 2 swap operations.

### 8.1.1 Third Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9. Before</td>
<td></td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>1st comparison</td>
<td></td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>2nd comparison</td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

The fourth traversal brings the fourth element into position, making 1 comparison and 1 swap.

### 9.1.1 Fourth Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10. Before</td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1st comparison</td>
<td></td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Before writing down the algorithm in more detail, it should be pointed out that the second traversal need not extend to the first element, since by the time the second traversal starts, the first position in the list already contains its rightful tenant: the smallest value in the list. Similarly, the third traversal need not consider the first 2 elements, etc. This leads to an algorithm that carries out N-1 such traversals (why not N?) to produce the final list. On each pass or traversal, the algorithm need only compare N-1 elements in its first traversal, N-2 elements in the second, N-3 in its third, and so on. Thus the bubble sort algorithm involves two nested loops. The outer loop controls the number of (successively smaller) passes or traversals through the array. The inner loop controls the pairs of adjacent entries being compared.

```plaintext
for x = N-1 downto 1
    for y = N downto N-x+1
        if A[y] < A[y-1]
            swap the values in A[y] and A[y-1]
        endif
    endfor
endfor
```
Selection Sort

The basic idea of selection sort is to make repeated selections from a list of values, moving the selected value into its proper position in the list. On the first pass, find the smallest number in the list and exchange it with the one in the first position (A[1]). On the second pass, find the smallest number from the values in positions 2 on down and exchange it with A[2]. On the third pass, find and place the smallest remaining value into the third position, and so on until there are no more values in the unsorted portion of the list. Each pass puts one element into proper order, and reduces by one the number of elements in the unsorted portion.

The figures below walk through sorting a 5-element list. Each row is labeled as a comparison between items in bold print or a swap, with arrows showing items that were exchanged. Each pass is indicated separately for easier reading. Note that in addition to putting one element in its proper place, each iteration causes some intermediate changes in the order of the other elements also.

The first pass involves comparing each of the values to find the smallest, keeping track of its position (call it MIN) until the last value has been considered. Then that value indicated by MIN is swapped with the item in position 1. Here, 4 comparisons are made, and one swap. Note that this first traversal does not guarantee the entire list is sorted—it only ensures that the first element is. The first pass produces:

### 10.1.1 First Pass

<table>
<thead>
<tr>
<th>11. Before</th>
<th>10</th>
<th>4</th>
<th>8</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st comparison</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2nd comparison</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3rd comparison</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4th comparison</td>
<td>10</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Swap</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>
The second pass can ignore the value in the first position and begin comparing values in positions 2 through 5. Notice that no swap is needed, since the value “4” was already in its proper position (in A[2]), yet 3 comparisons are made.

### 11.1.1 Second Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12. Before</td>
<td>1 (in place)</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1st comparison</td>
<td>1 (in place)</td>
<td>4</td>
<td>min</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2nd comparison</td>
<td>1 (in place)</td>
<td>4</td>
<td>min</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>3rd comparison</td>
<td>1 (in place)</td>
<td>4</td>
<td>min</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Swap</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>

The third pass will locate the smallest value in positions 3-5 and place it in position 3. This requires 2 comparison operations and one swap.

### 12.1.1 Third Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13. Before</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>1st comparison</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>8</td>
<td>6</td>
<td>min</td>
</tr>
<tr>
<td>2nd comparison</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>8</td>
<td>6</td>
<td>min</td>
</tr>
<tr>
<td>Swap</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>6 (in place)</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
The fourth and final pass yields the sorted list, using one comparison but not needing any swap operations:

### 13.1.1 Fourth Pass

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14. Before</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>6 (in place)</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>1&quot; comparison</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>6 (in place)</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Swap</td>
<td>1 (in place)</td>
<td>4 (in place)</td>
<td>6 (in place)</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Before writing the algorithm for this sorting procedure, note the following:
1. If the array is of length N, then it takes N-1 steps to put it into order.
2. We must be able to find the smallest number. Numbers that are equal are not considered smaller than each other.
3. We need to exchange appropriate array components that are out of order (inverted)

```plaintext
for j = 1 to N-1
    for j = i to N
        MIN = the index of the smallest value encountered
    endfor
    swap the values in A[J] and A[MIN]
```
Questions

Use the data set below to consider answers for questions 1-5:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15. Initial Values</td>
<td>8</td>
<td>4</td>
<td>26</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

1. Which element will move into the leftmost position on the first pass?

2. How many comparisons will be needed to complete the first pass?

3. How many swaps will occur in the first pass?

4. How many passes will it take until the remaining values are in place?

5. Write the order of the elements as they would appear at the completion of the second pass

6. The lines to the Bubble Sort algorithm are out of sequence below. From memory, try to number them to represent the correct sequence

<table>
<thead>
<tr>
<th>Line #</th>
<th>Algorithm Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>for i = N-1 downto 1</td>
</tr>
<tr>
<td></td>
<td>compare A[J] with A[J+1], exchanging if necessary</td>
</tr>
<tr>
<td></td>
<td>for j = N downto N-i</td>
</tr>
<tr>
<td></td>
<td>endfor</td>
</tr>
</tbody>
</table>

7. Do the same as problem #5 for the Selection Sort algorithm below:

<table>
<thead>
<tr>
<th>Line #</th>
<th>Algorithm Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>for j = i to N</td>
</tr>
<tr>
<td></td>
<td>endfor</td>
</tr>
<tr>
<td></td>
<td>exchange the values of A[J] and A[MIN]</td>
</tr>
<tr>
<td></td>
<td>MIN = the index of the smallest value encountered</td>
</tr>
<tr>
<td></td>
<td>for j = 1 to N-1</td>
</tr>
<tr>
<td></td>
<td>endfor</td>
</tr>
</tbody>
</table>
Appendix B

PostTest for Experiment III

(Knowledge improvement measurement tool)
Knowledge Improvement Survey

NOTE: Your answers to these questions WILL NOT affect your grade in any way; they merely help us understand how effective the experimental material presented to you about algorithms was. Please answer each question to the best of your ability.

1. Show the order of elements in ARRAY after the first pass of an ascending Selection Sort algorithm:

<table>
<thead>
<tr>
<th>ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
</tr>
<tr>
<td>After 1 pass</td>
</tr>
</tbody>
</table>

2. How many swap operations would occur in the problem described above (first pass of ascending Selection Sort)?

3. In a sentence or two, describe the basic behavior of the Selection Sort algorithm

4. Show the order of elements in ARRAY after the first pass of an ascending Bubble Sort algorithm:

<table>
<thead>
<tr>
<th>ARRAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
</tr>
<tr>
<td>After 1 pass</td>
</tr>
</tbody>
</table>

5. How many swap operations would occur in the problem described above (first pass of ascending Bubble Sort)?

6. In a sentence or two, describe the basic behavior of the Bubble Sort algorithm
7. The pseudocode to the right implements which popular sort algorithm?

8a. Does the pseudocode above re-order the elements in ARRAY into descending order?

8b. If not, make pen and ink changes to alter the pseudocode above to produce a descending sequence

9. Using the algorithm shown in problem 7 above, how many comparisons and swaps would be made to produce a sorted sequence given the input [1,2,3,4]

<table>
<thead>
<tr>
<th>Comparisons</th>
<th>Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. Using the modified algorithm below, how many comparisons and swaps would be needed to produce a sorted sequence given the input [1,2,3,4]?

```
for x = N-1 downto 1
  for y = N downto N-x+1
    if ARRAY[y] < ARRAY[y-1]
      swap ARRAY[y] and ARRAY[y-1]
    endif
  endfor
endfor

for x = N-1 downto 1
  SORTED = 1
  for y = N downto N-x+1
    if ARRAY[y] < ARRAY[y-1]
      SORTED = 0
      swap ARRAY[y] and ARRAY[y-1]
    endif
  endfor
  if SORTED = 1
    exit
  endif
endfor
```
11. This pseudocode to the right implements which popular sort algorithm?

12a. Does the pseudocode above re-order the elements in ARRAY into descending order?

12b. If not, make pen and ink changes to alter the pseudocode above to produce a descending sequence

13. Using the algorithm above, how many comparisons and swaps would be made to produce a sorted sequence given the input [1,2,3,4]

<table>
<thead>
<tr>
<th>Comparisons</th>
<th>Swaps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14. Using the modified algorithm below, how many comparisons and swaps would be needed to produce a sorted sequence given [1,2,3,4]?

for x = 1 to N-1
  CHOICE = x
  for y = x+1 to N
    if ARRAY[y] > ARRAY[CHOICE]
      CHOICE = y
    endif
  endfor
  if CHOICE not = x
    swap ARRAY[CHOICE] and ARRAY[x]
  endif
endfor
15. Circle the item(s) below that are true:
   
   A  On average, Bubble Sort makes fewer swaps than Selection Sort
   B  On average, Bubble Sort makes more swaps than Selection Sort
   C  On average, Bubble Sort makes the same number of swaps as Selection Sort

16. After $k$ (some arbitrary number) passes, the first $k$ items are always in the proper place for:
   
   A  Bubble Sort
   B  Selection Sort
   C  Both Bubble Sort and Selection Sort
   D  Neither Bubble Sort nor Selection Sort

17. Organize the values [1,2,3,4,5] into an input sequence that causes the standard Bubble Sort algorithm to make fewer swap operations than the Selection Sort algorithm.

   Answer: 
   
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>