Visualizing Algorithm Analysis Topics

Mohammed Fawzi Seddik Farghally

Dissertation submitted
to the Department of Computer Science,
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Clifford A. Shaffer, Chair
Christopher L. North
Donald S. McCrickard
Jeremy V. Ernst
Susan H. Rodger

October 17, 2016
Blacksburg, Virginia

Keywords: Algorithm Analysis Visualizations, Visual Proofs, Student Engagement, Logged Data Analysis, Educational Data Mining, Online Learning Environments, Performance Evaluation, Concept Inventory

Visualizing Algorithm Analysis Topics

Mohammed Fawzi Seddik Farghally

(ABSTRACT)

Data Structures and Algorithms (DSA) courses are critical for any computer science curriculum. DSA courses emphasize concepts related to procedural dynamics and Algorithm Analysis (AA). These concepts are hard for students to grasp when conveyed using traditional textbook material relying on text and static images. Algorithm Visualizations (AVs) emerged as a technique for conveying DSA concepts using interactive visual representations. Historically, AVs have dealt with portraying algorithm dynamics, and the AV developer community has decades of successful experience with this. But there exist few visualizations to present algorithm analysis concepts. This content is typically still conveyed using text and static images.

We have devised an approach that we term Algorithm Analysis Visualizations (AAVs), capable of conveying AA concepts visually. In AAVs, analysis is presented as a series of slides where each statement of the explanation is connected to visuals that support the sentence. We developed a pool of AAVs targeting the basic concepts of AA. We also developed AAVs for basic sorting algorithms, providing a concrete depiction about how the running time analysis of these algorithms can be calculated.

To evaluate AAVs, we conducted a quasi-experiment across two offerings of CS3114 at Virginia Tech. By analyzing OpenDSA student interaction logs, we found that intervention group students spent significantly more time viewing the material as compared to control group students who used traditional textual content. Intervention group students gave positive feedback regarding the usefulness of AAVs to help them understand the AA concepts presented in the course. In addition, intervention group students demonstrated better performance than control group students on the AA part of the final exam.

The final exam taken by both the control and intervention groups was based on a pilot version of the Algorithm Analysis Concept Inventory (AACI) that was developed to target fundamental AA concepts and probe students' misconceptions about these concepts. The pilot AACI was developed using a Delphi process involving a group of DSA instructors, and was shown to be a valid and reliable instrument to gauge students' understanding of the basic AA topics.

This work received support from the VT-MENA program of Egypt, and the US National Science Foundation under Grant Numbers DUE-0836940, DUE-0937863, and DUE-0840719.

Visualizing Algorithm Analysis Topics

Mohammed Fawzi Seddik Farghally

(GENERAL AUDIENCE ABSTRACT)

Data Structures and Algorithms (DSA) courses are critical for any computer science curriculum. DSA courses emphasize concepts related to how an algorithm works and the time and space needed by the algorithm, also known as Algorithm Analysis (AA). These concepts are hard for students to grasp when conveyed using traditional textbook material relying on text and static images. Algorithm Visualizations (AVs) emerged as a technique for conveying DSA concepts using interactive visual representations. Historically, AVs have dealt with portraying how an algorithm works, and the AV developer community has decades of successful experience with this. But there exist few visualizations to present concepts related to algorithm efficiency. This content is typically still conveyed using text and static images.

We have devised an approach that we term Algorithm Analysis Visualizations (AAVs), capable of conveying efficiency analysis concepts visually. In AAVs, analysis is presented as a series of slides where each statement of the explanation is connected to visuals that support the sentence.

AAVs were tested through a study across two offerings of CS3114 at Virginia Tech. We found that students using AAVs spent significantly more time viewing the material as compared to students who used traditional textual content. Students gave positive feedback regarding the usefulness of AAVs to help them understand the efficiency concepts presented in the course. In addition, students using AAVs demonstrated better performance than students using text on the efficiency part of the final exam.

The final exam was based on a pilot version of the Algorithm Analysis Concept Inventory (AACI) that was developed to target fundamental efficiency concepts and probe students' misconceptions about these concepts. The pilot AACI was developed through a decision making technique involving a group of DSA instructors, and was shown to be a valid and reliable instrument to gauge students' understanding of the basic efficiency topics.

Dedication

This dissertation is dedicated to my loving mother, Nagwa. Her support, encouragement, and constant love have sustained me throughout my life.

I also dedicate this dissertation to my wife, Rehab. I give my deepest expression of love and appreciation for the encouragement that she gave and the sacrifices she made during my graduate program.

This dissertation is also dedicated to my sister, Mayada, my beloved nephews, Omar and Malek, and my niece, Razan.

Also I dedicate this dissertation to the soul of my father, Fawzi Farghally, for his unlimited support till his last moment.

Finally, I dedicate this dissertation to my lovely little angels, Basel and Lamar.

Acknowledgments

First of all, all thanks due to ALLAH the almighty. May His peace and blessings be upon his prophet, Mohammed, for granting me the chance to successfully complete my PhD.

My heartfelt gratitude to my advisor, Professor Clifford A. Shaffer, for his inspiration, enthusiasm, invaluable support, and patience. He has taught me many things, and this work would not have been possible without his encouragement and support.

I would like to thank the OpenDSA research group, Kyu Han Koh, Eric Fouh, Hossam Shahin, and Sally Hamouda, who helped me through out my work. I would like to extend my thanks to all the instructors who used OpenDSA in their classes, and to all the students who used OpenDSA.

I wish to express my sincere gratitude to VT-MENA program director, Professor Sedki Riad, for his endless support and invaluable guidance. He has been and will always be a real father for us.

Once again, thanks to ALLAH almighty God for giving me the ability, mindset, and perseverance to be where I am now.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Data structures and algorithms are fundamental topics considered critical in computer science education. Data structures and algorithms (DSA) courses include concepts related to algorithm dynamics (how an algorithm works), algorithm proof of correctness (the algorithm correctly performs its job), and algorithm analysis (algorithm efficiency). DSA concepts are traditionally hard for many students to grasp as they are abstract and mathematical in nature [10, 24, 69]. Accordingly, special attention must be given to the ways that these topics are presented to students by attempting to find better techniques that can improve the students' understanding in this area.

Algorithm dynamics concepts are important in any DSA course as this defines how an algorithm works. The focus is on the nature of the algorithm in terms of its execution steps and how each step updates the state of the underlying data structure. In 1981, Roland Baecker first included algorithm and data structure visualizations (AVs) into the curriculum through the "Sorting out Sorting" video. A good AV should include an illustration of the step-by-step execution of algorithms and how each step affects the state of the underlying data structure [78]. As the old adage states "A picture is worth a thousand words", there was an appealing belief that AVs will be pedagogically effective when used in courses [40, 62]. Two reasons for this belief were stated in [10]:

1. Abstract conceptual phenomena are notoriously difficult for learners to grasp since they have no physical manifestation. Visualizations can aid learners in constructing a mentally runnable simulation of dynamic processes by providing a concrete pictorial view of abstract concepts.
2. Computer algorithms are dynamic processes that evolve over time. Algorithm textbooks are filled with pictures associated with instructions to assist in the explanation. Visualizations go one step further. While static pictures can provide learners with the

> essence of how something looks, visualizations appear better able to explain a dynamic evolving process as they help viewers to track patterns and observe relationships in a display. Static images are not as so good in conveying this time evolution.

Baecker's work set the stage for future research on AVs. Now AV researchers know how to implement AVs and apply them in classrooms[78]. But, to our knowledge, all of the current existing AVs are concerned only with visualizing algorithm dynamics. They display the data structure in an initial state and then display state changes after the execution of each algorithm step. This gives students a better intuition about how an algorithm works. But what about understanding the algorithm's efficiency?

Separate from dynamics, algorithm analysis concepts are crucial in any DSA course. Students are expected to understand the nature of the algorithm in terms of resource requirements, such as processing time and storage to gauge the efficiency of an algorithm in different situations and problem instances. These measures can then be used to compare algorithms to pick the more efficient for a given task. It was stated in [24, 69] that analysis concepts are not fully comprehended by most of the students since these concepts are presented and justified by invoking sophisticated mathematical arguments. Current AVs provide little support for visualizing algorithm analysis concepts[78]. Hence the AV field is missing one of the most important concepts in any DSA course. Proof of correctness concepts are similar to algorithm analysis, as they both rely on a series of mathematical arguments. To our knowledge, there is nothing in the literature providing any innovation in presenting proof of correctness concepts to students. However, some attempts exist trying to visualize some NP-completeness proofs [13, 55]. In addition, the JFLAP system [72] provides some construction type proofs showing how to convert an NFA to DFA for automata courses.

The main focus in this dissertation is to develop and evaluate a new generation of visualizations that we name Algorithm Analysis Visualizations (AAVs), created specifically to illustrate algorithm analysis concepts. Designing such visualizations is a challenge, because unlike AVs * there is no dynamic content to be presented, and analytical material is much more abstract than that of algorithm dynamics as it focuses on mathematical arguments and manipulations. In order to be able to use AAVs in a DSA course as part of the curriculum, we have developed AAVs in a way that they can be easily embedded in an online eTextbook. Our choice was to use the OpenDSA eTextbook. The OpenDSA project at Virginia Tech [19, 20] is concerned with building a complete open-source infrastructure and body of methods with which to create electronic textbooks for DSA courses enhanced with various embedded artifacts, such as algorithm visualizations, exercises with automated assessment, and slideshows to improve the understanding and learning experience of students during the course. The major aim of OpenDSA is to allow instructors to create their own instances of complete interactive eTextbooks that integrate interactive artifacts with the prose content. OpenDSA contains a pool of AVs designed using the JavaScript Algorithm Visualization

*Throughout this dissertation, this term will be used to refer to algorithm visualizations that emphasize algorithm dynamics concepts.

JSAV framework [45, 46] to support various topics in undergraduate DSA courses such as sorting, hashing, linked lists, trees, and graphs. OpenDSA contains many AVs illustrating algorithm dynamics concepts, but prior to this study, there was little innovation in OpenDSA related to presenting analytical material visually. There were no available techniques in OpenDSA for presenting a visualization to illustrate the analysis of an algorithm. Accordingly, algorithm analysis concepts were presented using textual paragraphs including mathematical equations supported by static images. This is the same way that traditional printed textbooks present the same material. Similar to existing OpenDSA AVs, AAVs were implemented using the JSAV framework to be consistent with the available AVs, and to be easily embedded into the HTML-based lecturing material. JSAV is designed mainly to support algorithm dynamics concepts, however, there is support that can be leveraged to implement AAVs. By presenting OpenDSA analytical material using visualization, we believe that students will have a better learning opportunity, as they are provided a better understanding of the basic concepts of algorithm analysis and a better intuition about algorithm running-time proofs, which are hard to grasp when relying merely on textual discussion and static images.

Evaluating the pedagogical effectiveness of AAVs is another major challenge. It was stated in [18] that measuring student learning gains made with technological interventions is a difficult endeavor. Regarding AVs, it was stated in [40], that despite the intuitive appeal of visualizations, it has been slow to catch on in mainstream computer science education. The experimental studies designed to substantiate the educational effectiveness of AVs have shown mixed results.

Time-on-task measurement is a widely used method to analyze student behaviors [6, 7], assess student engagement in learning [27, 47], and measure student learning [71]. After analyzing OpenDSA student interaction logs from Fall 2014, we found that most students do not spend a reasonable amount of time studying algorithm analysis concepts that are presented in OpenDSA. Sometimes they skip it entirely to jump to other module parts containing exercises so that they can acquire credit. Given these results and the difficulty of evaluating the pedagogical effectiveness of AAVs based on student performance, we think that a good indicator for successful presentation may come from time evaluation [60]. In time evaluation, we can see if the time spent by students in the analytical material with the visualizations available is more than that spent without them. This can be a good indicator of student engagement. To do this, we have first to determine the time spent by students reading the analytical material for key OpenDSA modules before introducing the visualizations, and then determine the time again with the analysis material replaced with the analysis visualizations. This was done as a between-subject quasi-experiment across two semesters (one semester is the control group and the subsequent semester is the intervention group) with two offerings of a CS3-level course. Time evaluation may be an indication of student engagement, which is crucial in evaluating the pedagogical effectiveness of AVs as stated in [62].

Of course we cannot claim that the visualizations are pedagogically effective relying merely

on time evaluation. It was stated in [28] that student engagement is a necessary prerequisite for learning, but it is not sufficient by its own to show effective learning. Accordingly, we conducted an experiment to see the impact of the visualizations on student performance for algorithm analysis questions. As in time evaluation, the experiment ran through two semesters with two offerings of a CS3-level course. In one semester, algorithm analysis concepts were presented using traditional OpenDSA material, while in the second semester, algorithm analysis concepts were conveyed through AAVs. The difference in performance between the two groups was measured through post-tests.

A question worth asking: "how can we be sure that the post-test actually measures students' performance in algorithm analysis?" This question is hard to ignore as we should not assume that a set of questions measure algorithm analysis performance without proving this empirically before offering it as a post-test in our evaluation experiment. Accordingly, we decided to design the Algorithm Analysis Concept Inventory (AACI), and test its validity and reliability using the methods available in the literature. This was challenging, since we needed to build the CI from scratch. To our knowledge, no work in the literature has addressed the problem of defining important and difficult concepts related to algorithm analysis. In addition, few attempts are available identifying misconceptions related to algorithm efficiency [21, 67]. The developed CI was administered as a post-test to provide a clear after view of student learning with and without AAVs.

1.2 Research objectives

This study seeks to satisfy three main research aims:

1. First, we developed a pool of interactive visualizations to enhance the algorithm analysis material presented in OpenDSA modules.
2. Second, we designed an experiment to evaluate the educational effectiveness of the implemented visualizations when embedded into OpenDSA modules and used in the classroom to see if they actually improve the students' level of understanding in terms of their engagement and performance in the course.
3. Third, we built the pilot version of the algorithm analysis concept inventory and tested its validity and reliability to be used as the the post- test in my evaluation experiment.

1.3 Research Questions

The primary research questions for this study are:

1. Are students more engaged with AAVs than with the traditional algorithm analysis content available in OpenDSA?

2. Is the performance in a post-test of students using AAVs higher than students using traditional OpenDSA analysis material?
3. What feedback do students from the intervention group give regarding their experience with AAVs (based on a survey at the end of the semester)?
4. Is our AACI a valid and reliable measure for student performance in algorithm analysis?

1.4 Major Contributions

The key contributions of this study are:

1. A set of interactive visualizations capable of conveying algorithm analysis concepts visually that can be embedded in the curriculum as part of an online eTextbook.
2. A quasi-experiment and results analysis to gauge the effectiveness of AAVs in terms of student engagement, satisfaction, and performance.
3. A set of algorithm analysis misconceptions that are believed to be held by students after taking a CS3-level course.
4. A set of important and difficult concepts related to algorithm analysis identified using a Delphi process.
5. A set of items that constitute the algorithm analysis concept inventory that can be used to distinguish between students who think in accordance to common conceptions on algorithm analysis versus students who think in accordance to common misconceptions in algorithm analysis.

1.5 Dissertation Outline

In Chapter 2, we present some previous work related to applying visualizations in education. We also present previous work related to designing concept inventories and testing their validity and reliability. Chapter 3 presents the idea behind AAVs with some examples and the motivation behind it from both OpenDSA student interaction log analysis and student surveys. Chapter 4 presents our work in building the algorithm analysis concept inventory along with the work to test its reliability and validity. Chapter 5 presents the results from the experiment performed to evaluate the effectiveness of AAVs according to student engagement, satisfaction, and performance. Finally, Chapter 6 presents future research directions.

Chapter 2

Related Work

2.1 Visualization in Education and Visual Proofs

During the last three decades, several attempts had been made to design and implement various types of AV tools with interactive capabilities to help students better understand DSA concepts. For a good review of some of these tools, refer to [78]. Examples of some well known AV systems include: GAIGS [63] and SWAN [79], which are considered as early attempts of utilizing object oriented technologies in creating AVs. JHAVE [61] and ANIMAL [73] are among the first platform independent tools in the AV field, since they were implemented in Java. JFLAP [72] is another Java-based AV system which was mainly directed towards automata theory classes. TRAKLA2 [48] was inspirational in creating the first meaningful DSA exercises with automated assessment. Here the student is required to simulate the operation of a particular algorithm by changing the states of the underlying data structure according to the algorithm steps and is provided with an immediate feedback. Other attempts for building effective AV tools are JAWAA [1], ALVIE [14], and MA&DA [49].

All of the aforementioned AV tools focus merely on algorithm dynamics, trying to illustrate the procedural nature of algorithms visually in terms of state changes in the underlying data structure. To our knowledge, there is no support for visualizing general algorithm analysis concepts such as upper bounds, lower bounds, best, average, and worst cases, and the running-time complexity of basic algorithms, which are crucial concepts that are hard to comprehend by most of the students since they are normally presented using sophisticated mathematical arguments [24, 69].

There have been paper-based attempts to present proofs visually. Goodrich, Tamassia, and Goldwasser [25] presented some algorithm running time complexity proofs visually using geometric shapes. The task of calculating the asymptotic running time complexity of a particular algorithm is reduced to calculating the areas of a collection of these shapes as each shape is representing an amount of work done by the algorithm. Visual proofs

[8, 24, 30, 80, 85] is an approach that caught the attention of algorithms researchers and instructors since it augments or even replaces inductive arguments that many students find difficult. Goodrich and Tamassia [24] presented simple visual proofs for several core topics in DSA courses (summing linear terms, counting nodes in a binary tree, analyzing binary tree traversal, analyzing bottom-up heap construction, and rebalancing AVL trees via rotations) in an attempt to justify the potential of using visual alternatives for teaching algorithm analysis concepts. Thompson and Chadhuri [85] presented an alternative visual analysis of the Build-heap algorithm (as previously presented in [24]). Blaheta [8] presented a visual proof for amortized-linear resizable arrays by proving that doubling the capacity is the best strategy when resizing linear arrays. Sher [80] presented a visual proof of the average case running time of a list searching algorithm. Hammack and Lyons [30] provided a picture proof of the alternating series test using simple comparisons of areas of rectangles to establish convergence.

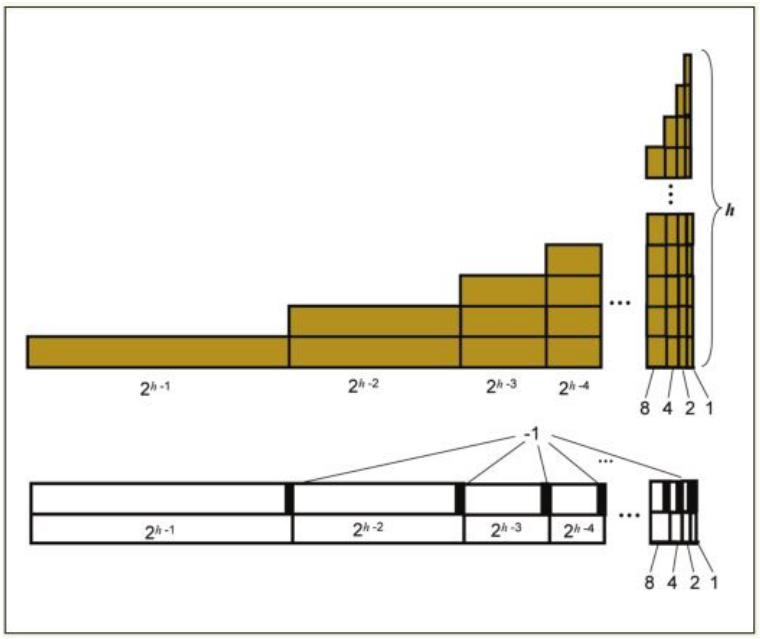


Figure 2.1: Build-heap worst-case running time Visual Proof, from [85]

In Figure 2.1, the Build-heap visual proof presented in [85] is shown. Most students demonstrate the misconception that the running time of the Build-heap procedure in the worst case is $O(n \log n)$, as the running time complexity of the heapify procedure in the worst case is $O(\log n)$ and the heapify procedure should be done for n nodes in the heap. This visual proof provide a visual intuition that the running time complexity of the Build-heap operation in the worst case is $O(n)$. The amount of work done at each heap level is translated to a set of rectangles in which the number of rectangles determines how many levels the nodes need to be pushed down the heap to maintain the heap property. The width of the rectangle represents the number of nodes in a particular heap level. For example,

at level 0 of the heap, there is only the root node, so we have h (the height of the heap) rectangles, each has one unit width to represent the work done at this level in the worst case (the rightmost set of rectangles in the upper shape). All the rectangles are unit height, so the surface area of the rectangle indicates the amount of work done. The upper shape is then rearranged in a way relying on the mathematical fact that for any positive integer k, $2^k - 1 = 2^{k-1} + 2^{k-2} + ... + 8 + 4 + 2 + 1$. Finally, the total running time of the Build-heap procedure is transformed to be the problem of calculating the total surface area of the lower shape in the Figure which is $= 2 \times [2^{k-1} + 2^{k-2} + ... + 8 + 4 + 2 + 1] = 2 \times [2^h - 1] = 2^{h+1} - 2$. Since $h = \lceil \log n \rceil$, the total surface area of the shape is $O(n)$.

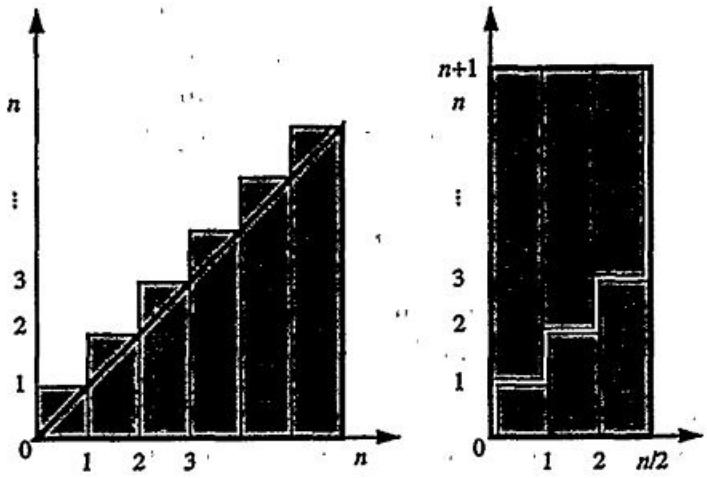


Figure 2.2: Summation from 1 to n Visual Proof, from [24]

In Figure 2.2, a visual proof to find the closed form solution of the summation $\sum_{i=1}^{n} i$ presented in [24] is shown. This summation is used to model the worst case running time of Insertionsort. Here, we have n rectangles, and each one represents a term in the summation. The height of the rectangle is the value of the term it represents. All of the rectangles are of unit width. The closed form of the summation is found by calculating the surface area of the resulting shape on the left, which is simply the area of the big triangle (base n, height n) plus the areas of the small n rectangles (base 1, height 1). We have the total surface area equal to $\frac{n(n+1)}{2}$. This is exactly the closed form solution of the summation. The shape on the right shows a different approach to calculate the surface area of the shape in the special case when n is even.

Both of the previously described visual proofs share a common principle that can be used to develop further visual proofs in a similar manner. Graphical primitives are leveraged to represent the amount of work required for each step, and then the total running time can be viewed as the total surface area of the resulting shape. We name this principle *"Area to Cost"* for future reference. The average-case list searching visual proof [80] also applied this

principle using rectangles. Here, each rectangle's width represents the amount of work done when the item is found at a specific position , while the rectangle's height represents the probability of finding the item in this position (assuming equal probability of $\frac{1}{n}$). By making an upside down copy of the resulting shape, we get a large rectangle. Now, the average case running time of the list searching process will be half the area of the big rectangle which is $\frac{n+1}{2}$.

The alternating series convergence [30], and the amortised linear re-sizable arrays [8] visual proofs have applied a similar principle relying on areas of rectangles. For example, Figure 2.3 shows the convergence of the alternating series $a_1 - a_2 + a_3 - a_4 + a_5 - a_6 + ...$ relying on comparing rectangle areas.

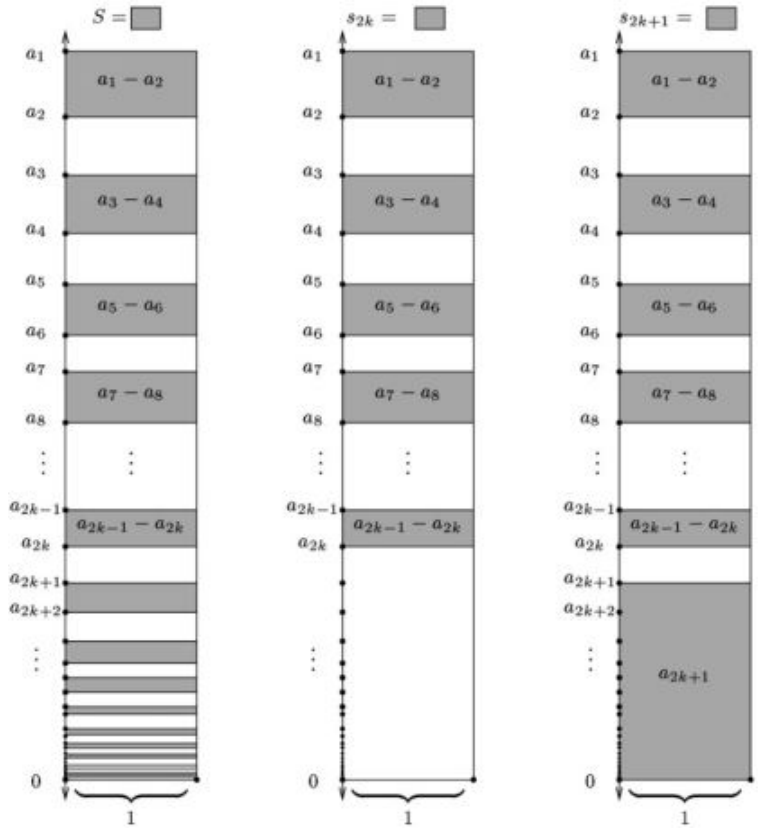


Figure 2.3: Alternating Series Convergence Visual Proof, from [30]

On the other hand, binary-tree traversal, bottom-up heap-construction, and the analysis of AVL trees proofs presented in [24] have applied a different principle. They simply present a series of visual depictions that support the discussion. For example, Figure 2.4 shows a visual proof of the running time complexity of the Euler tour that visits each node of a binary tree once, such that the edges of the tree are always on the left. As shown in the figure, there are no graphical primitives or areas used in this proof. The proof mainly relies on the intuition provided by the visual depiction of the traversal process. We name this

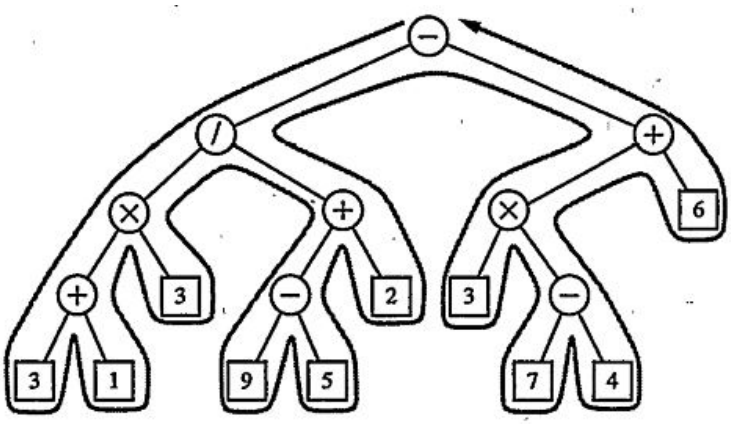


Figure 2.4: Euler Tree Traversal Visual Proof, from [24]

principle *"Visual Description"* for future reference.

To our knowledge, there have been no attempts to present algorithm analysis concepts using interactive visualizations that can be integrated into the curriculum as is done with AVs. In addition, all of the aforementioned visual proof attempts were not accompanied by any empirical evaluation to determine whether these static depictions are really engaging and useful for student learning. Evaluating the educational effectiveness of visualizations is a challenging task for many reasons. Some of the reasons are described in the following paragraphs.

First, the educational impact of visualizations depends not only on student learning gains when they use it, but also it depends on how widely it is used by instructors [60]. Among the obstacles faced by instructors to include AVs into their curriculum is the time and effort required to develop and integrate AVs into the curriculum, as well as the lack of the appropriate software tools to support this process [44, 60].

Second, despite the intuitive appeal of educational use of visualizations, results are mixed regarding the way of using them and the key features they should have in order to be effective in learning [40, 86, 87]. In [10], it was stated that for some visualization evaluation studies there were some benefits detected for students but not at the level hoped or expected by developers. An important result from [40] is that greater impact of AVs will be reached when the students are more engaged with the visualizations (how students use AVs rather than what AVs show them). Based on this, in [62], a taxonomy of six student engagement levels were proposed (no viewing, viewing, responding, changing, constructing, and presenting), and the main hypothesis was that the higher the level of student engagement with the visualization, the more the impact the visualization has on the student's performance. Some studies made use of this taxonomy and tried to test this hypothesis, however the results are mixed again. Some studies supported the hypothesis as in [60], while other studies did not fully support it [53, 87]. Other efforts were directed towards finding AVs features (i.e.,

user control and pseudo-code display) that affect its educational effectiveness [74]. Results suggest that allowing students to control the steps through the visualization and providing an example dataset has a significant effect on the AV's educational value, while displaying pseudo-code has no significant effect.

Third, assessing students' performance or learning gains is necessary and imperative in evaluating the educational effectiveness of visualizations. Affective gains (i.e., emotional and social influence) is another neglected measure that can directly affect the effectiveness of a visualization and are as important as and typically precede and drive learning gains [18].

Fourth, sometimes for ethical reasons some instructors are reluctant to divide their students of the same class into two groups and give each one a different treatment and then compare the learning gains of each group to see the result of the treatment [18]. There are some workarounds to this problem, such as comparing one classroom to another (maybe in different course offerings), but then we will end up with another problem which is how to show that both classes are similar in their level of knowledge and that they can be compared to each other based on the treatment.

Fifth, the visualization itself may provide some facility and this facility is the key factor in aiding learning rather than the visualization per se. For example, in [10], a study was conducted to evaluate AVs as a learning aid. Results suggested that one way visualizations may be a learning aid for algorithm dynamics concepts is by encouraging students to predict the algorithm behavior and make implicit guesses about the algorithm's next step. The study also concluded that static images can also stimulate students to predict the algorithm's behavior. Therefore, the visualization per se may not be the key factor in aiding learning.

Finally, students' experience can affect the evaluation of visualizations. For example, in [52], a study was conducted of whether a program visualization tool called Ville helps students to learn programming. By dividing the students into control and test groups and providing Ville as a treatment for the test group, the results favored the test group, however, the differences were too low to reject the null hypothesis. When the two groups were further divided into students with some programming experience and students with no programming experience, there was a significant difference between the performance of the no experience students in the control group and test groups. Accordingly, there was solid evidence that Ville enhanced the learning of students with no previous programming experience.

Despite the aforementioned challenges, researchers have tried to find measurements to evaluate the learning outcomes of visualizations. There are two forms of evaluating the learning outcomes of students when using visualizations: summative evaluations and formative evaluations [60]. Summative evaluations are those quantitative measurements that occur after providing the treatment, while formative evaluations are those qualitative assessments that occur during the study. Examples of summative evaluations include pre- and post-content tests to gauge the content mastery after using the visualization, in which the purpose of the pre-test is to measure the level of the learners' prior knowledge, and the purpose of the post-test is to measure the level of knowledge after using the visualization. Students' grades are

another example of summative evaluations that are used to measure the level of success in a course. We may have some evidence about students' learning outcomes by comparing their grades before and after using the visualizations. Time evaluation [60] can be thought of as both summative and formative. The goal is to record how much time is spent by the learner working with the visualization, which may be a good indicator in assessing the learner's interest or engagement in the visualization. Tools to log the learner's interactions with the visualization are useful to be able to perform time evaluation. These tools in OpenDSA are discussed in [9, 20].

By analyzing studies about evaluating learning gains from educational interventions, we found that the evaluation process is similar among different types of interventions such as visualizations [10, 51, 52, 53, 60], online tutorials [31, 50], educational games [29, 38, 68], and student self reflection [22]. Based on these studies, the evaluation protocol can be defined by the following steps:

1. **State the hypotheses:** The set of hypotheses to be tested should be clearly defined. For example, in [68], one of the hypothesis was "The students of the test group will exhibit significantly greater achievement in terms of computer memory knowledge than those of the control group". In [50], the hypothesis was "Practicing with the online tutor will be at least as effective as practicing with a printed workbook that includes answers to problems as an appendix". In [53], the hypothesis was "The changing level of engagement will result in significantly better learning than the viewing level, and the constructing level is expected to yield significantly better results than changing".
2. **Define Participants:** The subjects should be selected and then assigned to either the control or the test (intervention) group (between-subjects evaluation). Control and test groups could be further divided into more groups to test the effect of some co-variant factors such as learning styles [60], gender [68], previous experience with the topic [52], and others. For example, in [68], the subjects were high school students studying computer memory concepts. Subjects were randomly assigned to control and test groups that are further divided into boys and girls to test the effect of the gender co-variant factor. In [52], the subjects were university students in their first year programming course. Again, subjects were randomly assigned to control and test groups that are further divided to students with no programming experience and students with some programming experience to test the effect of previous knowledge on the learning gains after using the treatment.
3. **Describe the Materials:** All materials (i.e., traditional method and treatment as well as pre- and post-tests) that will be used to conduct the experiment should be clearly defined. For example, in [68], a gaming application called "LearnMem1" was defined to be the treatment, while another online application called "LearnMem2" was defined to be the traditional method. Also a computer memory knowledge test composed of 30 True/False questions was constructed by the researcher and was used as a pre- and post-test. In [26], the JHAVE visualization tool supplementing the lecturing slides was used as the treatment, while the traditional method was composed of the lecture slides

provided for the treatment accompanied with a five page study guide. The pre- and post-tests were defined in an appendix at the end of the study.

4. **Describe the Procedure:** The steps for conducting the experiment should be stated. In most cases, summative evaluations using pre- and post- content tests are adopted. That is, a pre-test is first given to the students divided into two groups, a control and a test group, to gauge their level of prior knowledge about the topic of interest. Then the intervention (or the treatment) is provided to the test group, while the control group is provided the traditional teaching method. Finally a post-test is given to both groups to see the difference in knowledge acquired.
5. **Conduct Data Analysis:** After conducting the experiment, this is the time to perform the statistical analysis and make inferences about whether the hypotheses would be accepted or not. The independent variables (in most cases it will be the treatments provided along with any co-variant factors) as well as the type of the statistical analysis (i.e., ANOVA, t-test) should be clearly defined. For example, in [26], non-parametric Kruskal-Wallis and Man-Whitney tests with 5% significant level were used for comparisons between pre-test scores of different groups. ANOVA was used for testing the significance between groups for post-test results. In [22], a statistical measure called the average normalized gain was used to measure the level of improvement between post-test and pre-test results. In [52], pre- and post-test results among the subject groups were analyzed using a two-tailed and pair-wise t-test.

2.2 Concept Inventories

The difficulty of evaluating the learning gains of pedagogical interventions has motivated a call for adopting standardized assessment tools [34]. Many students begin their courses with a set of preconceptions that can be leveraged to guide them in deep understanding of a particular concept or it may lead to misconceptions that if not treated may persist and hinder their understanding [89]. Before instructors can hope to develop curricular interventions to repair students' misconceptions, they have first to identify which concepts their students misunderstand and what are the prevalent misconceptions [64].

A Concept Inventory (CI) is a standardized assessment tool that evaluates whether a student's conceptual framework matches an accepted conceptual framework of a particular topic [17, 34]. Thus, it can be used to classify students into two groups: a group that thinks in accordance with accepted conceptions in a discipline, and a group that thinks in accordance with common misconceptions in the same discipline. A CI can also be thought of as a set of Multiple Choice Questions (MCQs) in which each item targets one or more concepts and item distractors correspond to student misconceptions related to this concept [11, 34]. A CI can be used to measure learning gains when administered as a pre- and post-test in a given evaluation study [64]. Accordingly, it can be used as a comparison tool to measure learning gains with different pedagogical interventions [34, 82].

The advent of CIs started with the Force Concept Inventory (FCI) [36] that was mainly designed as a set of MCQ questions to probe students' beliefs about the basic concepts of Newtonian mechanics. The FCI revealed a vast difference between student and expert understanding of core physics concepts and it is now thought of as the gold standard that all other CIs are compared to [82]. The impact of the FCI had set the stage for more attempts to develop CIs for STEM disciplines. STEM CIs have been developed for the fields of statics [81], statistics [2], thermal and transport [64], geoscience [54], signals and systems [88], fluid mechanics [58], electromagnetics [65], and heat transfer [41].

In CS, CI development still in its beginning [82]. However, there have been a number of attempts and ongoing work regarding the development of reliable and valid CIs for CS topics [84]. Some of the attempts were directed towards CS1 level courses as in introductory programming [11], discrete mathematics [4], and CS1 fundamentals [83]. Other attempts were directed towards other courses as in digital logic [34], operating systems [89], algorithms and data structures [17], and computer architecture [70]. The interested reader can consult [31, 82] for more CS CIs.

Designers of CIs for CS topics face many challenges. Some of these challenges are not specific for CS like those described in [70], while the others are unique to CS such as those presented in [82]. The most concerning to us are:

- Most of the time administering a CS CI as a pre-test is pointless regarding detecting student misconceptions as student misconceptions in a CS topic is unlikely to come from their misunderstanding of the world as in other disciplines like physics [83]. Accordingly the misconceptions revealed from a pre-test administration of a CS CI could be false positives and the actual problem may be just a student problem in understanding the meaning of the notations or programming constructs [70, 89].
- If the CI items contain notations from a specific programming language, then if a curricular change happens, the CI items should be changed accordingly [82]. [84] presents an attempt to develop a language-independent CI for CS1 topics relying on pseudo-code. However, students should be familiar with the psedo-code notation before attempting the test, which may add some unnecessary learning curve.

According to [17, 23, 34, 64], the major steps in developing a CI are defined as follows.

1. **Define the scope:** A CI is not intended to be comprehensive like exams, it only focuses on critical concepts for a topic [11]. It is typically administered as both a pre-test at the beginning of a course and a post-test at the end to measure the learning gains from a particular instruction method [23]. Accordingly, the scope of the test must be determined at the beginning to include only those course topics (concepts) that are critical for the course and that can distinguish students according to their level of conceptual understanding [11, 23]. To come up with the list of concepts, a Delphi process [16] applied in [23, 34, 64] may be adopted to obtain a consensus among a group of experts through informed decisions. The Delphi process has the advantage of overcoming the bias that may come from the single expert approach due to experience

and specialty [64]. The Delphi process also allows each person's views to be heard, and provides the benefit of anonymous access to the views of other group members so that the influence is only based on the logic of the arguments not the experts' reputation [23]. In addition, critical concepts can be also identified by intensive synthesis of different sources such as textbooks, papers, and exams [11].

2. **Identify misconceptions:** After defining those concepts that are important for the underlying topic, the emphasis then should be directed towards coming up with a list of students' common misconceptions in understanding those concepts. Both instructors and students should be consulted in this phase [34]. Instructors (maybe through a Delphi process) can identify misconceptions from their teaching and grading experience. Students also should be interviewed to determine why they fail to understand key topics correctly. According to [23], only students can provide reliable information about their misconceptions. The list of important concepts should be revised after this step to include only those concepts that are both important and difficult for students to understand [23, 34].
3. **Develop questions:** Using the list of concepts and misconceptions, CI developers should construct questions related to each defined concept, that should trap those students with misconceptions. The list of questions should cover each concept multiple times to strengthen the validity and reliability of the measurement [11, 34]. The type of questions can include both MCQs and open ended questions. Open ended questions are able to obtain a direct vision of the students' thinking process and understanding of how deeply they interpret the concepts [67]. Open ended questions are helpful when used in an alpha administration of the CI to catch new student misconceptions not already defined in the current misconception list [11]. In [23], open ended questions were generated by experts for each targeted concept and students were invited to answer these questions. A think-aloud protocol was applied to discover students' mental models of the targeted concepts. During the think-aloud sessions, some misconceptions emerged which were then used to design reasonable distractors for MCQ corresponding to these misconceptions. On the other hand, MCQs can be effective when each distractor matches a student misconception [34]. Those distractors that are not chosen by students may be indicators that the corresponding misconceptions are not held by the students and hence, the misconception list should be updated accordingly [34].
4. **Check reliability and validity:** Testing the reliability and validity of a CI is a tedious process [64]. For a CI to be widely accepted as a testing instrument it should be both valid and reliable [34] and a CI cannot be valid if it is not reliable [64]. Reliability measures the instrument's consistency among test scores [81] whether it is an internal consistency among individual test items (single administration measures) or it is the consistency among scores in repeated administrations (multiple administration measures) [34]. A CI reliability test answers these questions: "Do repeated administrations of the test yield the same results?" [64] or "Is there is any correlation between the scores of individual test items?". Or in other words "How well do the items on the test measure the same construct?" It was stated in [3] that internal consistency reliability

has an advantage over multiple administration reliability as it doesn't require giving the test more than one time. Measures for internal consistency include Cronbach's alpha and its Kuder-Richardson variations KR-20 and KR-21. KR-20 is a special case of Cronbach's alpha for dichotomous items (the answer is either true or false with no partial credit). KR-21 is a special case of KR-20 when all the items are assumed to have equal difficulty. It was stated in [66] that an alpha value of 0.7 or higher might be reasonable. In [34], KR-21 coefficient was applied and it was stated that a value of 0.4 is considered acceptable.

Validity measures whether the instrument actually satisfies the purpose for which it is used [81, 83]. In other words, a CI validity test answers the question: "Are we measuring what we think we are measuring? Have we covered the required concepts?" [64]. There are three main types of validity:

(a) **Content validity:** Content validity is satisfied when the instrument's items appropriately measures the construct it intend to measure [83, 84]. This can be performed through rational analysis of the test content based on personal subjective judgment when an expert or even an examinee examines the test and concludes that it measures the required trait [3]. It was stated in [34] that for a CI to be considered successful, expert content validity should be performed in which a panel of experts evaluates each CI question on whether it targets the required concept and addresses the intended misconception(s). Determining if the instrument accurately reflects students misconceptions is another form of content validity, named student content validity [34]. This is done after the CI is administered to see whether all or most of the misconceptions are represented in the students' answers. In addition, student interviews should be conducted to understand whether distractors were chosen because of a student misconception or a misunderstanding of the question. A similar content validation process relying on expert validity and student interviews was also done in [75].

(b) **Criterion-related validity:** Unlike content validity, criterion-related validity is based on statistical measures [3]. In order to have criteria-related validity, a correlation between test scores and other related measure should exist [81] (i.e., correlation between test scores and student GPA or overall course grade).

(c) **Construct validity:** The construct is the actual trait or characteristic an instrument is measuring [83]. Construct validity has traditionally been defined as the experimental demonstration that a test is measuring the construct (i.e., proficiency in a certain topic) it claims to be measuring [3]. Construct validity can be measured in two ways, convergent validity and discriminant validity [3, 39]. In convergent validity, another instrument that is known to be measuring the same construct should be consulted and scores from both instruments should be strongly correlated. In discriminant validity, scores between the instrument to be validated and another instrument that is known to be not related to the target construct should not be correlated. Construct validity could take the form of a between-group study, wherein the performances on the test are compared for two

groups: one that has the construct and one that does not have the construct. If the group with the construct performs better than the group without the construct, that result is said to provide evidence of the construct validity of the test. Also construct validity could take the form of a within-subject study wherein a group that is known to be weak in the construct is measured using the test, then taught the construct, and measured again. If a non-trivial difference is found between the pre-test and post-test, that difference can be said to support the construct validity of the test.

Determining if the instrument is biased is another form of validity applied in [81] and [34]. This can be tested by administering the CI for two or more groups having different majors, gender, race, ethnicity, culture, etc, and then conduct the appropriate statistical test (i.e., ANOVA) to see if there is no significant difference between the scores of the two groups.

In addition, CI developers are interested in evaluating whether each individual CI item performs as desired. Classical Test Theory (CTT) [15] and Item Response Theory (IRT) [5] provide the tools to answer the question of whether an individual test item is useful according to some metrics such as difficulty and the discrimination between good performers and bad performers on the test. [34] applied the Item Response Curves (IRCs) (also called item to total correlation curves [15]) to check item quality in terms of whether the performance in the CI as a whole is correlated to the performance in each individual item. [64] applied item difficulty and item discrimination measures to decide whether CI items are truly performing as desired. An item that is difficult to the extent that nearly all students didn't answer it correctly, or is easy to the extent that nearly all students answered it correctly, may be useless. In addition, an item may be also useless if it fails to well discriminate between good performers and bad performers in the whole test (i.e., has low discrimination index). Similarly, item and discrimination measures were also applied in [81].

Chapter 3

Algorithm Analysis Visualizations

3.1 Introduction

OpenDSA's material on algorithm analysis was originally based on text and static images, similar to the presentation found in any traditional DSA textbook. In this chapter, we present the motivation behind developing a new generation of visualizations that we name Algorithm Analysis Visualizations (AAVs). AAVs are capable of conveying algorithm analysis material as presented in a typical CS3-level course in a visual interactive way. We present results from analyzing OpenDSA student interaction logs and surveys showing that students were not engaged in the original algorithm analysis material available in OpenDSA.

In response to the motivation for AAVs and with the inspiration by visual proofs, a pool of AAVs was developed and integrated into OpenDSA sorting and algorithm analysis introduction modules. In this chapter, we present some examples of the developed AAVs, and show how they rely on the two visual proof principles described in Section 2.1.

3.2 Motivation

3.2.1 Algorithm Analysis Topics are Hard

Since students in our DSA course are using OpenDSA as the course textbook, we have the opportunity to closely monitor how students make use of the course materials, and how they perform on the online exercises. We first tried to determine which topic areas or exercise types give students the greatest difficulty. OpenDSA provides a collection of online, open-source tutorials that combine textbook-quality text with algorithm visualizations, randomly generated instances of interactive examples, and exercises. Content within OpenDSA is organized into modules, each focusing on a specific topic such as Quicksort or Closed Hashing.

The modules contain a wide variety of exercises. Some require that the student manipulate a data structure to show the changes that an algorithm would make on it. We refer to these as "Proficiency Exercises" (PEs). This type of exercises was pioneered in the TRAKLA2 system [56]. In addition, OpenDSA uses the Khan Academy (KA) exercise framework * to provide Multiple Choice Questions (MCQ), True/False (T/F), and short answer exercises.

We studied 143 student participants enrolled in a CS3-level course at Virginia Tech during Fall 2014. OpenDSA was used as the main textbook, and students had until the end of the semester to complete the OpenDSA exercises. OpenDSA exercises accounted for 20% of the course final grade. We analyzed OpenDSA exercises with respect to the number of hints used, and the appearance of a trial-and-error strategy to "guess" the answers. Harder exercises are expected to display a higher rate of hints use and/or trial-and-error.

Exercises using the KA framework generate a series of question instances on the topic. The student must get a certain number of correct instances (typically five to six) to complete the exercise. One point is deducted from the student's credit toward this requirement when they submit an incorrect answer, to discourage guessing. Students can also take one or more hints that explain the answer to the question. In this case, the attempt is not graded (no point is awarded or deducted toward the threshold).

To analyze exercises based on students' hint use, we computed the hint ratio for each KA exercise as follows.

$$hr = \frac{\text{\#of hints used}}{\text{\#of total attempts}}$$

To analyze exercises based on the rate of trial-and-error, we calculated the incorrect ratio for each KA exercise as follows.

$$ir = \frac{\text{\#of incorrect answers}}{\text{\#of total attempts}}$$

Inspecting exercises in the fourth quartile (exercises in the highest 25% incorrect ratio), we found that they are related to the topics of algorithm analysis, heaps, quicksort, radixsort, shellsort, and heapsort.

The seven exercises shown in Table 3.1 had high hint or high incorrect answer ratios. They relate to topics covering mathematical background and runtime analysis of quicksort, hashing, and shellsort. 45% of students heavily (third quartile and up for all exercises) used hints, and provided many incorrect answers when solving these seven exercises. We found that most exercises with low incorrect answer and hint ratios are for stacks, arrays, and lists. These are topics that most students know from previous courses. When using high rate of hint use as a measure of exercise difficulty, we found that exercises related to algorithm analysis and mathematics topics appeared to be more "difficult".

*`http://github.com/Khan/khan-exercises`

Table 3.1: *ir* and *hr* for difficult exercises

Exercise	*hr*	*ir*	Topic
ListOverhead	0.93	0.6	List Overhead Analysis
TreeOverheadSumm	0.78	0.73	Tree Overhead Analysis
QuicksortSumm	0.32	0.67	Quicksort Analysis
AlgAnalSumm	0.24	0.77	Algorithm Analysis
MthBgSumm	0.25	0.63	Mathematical background
ShellsortSumm	0.16	0.61	Shellsort
QuicksortPartitionPRO	0.27	0.58	Quicksort's partition

3.2.2 Why Algorithm Analysis is Hard? Evidence from Fall 2014 Interaction logs

OpenDSA modules originally presented algorithm analysis material relying mainly on textual discussion supported by a small number of static images, similar to a standard textbook. In order to see whether this material was engaging for students, we analyzed student interaction logs from use of OpenDSA's analysis material related to the sorting chapter (this chapter contains more than 40% of all analysis material in a typical CS3 course). The goal was to estimate how much time students spent reading the analysis material. Time spent here was used as a proxy for engagement. Unfortunately, at that time, OpenDSA's data collection tools [9] did not provide a direct method that we could use to estimate the time spent by students reading the analysis material in each module, because the main focus of these tools was to collect user interactions with the interactive content. Accordingly, the OpenDSA authoring system was extended to support wrapping arbitrary module sections behind a show/hide button as is done with other interactive content. When the button is pressed to show the content, the time is recorded and can be used as an estimate of the time when the student started reading the material. The time for finishing the material can be estimated from the time of the next event, whether it is the loading of the next exercise right after the analysis material, or leaving the module as recorded by the data collection tools. Figure 3.1 shows the user interface for this show/hide support for the analysis content in the Insertionsort module.

We analyzed the interaction logs available for three OpenDSA book instances used at three different universities (Virginia Tech, university of Texas El Paso, and University of Florida) for three different courses (CS3114, CS2401, and COP3530, respectively) during Fall 2014. Figures 3.2, 3.3, and 3.4 present the distribution of the estimated reading time for three sorting modules (Insertionsort, Mergesort, and Quicksort) for Virginia Tech (VT), University of Texas El Paso (UTEP), and University of Florida (UF), respectively. Results are summarized in Table 3.2. In this analysis we have excluded estimated times more than 10 minutes as it may be the case that the student has pressed the show/hide button to start reading the content and then he left the browser window open while doing something else.

In this situation, time to the next event is not an accurate estimate of time spent.

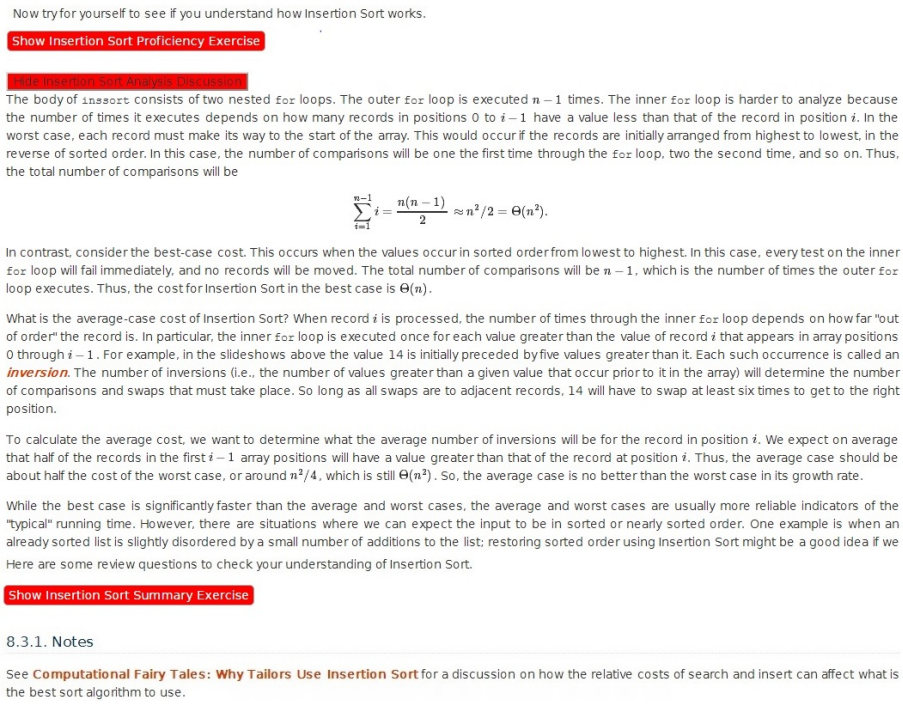
As we see from the results, more than 74% of the students spent less than 1 minute on the analysis material for a given module, for all 3 universities. Based on this result, we believe that most of the students did not read the analysis material.

Table 3.2: Results Summary

University	Module	N	μ(Sec)	% <1min
Virginia Tech	Insertionsort	98	63.57	74.48
	Mergesort	96	39.79	78.12
	Quicksort	92	64.71	73.91
UTEP	Insertionsort	26	49.84	80.76
	Mergesort	22	41.45	77.27
	Quicksort	16	16.18	93.75
Florida	Insertionsort	53	40.39	84.90
	Mergesort	44	18.63	95.45
	Quicksort	39	26.12	92.30

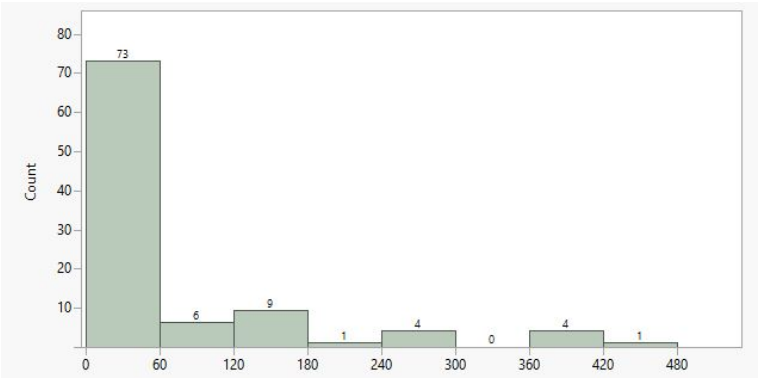
In order to see whether there is a significant difference in the time spent in the analysis material for each module between the 3 universities, three Kruskal Wallis tests were conducted for each module. We found that the difference is not significant for the three modules (Insertionsort $p = 0.171$, Mergesort $p = 0.2662$, and Quicksort $p = 0.1288$). We also tested whether there is a significant difference in the time spent in the analysis material between modules. Three Kruskal Wallis tests were conducted for the three sorting modules within each university. We found that the difference is not significant for the three universities (Virginia Tech $p = 0.2495$, UTEP $p = 0.1256$, and Florida $p = 0.2062$). We were expecting a significant difference between the modules since they have different levels of difficulty regarding the analytical material (e.g., Insertionsort material is easier than that of Quicksort). Since we didn't get this difference from our analysis, we can conclude that the students did not read the analytical material for any of the three modules.

(a) Insertionsort analysis hidden

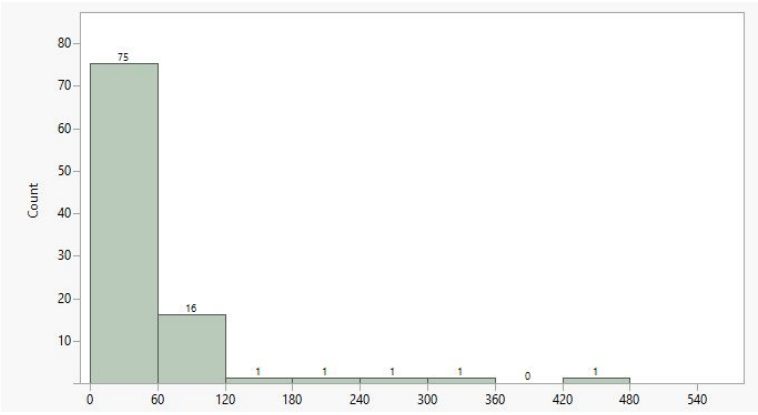


(b) Insertionsort analysis shown

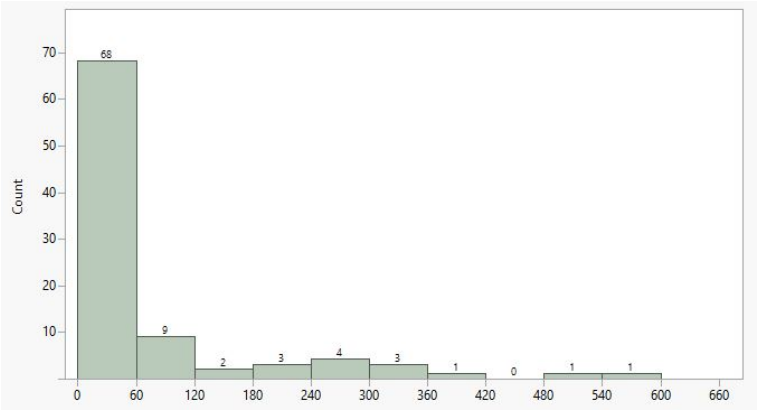
Figure 3.1: Wrapping Insertionsort analysis content behind a show/hide button



(a) Insertionsort Module

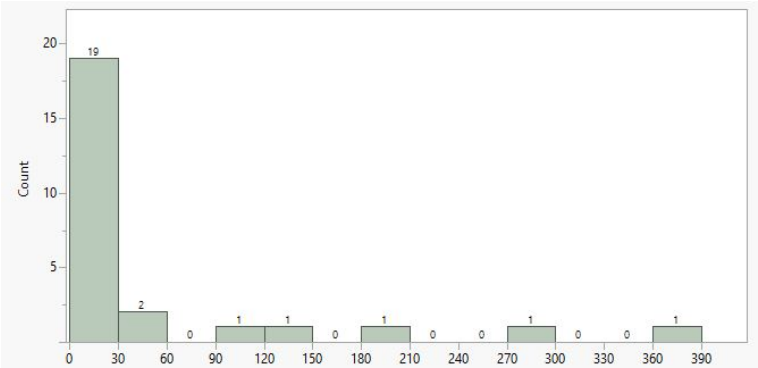


(b) Mergesort Module

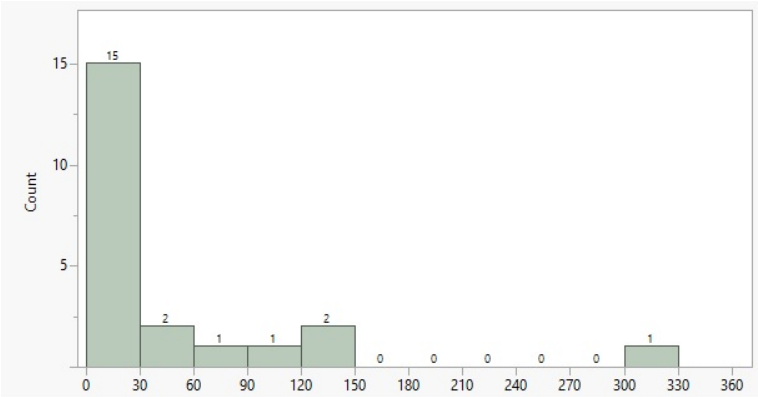


(c) Quicksort Module

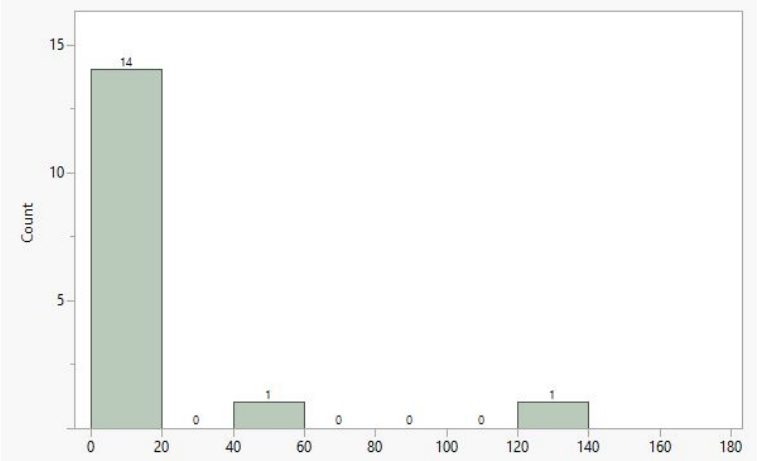
Figure 3.2: Estimated time spent by students reading the analysis material at VT



(a) Insertionsort Module

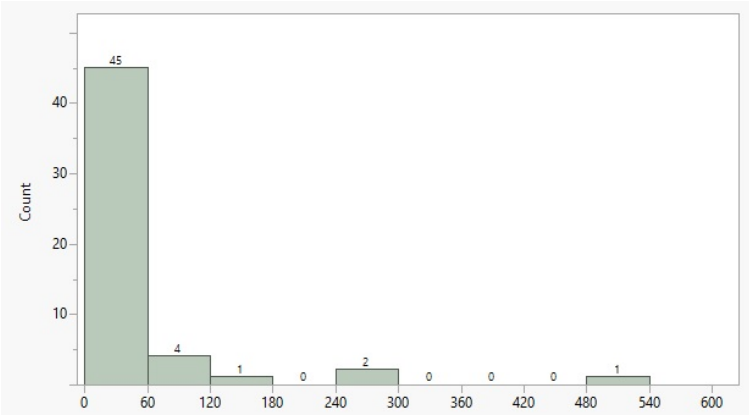


(b) Mergesort Module

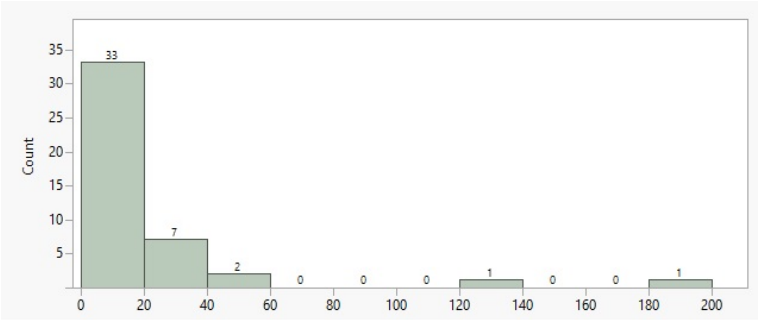


(c) Quicksort Module

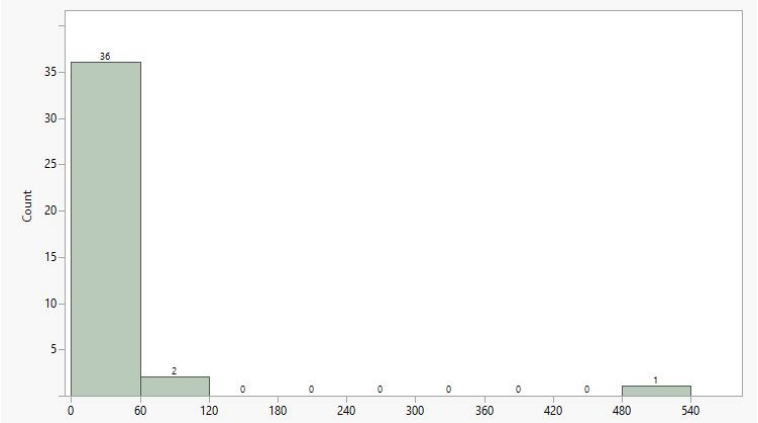
Figure 3.3: Estimated time spent by students reading the analysis material at UTEP



(a) Insertionsort Module



(b) Mergesort Module



(c) Quicksort Module

Figure 3.4: Estimated time spent by students reading the analysis material at UF

3.2.3 Evidence from Student Survey

To further support the results from the analysis in the previous section, student surveys were given at the end of the semester. 84 students were surveyed at the end of the CS3114 course offered at Virginia Tech during Fall 2014. Students were asked four questions related to algorithm analysis content. They were asked to provide a rating on a scale of (1-5) about their level of confidence in understanding algorithm analysis material after the course. The second question asked which task is easier for them, understanding how an algorithm works (dynamics) versus understanding its asymptotic running time. They were asked, whether the reason behind this is intrinsic to the material itself or due to the method of presentation in OpenDSA. They were asked to mention whether OpenDSA's content for algorithm analysis was useful to them. Finally, they were asked to provide suggestions for enhancing the presentation of analytical material. The complete survey is available in Appendix C.

For the first question, 18% of the surveyed students rate their level of confidence in algorithm analysis by five, 67% by four, and 14% by three. This indicates that there is a significant group of students who are still not comfortable with the analysis content after finishing the course.

For the second question, 85.58% of the surveyed students found that it is easier for them to understand how an algorithm works than analyzing the running time of an algorithm. Here are some representative quotes.

- *"Determining asymptotic running time because it is harder to visualize and less intuitive."*
- *"Complexities are confusing and math-like."*
- *"I think understanding how an algorithm work is easy. It is the style of presentation."*
- *"How the algs work. It is dependent on material, also abstract stuff is harder for me to understand."*

78% of the students who are more comfortable with dynamics attributed this to the material itself as algorithm analysis is more abstract and requires some familiarity with mathematical notations, while 22% attributed this to the way both concepts are presented in OpenDSA (dynamics are presented using AVs, while analysis is presented mostly through text).

Regarding the usefulness of OpenDSA content for algorithm analysis, here are some representative quotes.

- *"Not any more useful than any other book."*
- *"Not as much as learning the algorithms themselves, but I felt it was as useful as any resource could be on the topic."*
- *"Yes, but not as much as understanding the algorithms."*
- *"Yes, but it could have been more interactive with showing why the analysis was the way that it was."*
- *"I found it much more useful on Data structures. Algorithm analysis doesn't benefit*

quite as much from animations."

- *"No, it was very detailed and kind of hard to follow."*
- *"Kind of. I'd like there to be more visuals for analysis."*

It is clear that most of the students did not find the material different from any other textbook on the same topic. This is not what they were expecting from OpenDSA, which is based on interactive content.

When the students were asked to provide suggestions for improving the way the analysis material is presented in OpenDSA, we found that most of them are expecting more interactive presentation in the form of animations. Here are some representative quotes.

- *"Visualizations definitely help. I would say to have a larger variety of questions in the graded parts."*
- *"I think making the clickthrough pictures into actual animations would be nice."*
- *"More animation, the visualizations are great!"*
- *"More visualizations is always good."*
- *"Visualizations always help :)"*
- *"Visualizations showing each step of analysis would help"*
- *"An animation will make a much bigger difference."*

In addition to the survey, four students were interviewed and were asked questions similar to those presented in the survey. The interview protocol is presented in Appendix D. All of the surveyed students reported that they were having troubles with algorithm analysis mostly because it is based on math and formulas. Two of the students stated that their knowledge of algorithm analysis was based on memorization. All of them were more comfortable with dynamics than analysis, and they liked the way dynamics are presented in OpenDSA using traditional AVs. All of them were in favor of seeing the analytical material presented visually, just like the dynamics are presented. However, one of them stated that it would be better to present both kinds of content, the textual and the visual, and then students are free to pick the style they want. One of the students also mentioned that she would like to see more exercises on algorithm analysis.

3.3 AAVs Implementation

Based on the previous motivation for a more engaging presentation to OpenDSA algorithm analysis material, a set of AAVs were developed. To be consistent with the current available AVs, AAVs were developed using the JSAV framework [45, 46]. The JSAV framework contains functionality to support the creation of interactive AVs that focus on algorithm dynamics, and OpenDSA is rich with this type of visualizations. JSAV allows developers to easily create AVs by providing graphical displays for several types of data structures such as arrays, lists, heaps, trees, and graphs. The developer's work will be only to implement the algorithm itself and display the data structure by adopting the manipulation functions

available in JSAV. JSAV also supports some constructs that can be used to develop AAVs. For example, when designing visual proofs of algorithm running-time complexity, primitive constructs are used such as rectangles and triangles. JSAV provides support for these constructs and their manipulation methods that allow developers to implement a visualization for the required proof.

Another reason of adopting the JSAV library is that it makes it easy to integrate AAVs with OpenDSA modules. JSAV is implemented in pure HTML5 and JavaScript, which allows the created visualizations to be easily embedded in OpenDSA's hypertext content and avoid the problems that were previously faced when using Java applets, Java web start, and other technologies described in [43]. Prose content in OpenDSA is created using ReStructured Text (RST), which is then compiled using Sphinx [†] to produce eTextbook modules in HTML format. Visualizations (which are in fact HTML/JavaScript pages) can be easily embedded into modules by using special Sphinx directives written in the RST file [20].

Overall, 28 AAVs were implemented. 14 AAVs were written for the sorting chapter. These focused on running-time proofs for the basic sorting algorithms. The remaining 14 AAVs present general algorithm analysis concepts in the algorithm analysis introductory chapter.

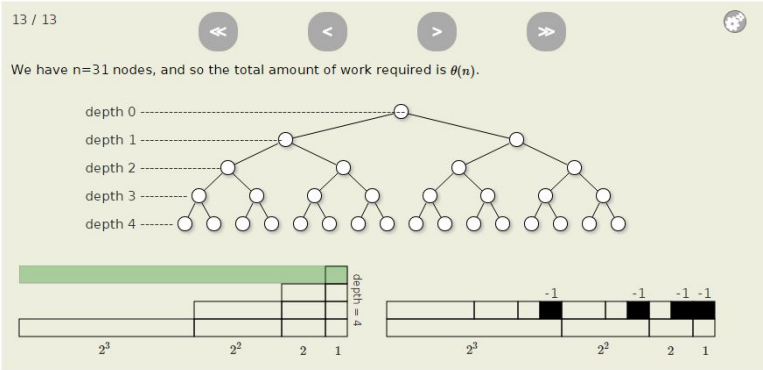
All AAVs were implemented relying on Mayer's multimedia principle presented in [59]. Our goal at this stage was to improve the presentation of the material by using a more visual approach. AAVs naturally fall into the viewing level of engagement according to Nap's engagement taxonomy defined in [62], as the student now can view the analysis material in a visual, interactive way. We note that it might be possible to change the approach at a deeper level, and so to engage students at a higher level in the engagement taxonomy. We mention some possibilities in Chapter 6.

3.3.1 Area-to-Cost AAVs

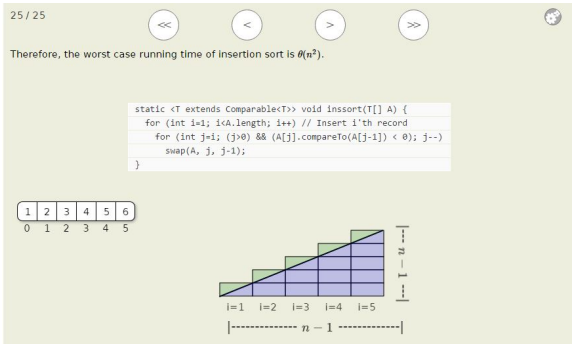
Inspired by the concept of visual proofs and by applying the area-to-cost principle described in Section 2.1, a set of AAVs were implemented for OpenDSA sorting modules (e.g., Insertionsort, Bubblesort, Selectionsort, Mergesort, and Quicksort). In addition, one AAV was developed for the heaps module presenting the Build-heap running time proof. Figure 3.5 shows some examples of these visualizations.

Figure 3.5(a) presents a visualization to illustrate the running time analysis of the Build-heap algorithm. The running time analysis of the Build-heap algorithm is confusing for students as intuitively it appears to be $\theta(n \log n)$ but it is actually $\theta(n)$. The visual analysis of the Build-heap algorithm was first introduced in [24] under the concept of visual proofs using only static images. Here we developed the same visual proof presented in [85] as an interactive visualization.

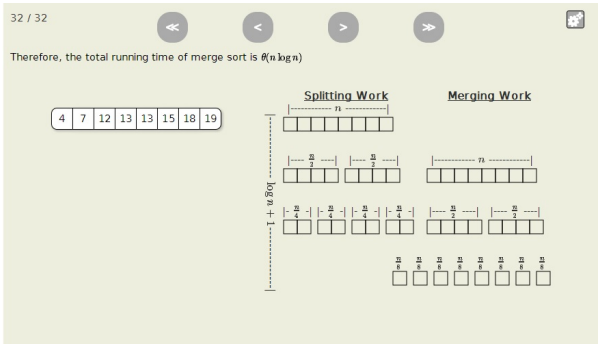
[†] `https://pypi.python.org/pypi/Sphinx`



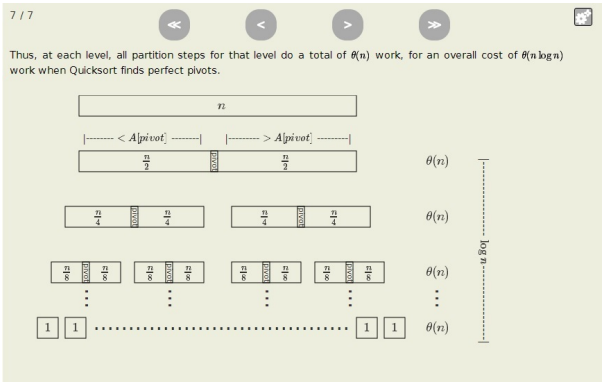
(a) Build-heap analysis



(b) Insertionsort worst-case analysis



(c) Mergesort analysis



(d) Quicksort best-case analysis

Figure 3.5: Visualizations illustrating the running time analysis of some sorting algorithms

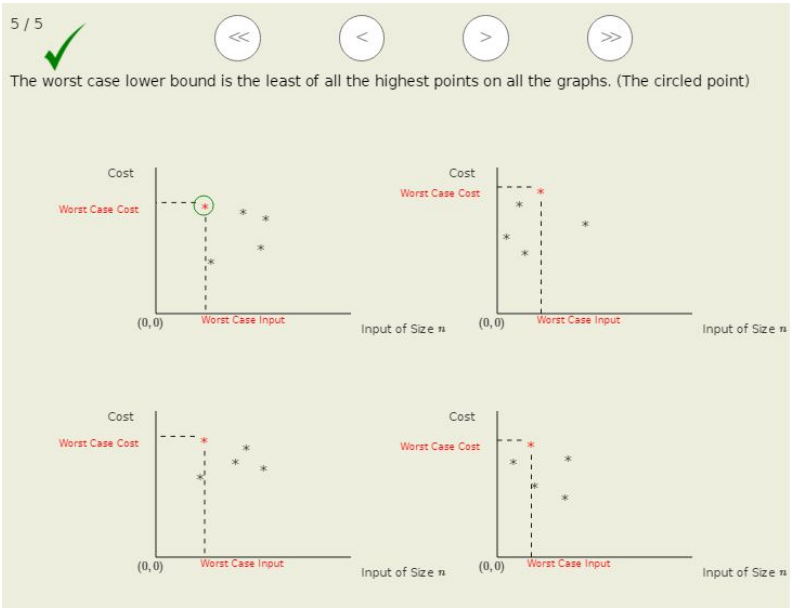
Figure 3.5(b) shows a visualization for the worst-case analysis of Insertionsort. An array example that produces a worst-case problem instance (array values are reversed in order) is displayed. A code display is shown to relate each algorithm step with the line of code that produces it. The emphasis here is to show the total amount of work done (unlike other AVs which are focused on how the algorithm works). The idea here is based on showing the step by step execution of the algorithm on the worst-case instance presented, and at each step show the amount of work that this single step produces which is here shown as a single rectangle with width and height of one unit. At the end we reach the shape shown in the figure, and the total amount of work now will be the surface area of this resulted shape which can be easily calculated from the figure as $\frac{(n-1)(n-1)}{2} + \frac{(n-1)}{2}$ which is finally evaluated to $\frac{n(n-1)}{2}$. We know that the worst-case running time of Insertionsort is described by the summation $\sum_{i=1}^{n-1} i$, where n is the size of the array. This summation is evaluated to $\frac{n(n-1)}{2}$ which is the same as the total surface area of the shape shown in the figure. By showing the amount of work graphically, this can help students acquire better intuition about the worst case running time of insertion sort which is known to be $\theta(n^2)$.

Figure 3.5(c) shows a visualization for Mergesort analysis. In Mergesort, the input array is partitioned into two halves for each call to the Mergesort method. In this visualization an arbitrary array is given at the beginning. The algorithm is executed step by step, and the amount of work for each step is shown on the right and is divided into splitting work and merging work. As we see from the figure, a set of squares are generated, each of which indicates a single unit of work. They are shown as levels to be related to the level in which it was generated (Mergesort is implemented as a recursive method). As shown in the figure we have a total of $\log(n) + 1$ levels, and at each level we have at most $2n = \theta(n)$ amount of work. Hence the total running time of the Mergesort is shown to be $\theta(n \log n)$.

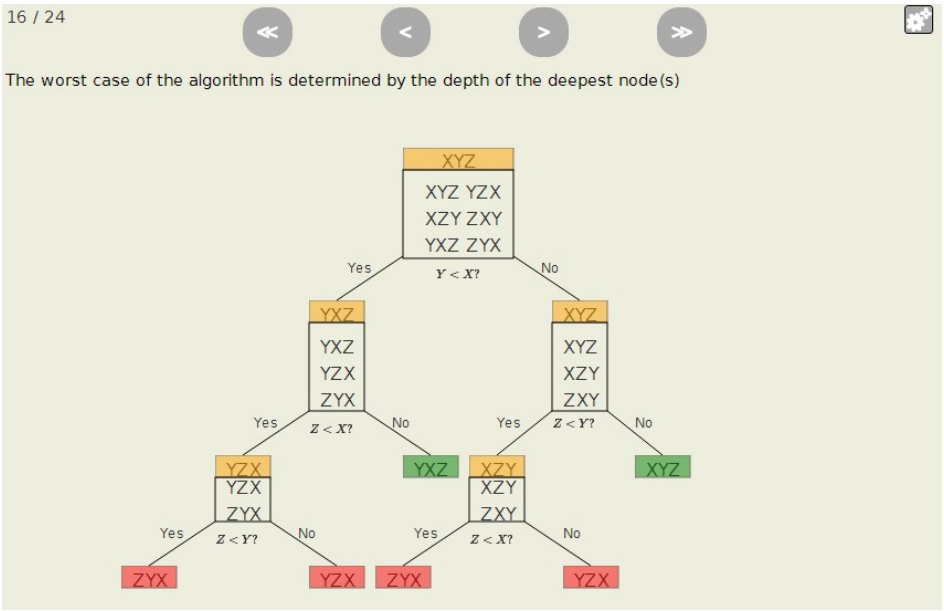
Figure 3.5(d) shows a visualization illustrating the best-case Quicksort running-time analysis. For Quicksort, the best-case running time occurs when the pivot partitions the input array into two halves at each level of calling the Quicksort method. Since the array values do not matter in our analysis, the visualization represents the array as a rectangle of width n. At each level, the array is partitioned into two halves resulting in $\log(n)$ levels. Since at each level the amount of work done is $\theta(n)$ (the cost of the partition method), then we have the total amount of work required by Quicksort in the best case is $\theta(n \log n)$.

3.3.2 Visual-Description AAVs

We have also developed a pool of AAVs relying on the visual-description principle described in Section 2.1. Most of these AAVs were used in the algorithm analysis introductory chapter in OpenDSA. Unlike the AAVs presented for sorting, these visualizations don't apply the area-to-cost principle. This type of AAVs presents an interactive visual depiction of the proof discussion step by step. Figure 3.6 presents two examples of this type of AAVs.



(a) Problem Lower Bounds



(b) The Lower Bound For the Sorting Problem Proof

Figure 3.6: Visualizations illustrating problem lower bounds

In Figure 3.6(a) a graphical depiction of the definition of the worst-case lower bound for a problem is presented. Each graph in the figure represents an algorithm that solves the underlying problem and each point in the graph represents the running-time cost of the algorithm for a problem instance of size n. The point circled representing the worst-case lower bound for the problem is shown to be the least of all the highest points in the graph. This represents the worst-case running-time of the problem.

Figure 3.6(b) presents an AAV to the sorting lower bound proof ($\Omega(n \log n)$). The proof is presented step by step by gradually creating the decision tree that models the processing of Insertionsort on an array of 3 elements.

Appendix A presents more examples of AAVs following the area-to-cost and visual-description principles.

Chapter 4

The Algorithm Analysis Concept Inventory

4.1 Introduction

In this chapter we present our efforts to develop a Concept Inventory (CI) for algorithm analysis topics for CS3-level courses. First we present the process of identifying difficult and important algorithm analysis topics in a typical CS3 course to be targeted by the CI items. Second, we present our initial list of student misconceptions developed based on previous work on identifying misconceptions related to program efficiency, years of experience with teaching CS3-level courses, and analyzing student answers from a post-test given to CS3114 students at Virginia Tech during Fall 2014. We present the validation results of our initial misconception list as evaluated by a panel of experts. Third, we present the process of creating the CI items based on the identified misconceptions. Then we present results from testing the validity and reliability of the developed CI items as revealed from administering the CI at two different universities during two subsequent semesters.

4.2 Concept Selection

A CI is not intended to be a comprehensive examination on the topic [11]. Its purpose is not to determine level of knowledge, but rather to identify whether or not a student has any of the identified misconceptions. Accordingly, a CI should mainly focus on those concepts identified to be difficult and important in a particular course. To our knowledge, there is nothing in the literature that talks about difficult and important algorithm analysis topics for a typical CS3-level course. Accordingly, based on a long teaching experience and the analysis of DSA textbooks (e.g., [76]), we have identified an initial list of potential topics to

focus on in the CI.

In order to evaluate these concepts in terms of importance in the course and difficulty to students, we have conducted a Delphi process [16] to consult a panel of experts to provide ratings for the concepts based on their importance and difficulty. The Delphi process has been applied previously in developing several CIs to obtain a consensus among a group of experts through informed decisions [23, 64].

For our study, we received feedback from 10 experts from different institutions and countries (we have some from the US, others from Europe, and one from the Middle East), each with years of experience in teaching CS3-level courses. Some of them also have authored textbooks for DSA courses.

The applied delphi process has three main phases. Figure 4.1 illustrates the process.

1. **Phase 1: Initial Rating**
   We sent the experts (through email) our initial list of concepts along with a detailed explanation of what each concept means. We asked them to rate the concepts according to their perceived difficulty to their students and importance in the course. The ratings are from 0 to 10, where 0 means not at all important or difficult, and 10 means very important or difficult. In addition, each expert was asked to add more concepts to our initial list if he believes that it should be added. Phase 1 ratings are presented in Table 4.1. In this table the medians of all 10 raters are presented along with the Inter-Quartile Range (IQR).
2. **Phase 2: Negotiation**
   The results from Phase 1 were sent to the experts along with a modified list of concepts to reflect those concepts added by experts in Phase 1. Similarly, the experts are required to provide two ratings for each concept according to its difficulty and importance in the light of the results from Phase 1. This phase is named negotiation as each expert can now see the overall evaluation of other experts in terms of medians and IQRs, and this can affect an expert's ratings in this phase. In addition, experts were asked to provide written justifications if their ratings for a concept falls outside the IQR presented for that concept. Results from Phase 2 ratings are presented in Table 4.2.
3. **Phase 3: Final Rating**
   The experts were presented with the results from Phase 2 along with the list of anonymized justifications provided by those experts who provided ratings outside the IQR. Experts are asked to provide their final ratings in the light of the results from Phase 2 and the list of justifications. Results from Phase 3 ratings are presented in Table 4.3. We found that only 3 experts changed their ratings a bit in this phase, in a way that did not change the overall medians and IQRs for any concept. Accordingly, we concluded with the same results as in Phase 2.

The ratings from the final phase were used to identify those concepts that are both important and difficult to be included in the CI. Figure 4.2 shows a scatter plot of the importance median against the difficulty median of each concept according to the ratings from Phase 3. The

lines in the figure represent the medians (with respect to all concepts) of the importance and the difficulty medians. We identify the Region Of Interest (ROI) as those concepts whose importance and difficulty medians equal to or exceed both the overall importance and difficulty medians of all the concepts. The red rectangle identifies our ROI (C2, C11, C12, C13, C16, C17, C19, C20, C21, C24, C26). In addition, we were also interested in four points close enough to the initial ROI, those representing C7, C10, C18, and C22. We are interested in those concepts because they are targeted by one or more identified misconception as we will see in the next section.

However, addressing 15 concepts in a single CI is hard. We are restricted with time constraints when administering the CI. Designing items for 15 concepts can result in a lengthy CI that requires much time to administer. A typical CI should not require more than 30 minutes to complete without time pressure [4]. Accordingly, we have to remove some concepts from our ROI that we believe can be part of other CIs. These concepts are C2, C19, and C20. C2 is related to proofs by induction. While this is an important mathematical prerequisite for a CS3-level course, we believe that it is not an intrinsic concept in algorithm analysis. It may be included in other CIs. A potential CI item was included for inductive reasoning in the discrete math CI [4]. C19 and C20 are related to recursive analysis. C19 is mainly focusing on the students ability to think about data structures in a recursive manner (i.e. a linked-list is an empty list or a linked-list node attached to a linked-list). C20 is concerned with the ability to write and analyze recurrence relations. While we believe that these concepts are important and may be hard for some students, we think they may better be part of a recursion concept inventory [31]. In addition, we believe that C20 is not addressed in great detail in a typical CS3 course. For example, at Virginia Tech, this concept is addressed in a senior level DSA course and a graduate level algorithms course.

Based on this, we prefer to include CI items that targets the concepts in the initial ROI as well as C7, C10, C18, and C22, excluding C2, C19, and C20.

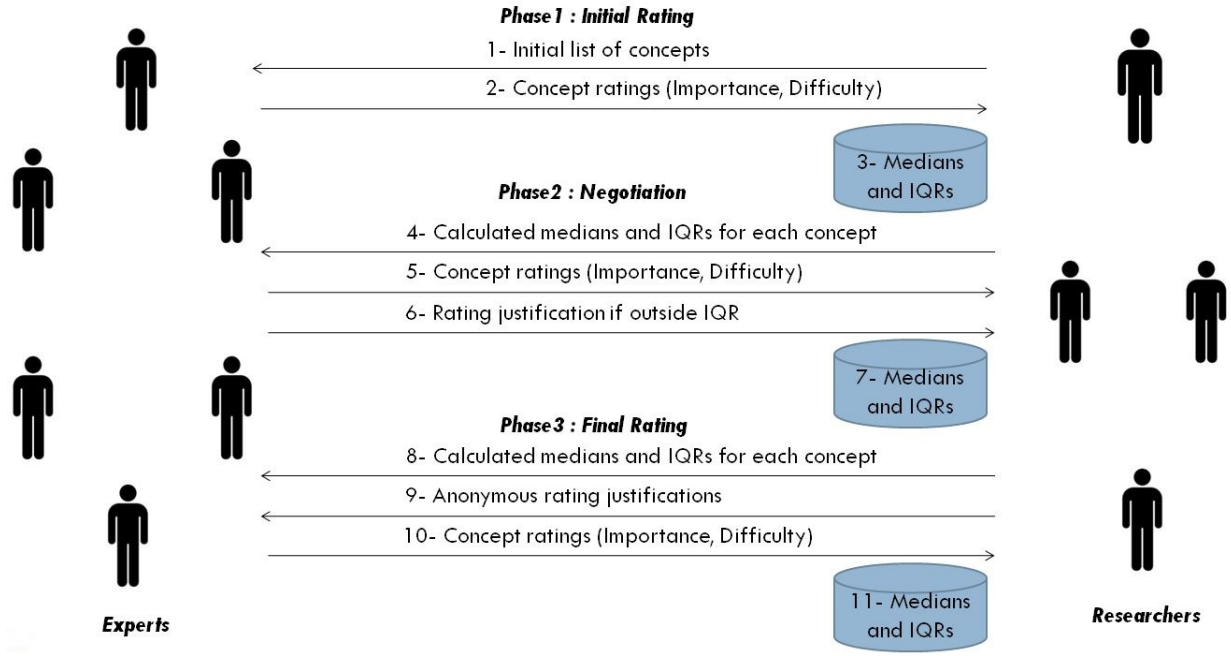


Figure 4.1: The Delphi Process

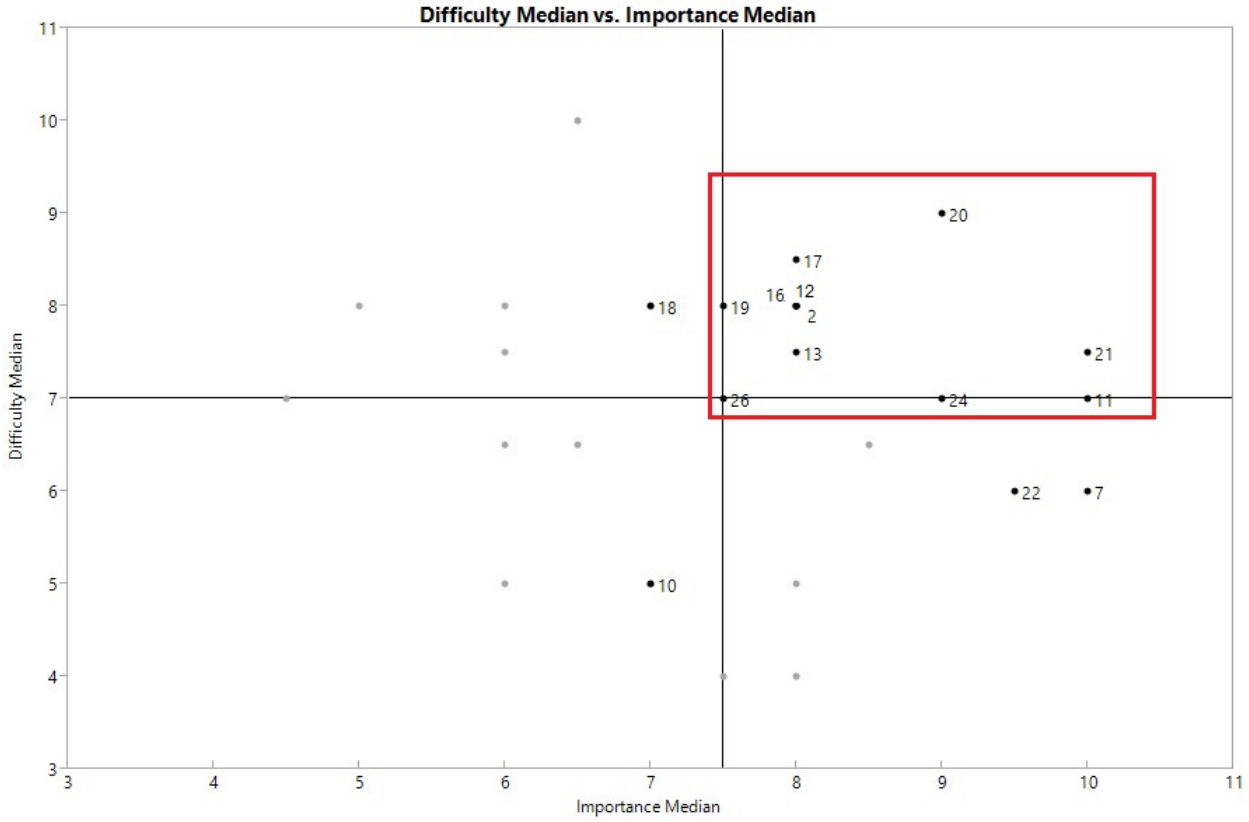


Figure 4.2: Importance versus difficulty

Table 4.1: List of concepts with their rating medians and IQRs from Phase 1. When the median appears as x|y, this indicates the middle two from an even number of values.

Concept	Importance median (IQR)	Difficulty median (IQR)
Mathematical Foundations		
1- Logs	8(5-10)	5(3-6)
2- Proofs by Induction	7|8(5-9)	8(7-9)
3- Proofs by contradiction	5(5-8)	6|7(5-9)
4- Limits	5|7(5-8)	5(4-7)
5- Summations	8(6-9)	4(3-6)
Growth rates of simple mathematical functions and their relationships		
6- Asymptotic growth	9|10(8-10)	5|6(4-8)
7- Inevitable crossing points	7(6-7)	5|6(5-6)
8- Effects of constants	8(6-9)	5(4-5)
9- Relative growth rates	7|8(7-9)	5(4-6)
Concepts and notation for asymptotic running time analysis		
10- Upper bounds (Big-$\mathcal{O}$)	10(9-10)	7(6-9)
11- Lower Bounds (Big-Ω)	8|9(7-10)	7|9(6-9)
12- Tight bounds (Big-Θ)	8(7-10)	8(4-9)
13- Comparing functions via limits	6|7(6-7)	5|6(5-8)
Upper and lower bounds of problems		
14- Problem upper bound	10(9-10)	7(6-9)
15- problem lower bound	8|9(7-10)	7|9(6-9)
16- Comparing upper and lower bounds of problems to upper and lower bounds of algorithms	8(7-10)	8(4-9)
Other topics		
17- Best, average, and worst case of an algorithm	9|10(9-10)	7|8(6-9)
18- Program efficiency and analyzing loop constructs	10(8-10)	6(5-7)
19- Analyzing space bounds of algorithms	8(7-9)	6(5-8)
20- Selecting an algorithm based ona given use-case scenario	8|9(7-9)	6|7(5-9)
21- Multi-parameter analysis	5|7(5-7)	6|7(6-8)

Table 4.2: List of concepts with their rating medians and IQRs from Phase 2. When the median appears as x|y, this indicates the middle two from an even number of values. Suggested concepts by experts from Phase 1 are italicized.

Concept	Importance median (IQR)	Difficulty median (IQR)
Mathematical Foundations		
C1- Logs	7|8(7-10)	4(4-5)
C2- Proofs by Induction	8(7-9)	8(8-9)
C3- Proofs by contradiction	6(5-8)	6|7(6-8)
C4- Limits	6(5-7)	5(5-6)
C5- Summations	8(8-9)	4(3-5)
C6- *Recurrence relations*	7(6-8)	8(7-8)
Growth rates of simple mathematical functions and their relationships		
C7- Asymptotic growth	10(9-10)	6(4-8)
C8- Inevitable crossing points	7(6-8)	5(5-6)
C9- Effects of constants	8(8-9)	5(4-5)
C10- Relative growth rates	7|8(7-9)	5(5-5)
Concepts and notation for asymptotic running time analysis		
C11- Upper bounds (Big-$\mathcal{O}$)	10(9-10)	7(6-9)
C12- Lower Bounds (Big-Ω)	8|9(7-10)	7|9(6-9)
C13- Tight bounds (Big-Θ)	8(7-10)	8(4-9)
C14- *Little o and* ω	4|5(2-6)	5|6(5-8)
C15- Comparing functions via limits	6|7(6-7)	6|7(5-8)

Concept	Importance median (IQR)	Difficulty median (IQR)
Upper and lower bounds of problems		
C16- Problem upper bound	8(8-9)	8(6-8)
C17- problem lower bound	8(7-9)	8|9(7-9)
C18- Comparing upper and lower bounds of problems to upper and lower bounds of algorithms	7(7-8)	8(7-9)
Recursive analysis		
C19- *Defining data structures recursively*	7|8(6-9)	7|8(8-9)
C20- *Writing and analyzing recursive functions*	9(7-10)	9(8-9)
Other topics		
C21- Best, average, and worst case of an algorithm	9|10(9-10)	7|8(6-9)
C22- Program efficiency and analyzing loop constructs	10(8-10)	6(5-7)
C23- Analyzing space bounds of algorithms	8(7-9)	6(5-8)
C24- Selecting an algorithm based on a given use-case scenario	8|9(7-9)	6|7(5-9)
C25- Multi-parameter analysis	5|7(5-7)	6|7(6-8)
C26- *Space/time tradeoffs*	7|8(6-9)	6|8(5-8)
C27- *Empirical analysis and code tuning*	4(4-7)	7|8(5-8)
C28- *Amortized analysis*	6(6-8)	8(7-9)
C29- *NP-Completeness*	6|7(5-8)	10(9-10)

Table 4.3: List of concepts with their rating medians and IQRs from Phase 3. When the median appears as x|y, this indicates the middle two from an even number of values. Concepts included in our ROI are emphasized.

Concept	Importance median (IQR)	Difficulty median (IQR)
Mathematical Foundations		
C1- Logs	7|8(7-10)	4(4-5)
C2- Proofs by Induction	8(7-9)	8(8-9)
C3- Proofs by contradiction	6(5-8)	6|7(6-8)
C4- Limits	6(5-7)	5(5-6)
C5- Summations	8(8-9)	4(3-5)
C6- Recurrence relations	7(6-8)	8(7-8)
Growth rates of simple mathematical functions and their relationships		
C7- **Asymptotic growth**	10(9-10)	6(4-8)
C8- Inevitable crossing points	7(6-8)	5(5-6)
C9- Effects of constants	8(8-9)	5(4-5)
C10- **Relative growth rates**	7|8(7-9)	5(5-5)
Concepts and notation for asymptotic running time analysis		
C11- **Upper bounds (Big-$\mathcal{O}$)**	10(9-10)	7(6-9)
C12- **Lower Bounds (Big-Ω)**	8|9(7-10)	7|9(6-9)
C13- **Tight bounds (Big-Θ)**	8(7-10)	8(4-9)
C14- Little o and ω	4|5(2-6)	5|6(5-8)
C15- Comparing functions via limits	6|7(6-7)	6|7(5-8)

Concept	Importance median (IQR)	Difficulty median (IQR)
Upper and lower bounds of problems		
C16- **Problem upper bound**	8(8-9)	8(6-8)
C17- **problem lower bound**	8(7-9)	8|9(7-9)
C18- **Comparing upper and lower bounds of problems to upper and lower bounds of algorithms**	7(7-8)	8(7-9)
Recursive analysis		
C19- Defining data structures recursively	7|8(6-9)	7|8(8-9)
C20- Writing and analyzing recursive functions	9(7-10)	9(8-9)
Other topics		
C21- **Best, average, and worst case of an algorithm**	9|10(9-10)	7|8(6-9)
C22- **Program efficiency and analyzing loop constructs**	10(8-10)	6(5-7)
C23- Analyzing space bounds of algorithms	8(7-9)	6(5-8)
C24- **Selecting an algorithm based on a given use-case scenario**	8|9(7-9)	6|7(5-9)
C25- Multi-parameter analysis	5|7(5-7)	6|7(6-8)
C26- **Space/time tradeoffs**	7|8(6-9)	6|8(5-8)
C27- Empirical analysis and code tuning	4(4-7)	7|8(5-8)
C28- Amortized analysis	6(6-8)	8(7-9)
C29- NP-Completeness	6|7(5-8)	10(9-10)

4.3 Identifying Misconceptions

Previous studies within CS mainly focused on identifying misconceptions related to programming [42], object oriented programming [37], operating systems [89], and digital logic [33, 35]. Little attention has so far been paid to misconceptions related to analyzing a problem structure and solution [17]. There is some research literature that tried to identify misconceptions related to algorithm efficiency. However, efficiency here does not mean the efficiency in an asymptotic sense (i.e., two algorithms may be in $O(n)$ but one algorithm is better than the other within a constant factor). For example in [21], a study was conducted to reveal misconceptions in perceiving the efficiency of algorithms by high school students. This study revealed the more basic misconceptions as: (Misconceptions are presented along with the concepts they are related to in braces)

M1- Shorter programs are more efficient. (C22)
M2- Programs containing less variables are more efficient. (C22)
M3- Programs having the same statements have the same efficiency even if the statements are in a different order. (C22)
M4- Two programs performing the same task are equally efficient. (C22)

A follow-up study [67] sought to determine if these four misconceptions are held by high school and early university students. The study revealed that both high school students and university students have the same misconceptions related to algorithm efficiency.

To our knowledge, there is no formal attempt in the literature to identify student misconceptions related to asymptotic analysis of algorithms, upper bounds, lowers bounds, tight bounds, analyzing problems, and relative growth rates. However, an initial list of such misconceptions was found in the OpenDSA eTextbook [77]. These misconceptions were revealed as a result of more than 25 years of the author's experience teaching algorithm analysis for CS3 students. An adapted list of these misconceptions include:

M5- Students are confused between the concepts of upper and lower bounds. (C11)(C12)(C16)(C17)
M6- Students are confused how to use upper bound (Big-$\mathcal{O}$), tight bound (Big-Θ), and lower bound (Big-Ω) notations. (C11)(C12) (C13) (C16)(C17)
M7- Students confuse the concepts and notation of upper bounds on one hand versus worst case on the other hand. (C11)(C21)
M8- Students confuse the concepts and notation of a lower bound on one hand versus best case on the other hand. (C12)(C21)
M9- Students confuse the concepts and notation of a tight bound on one hand versus average case on the other hand. (C13)(C21)
M10- Students think that the best case will occur when the input size is as small as possible, and that the worst case will occur when the input size is as large as possible. (C21)
M11- Students are confused about the distinction between the upper and lower bounds of an algorithm on one hand versus the upper and lower bounds of a problem on the other.

(C18)

Based on our own observations from a post-test analysis given to 53 students of CS3114 at Virginia Tech during Fall 2014, we have identified these additional misconceptions that we believe are held by some students.

M12- Poor intuition for the relative differences in common growth rates of algorithms. Most importantly, they have poor intuition for the difference between n^2 vs. $n \log n$ on the one hand and $n \log n$ vs. n on the other. (C7)(C10)

M13- A weak grasp of logarithms. One way that this expresses itself is that they do not recognize that n versus $\log n$ is the same relationship as 2^n versus n. (C7)(C10)

M14- Confusion about when to add and when to multiply when looking at (for example) loops for a function. One way that this expresses itself is to think that something costs n! (because they take the multiplication of i from 1 to n) instead of n^2 (because they should sum i from 1 to n). (C22)

M15- Belief that loops always have cost that is linear on the maximum size of the control variable. That is, they assume that the cost is always the same as if they were incrementing/decrementing the control variable by one. This is incorrect when, for example, the loop control variable is doubled or halved on each iteration. (C22)

M16- An algorithm with less number of loops is always more efficient. (C22)

M17- Cannot find and/or differentiate between the best, average, and worst case scenarios of a particular algorithm. (C21)

As we see from the initial list of misconceptions, each concept from our ROI is covered at least once in the list. Accordingly, this list can be used as the basis for an initial CI items. But before this step, it is necessary to validate this list through our Delphi experts. Accordingly, we have asked the experts to provide us their feedback about our initial misconceptions list. We believe this to be an important and a necessary step before item creation, since based on their long teaching experience, they should have an opinion regarding whether they feel each misconception is held by their students. Accordingly, each expert was required to rate each misconception from our list according to whether it is more important, important, or should be removed. Results from these ratings are summarized in Table 4.4.

As we see from the table, all the misconceptions except M2 and M3 have at least 8 experts who have rated it as more important or important. Accordingly, we believe that our CI items should properly address these misconceptions. For M2 and M3 we have 6 and 4 experts who have suggested the removal of these two misconceptions from the list, respectively. However, before taking any decision to remove any misconception from the list, we have validated the list based on real student answers from four administrations of the initial CI. Results from these adminstrations are presented in Section 4.6.2 as part of the student content validation of the CI. Results revealed that a sufficient number of students held most of the misconceptions identified in the list. The only exception was for M2, in which we didn't find any evidence for its existence within student answers. Accordingly, we decided to remove it from the list.

Table 4.4: Expert feedback on the initial list of misconceptions. Each Misconception is presented along with how many expert rated it as a more important, important, or should be removed.

Misconception	**More Important**	**Important**	**Should be Removed**
M1	1	7	2
M2	1	3	6
M3	1	5	4
M4	4	5	1
M5	4	6	0
M6	7	3	0
M7	7	3	0
M8	7	3	0
M9	3	6	1
M10	5	3	2
M11	4	5	1
M12	5	5	0
M13	6	4	0
M14	2	7	1
M15	3	7	0
M16	2	6	2
M17	4	6	0

4.4 Item Creation

A CI should be concise and capable of probing the student misconceptions of core concepts [4]. Accordingly, based on the list of misconceptions defined in Section 4.3, a set of initial CI items was developed. The initial items are a mix of MCQs with justification, True and False with justification, and open ended questions. While items in most CIs are MCQ items, open ended questions may be used to reveal other misconceptions that were not initially captured [11, 34]. In addition, a student's justification for his answer to an MCQ or True and False question may be helpful as a supporting evidence that he is holding the misconception targeted by this question. It can also be helpful in revealing new misconceptions as for open ended questions. Next we present some item examples.

- Below are two Java functions to compute $(x^y)^z$.

Function A

```
public int TwoPowers(int x, int y,
int z) {
  int i, result = 1;
  for (i = 1; i <= y * z; i++) {
    result *= x;
  }
  return result;
}
```

Function B

```
public int TwoPowers(int x, int y,
int z) {
  int i, result = 1;
  int finalResult = 1;
  for (i = 1; i <= y; i++) {
    result *= x;
  }
  for (i = 1; i <= z; i++) {
    finalResult *= result;
  }
  return finalResult;
}
```

Which function, A or B, do you think usually runs faster? Justify your answer.
(a) Function A usually runs faster.
(b) Function B usually runs faster.
(c) Both functions take about the same amount of time.

This item presents two algorithms implemented as two Java functions to compute the value of $(x^y)^z$. Function A is shorter than Function B and it contains only one for loop. Accordingly, any student with misconceptions M1 and M16 will be inclined to pick answer (a). The student's justification should make it clear about which misconception exactly is held. In addition, a student with misconception M4 will be inclined to pick answer (c) as both algorithms are doing the same task but in different ways. Note that the difference in efficiency between both algorithms is not with a constant factor (i.e., A is in $O(y * z)$ and B is in $O(y + z)$).

- Provide the running time of the following code snippet.

```
for(i = 1; i <= n; i++)
  for(j = 1; j <= n; j = j * 2)
    System.out.print(i * j);
```

This item asks the student to write down the asymptotic (i.e., in terms of Θ) running time of the presented code snippet. This is an example of an open ended item. A student having misconception M15 will provide an answer in $\Theta(n^2)$, ignoring that the second loop steps in multiples of 2. In addition a student with misconception M14 will provide an answer in $\Theta(n!)$ or $\Theta(n^n)$.

- Which of the following statements is/are true?
  (I) The relationship between $n/n\log_2 n$ is the same as the relationship between $n\log_2 n/n^2$.
  (II) The relationship between $\log_2 n/n$ is the same as the relationship between $n/2^n$.
  (a) (I) Only.
  (b) (II) Only.
  (c) (I) and (II).
  (d) Neither (I) nor (II).

This item tests the student's ability in understanding the relative growth rates of simple mathematical functions. It targets misconceptions M12 and M13. A student with misconception M12 will be inclined to pick answer (c) as he will fail to detect that statement (II) is correct. A student with misconception M13 will be inclined to pick answer (d) for the same reason. A student with both misconceptions will be inclined to pick answer (a).

- Mark all statements below about the upper, lower, and tight bounds of an algorithm as True or False. For any false statement, either correct it or explain why it is false.
  (a) The upper bound of an algorithm is the growth rate that the algorithm has in its worst case.
  (b) The lower bound of an algorithm is the growth rate that the algorithm has in its best case.

These two items test the ability of the student to clearly distinguish between the concepts of cases (best and worst), and the concepts of bounds (upper and lower). A student with misconception M7 will answer item (a) as True as he is confused between the concepts of upper bound and worst case. In addition, a student with misconception M8 with answer item (b) as True as he is confused between the concepts of lower bound and best case.

In a similar manner a list of 10 questions was developed. Some of them are divided to subquestions, for a total of 29 items. Each item addresses one or more student misconceptions from our initial misconception list defined in Section 4.3. The whole set of items are presented in Appendix B. Table 4.5 presents the concepts and misconceptions addressed by each CI item.

Table 4.5: Concepts and Misconceptions covered by each CI item.

Item	**Concepts**	**Misconceptions**
Item 1	C22	M1, M2, M4, M16
Item 2	C22	M3, M4, M16
Item 3(a)	C11	M7
Item 3(b)	C12	M8
Item 3(c)	C13	M5
Item 3(d)	C11	M5
Item 3(e)	C12	M5
Item 3(f)	C13	M9
Item 3(g)	C11, C12	M7, M8
Item 4(a)	C21	M17
Item 4(b)	C21	M10
Item 4(c)	C21	M10
Item 4(d)	C21	M17
Item 5(a)	C16	M7
Item 5(b)	C17	M8
Item 5(c)	C16, C17	M5
Item 5(d)	C17, C18	M11
Item 6(a)	C22	M14, M15
Item 6(b)	C22	M14, M15
Item 6(c)	C22	M14, M15
Item 7(a)	C7, C10, C11	M6
Item 7(b)	C7, C10, C11	M6
Item 7(c)	C7, C10, C12	M6
Item 7(d)	C7, C10, C13	M6
Item 8	C7, C10	M12, M13
Item 9(a)	C7 C10 C11 C12	M6
Item 9(b)	C7 C10 C11 C12	M6
Item 9(c)	C7 C10 C11 C12	M6
Item 10	C21, C24, C26	M17

It is clear from the table that each concept from our ROI is covered in the CI at least once, as well as all of the misconceptions from our initial misconceptions list. There is no direct relationship between the difficulty or importance of a concept and the number of times it appears in an item. What motivated the number of questions per concept are the number of misconceptions for that concept, and the number of items we believe a specific misconception can be revealed with.

4.5 AACI Administration

In order to test the validity and reliability of our pilot AACI items, the AACI was administered four times in two different universities (Virginia Tech and Christopher Newport university) during Fall 2015 and Spring 2016. For both semesters in Virginia Tech (VT), the CI was given as part of the final in a CS3-level course. The overall time of the final was 2 hours, and we believe that most of the students should need no more than 30 minutes solving the AACI items. For both semesters in Christopher New Port (CNP), the students of a CS3 level course were given 30 minutes to solve the AACI given as a post-test during the last week of the semester directly before their final exam.

23 items were identical across the four administrations. 6 items were a little bit different. The main difference was in which experimental items were included and a little bit of wording change. However, all the items in the four administrations were at the same difficulty and were testing the same concept(s) and misconception(s). Accordingly, we believe that all four administrations were testing a consistent set of items. Table 4.6 presents some summary statistics for the AACI administrations.

Table 4.6: Summary statistics of AACI administrations.

Metric	CNP Fall 2015	VT Fall 2015	CNP Spring 2016	VT Spring 2016
N	40	67	32	155
Mean	12.30	16.52	13.21	20.87
Standard Deviation	3.9	4.31	4.28	4.06
Median	12.5	16	13	21
Minimum	6	9	6	10
Maximum	22	26	26	28
Distribution	Normal	Normal	Normal	Normal

4.6 AACI Reliability and Validity

We have checked the reliability and validity for our pilot AACI through several methods.

4.6.1 Reliability

To test the reliability or our initial AACI, we have adopted single administration measures based on items' internal consistency [3]. We have calculated one Cronbach's alpha coefficient for each of the four administrations as well as an overall coefficient for all four administrations.

Cronbach's alpha was calculated based on the following equation*:

$$\alpha = \frac{K}{K-1}(1 - \frac{\sum_{i=1}^{K} \sigma_{Y_i}^2}{\sigma_x^2})$$

Here K is the number of items in the instrument, $\sigma_{Y_i}^2$ is the variance of the students' scores on item Y_i, and σ_x^2 is the variance of the scores on the whole instrument. Table 4.7 summarizes these results based on the following equation:

Table 4.7: Cronbach's-alpha coefficient for AACI.

	CNP Fall 2015	VT Fall 2015	CNP Spring 2016	VT Spring 2016	All
α	0.66	0.75	0.68	0.73	0.82

According to [66], a Cronbach's alpha coefficient of 0.7 or higher can be a good indicator of a reliable CI. We see from the table that the AACI has a good reliability coefficient except for the CNP administrations. However, this is not so bad as the coefficients are above 0.65 and close to 0.7. We believe the smaller coefficients for the CNP administrations may be attributed to the small number of students who were given the CI (40 for Fall 2015 and 32 for Spring 2016). The coefficients were good for both VT administrations as they are both above 0.7. The best result is from the overall coefficient. Here we only calculated the reliability for those identical 23 items across the four administrations with a total of 294 students.

Overall, it is clear from the table that the AACI items have a good reliability. In other words, there is an acceptable internal consistency among test items, as they are measuring the same construct or trait.

4.6.2 Validity

Reliability is necessary but not a sufficient condition to show that a CI is valid [34]. Given the positive results we had from the AACI reliability testing, we can now check its validity. We have adopted two methods of validation. Expert content validity and student content validity [34].

Expert Content Validity

In order to check for face content validity [3], we have consulted a panel of 11 experts (one expert was added to our original panel used in the Delphi process in Section 4.2), and

*https://en.wikipedia.org/wiki/Cronbachs_alpha

asked them to give us feedback about our pilot version of the CI items. This validation step was done before administering the AACI so that any expert feedback can be taken into consideration before giving the instrument to students. Several criteria were taken into account to select each expert and include him in our experts panel. Each expert should have years of experience as an instructor of a CS3-level course. Each expert should have published textbooks or at least have a good publication record in computer science education. Experts were also chosen to be diverse in race, geographical location, and type of institution. A form was sent to each expert via email containing all 10 questions of our AACI including sub-questions for a total of 29 items along with their rubrics and the misconception list. We asked each expert to read each CI item carefully and write his or her opinion on whether this is a good item, needs some sort of revision, or doesn't address an important concept or misconception. A space was available on the form for each expert to add additional concepts that should be addressed in the CI that we have missed. After that, we asked each expert whether he or she thinks that the CI as a whole would do a good job in identifying the key misconceptions an average CS3-level student has. Finally, each expert was asked whether he or she thinks that a typical student who does well on this test is likely to have mastered the key concepts related to algorithm analysis as taught in a typical CS3-level course, and conversely, whether he or she thinks that doing poorly on this test will accurately indicate that the student does not have a good understanding of algorithm analysis topics. Table 4.8 presents the feedback we received from experts regarding the quality of each CI item.

Table 4.8: Expert feedback on CI items. Each item is presented along with how many expert rated it as a good item, needs revision, or doesn't address important concept or misconception.

Item	**Good Item**	**Needs Revision**	**Doesn't Address Important Concept or Misconception**
Item 1	10	1	0
Item 2	9	2	0
Item 3	9	2	0
Item 4	9	2	0
Item 5	10	0	1
Item 6	11	0	0
Item 7	11	0	0
Item 8	8	3	0
Item 9	10	1	0
Item 10	8	2	1

It is clear from the table that for each item, most of the experts (at least 8 out of 11) found the item to be of good quality, that addresses important concept(s) as well as important misconception(s). However, some of the experts suggested some modifications, while a few

of them suggested removing or replacing the item. For example, one of the experts suggested refactoring Item 5 (this item tests problem upper bounds and lower bounds concepts) since he found it to be generic and that it should be written in a particular context (e.g., sorting problems). One of the experts also suggested removing Item 10 from the CI (this item tests the ability of students of selecting an algorithm according to a given use-case scenario), since he feels that this sort of question doesn't address an important concept that he is expecting students to learn. However, based on our Delphi ratings as shown in Table 4.2, this is an important and difficult concept that lies in our concept ROI. Most of the modifications suggested were just rewording the question or some changes to the written algorithms so that the question can be more clear for students. All of these suggestions were addressed before administering the pilot AACI.

Nine experts clearly stated that the pilot AACI as a whole will be a good instrument in revealing key misconceptions that a typical CS3 student has, and that the performance of students on this test clearly determines to what extent they have mastered the key concepts related to algorithm analysis as taught in a typical CS3-level course. One expert mentioned that the CI items need some probing. Another one mentioned that it is hard to say that this is a good CI, since it doesn't contain enough questions. We can reply to this by saying that a CI is not intended to be comprehensive by definition and that in administering the AACI, we are restricted with the amount of time that a typical student should need to answer the questions (typically 30-45 minutes). We cannot add more items without increasing the time for answering the AACI.

Student Content Validity

Because the AACI items were developed based on student misconceptions, one measure of validity is to check whether all misconceptions defined in our initial misconception list presented in Section 4.3 are expressed in the students' answers to the pilot AACI administrations. Table 4.9 and Figure 4.3, present the results of the AACI student content validation in terms of the number of students who demonstrated a misconception in their answer.

As we see from the results, our pilot AACI was able to detect almost all student misconceptions as defined in our misconception list. The only misconception we didn't find any supporting evidence for its existence within students answers is M2. This misconception states that students think that a program with less variables is more efficient than a program with more variables. This misconception was defined in [21] and [67] as one of the misconceptions held by high school and early university students in algorithm efficiency topics. But as we see here, there is no evidence as revealed from our pilot AACI that this misconception is held by CS3 students. This result coincides with what we found in Section 4.3 when we asked experts to evaluate our misconceptions list. 60% of the experts stated that this misconception should be removed from the list. In addition, [21] and [67] also defined M1, M3, and M4 as other misconceptions held by high school and early university

students in algorithm efficiency topics. However, we found only small evidence that these misconceptions are held by CS3 students. Only the answers of 2.4% of the students among all four administrations expressed misconceptions M1 and M3 and the answers of 2.7% of the students expressed misconception M4. While these are not significant compared to other misconceptions on the list, still some CS3 students are holding these misconceptions. Based on this, we have decided to remove M2 from our misconception list, while leaving M1, M3, and M4.

Table 4.9: Results of the AACI student content validity. Each misconception is presented along with how many students provided answers that demonstrated this misconceptions for each AACI administration. Here, A_1 stands for CNP Fall 2015 administration, A_2 stands for VT Fall 2015 administration, A_3 stands for CNP Spring 2016 administration, and A_4 stands for VT Spring 2016 administration. Percentages are shown in brackets.

Misconception	A_1 N = 40	A_2 N = 67	A_3 N = 32	A_4 N = 155	All N = 294
M1	4 (10%)	0	3 (9.3%)	0	7 (2.4%)
M2	0	0	0	0	0
M3	4 (10%)	0	0	3 (1.93%)	7 (2.4%)
M4	2 (5%)	3 (4.5%)	2 (6.2%)	1 (0.65%)	8 (2.7%)
M5	4 (10%)	8 (12%)	3 (9.3%)	40 (25.8%)	55 (18.7%)
M6	34 (85%)	29 (43.2%)	24 (75%)	87 (56.12%)	174 (59.1%)
M7	38 (95%)	60 (89.5%)	31 (96.8%)	72 (46.4%)	201 (68.3%)
M8	39 (97.5%)	57 (85%)	31 (96.8%)	67 (43.2%)	194 (66%)
M9	30 (75%)	32 (47.7%)	24 (75%)	54 (34.8%)	140 (47.6%)
M10	34 (85%)	42 (62.6%)	23 (71.8%)	46 (29.6%)	145 (49.3%)
M11	24 (60%)	31 (46.2%)	23 (71.8%)	19 (12.2%)	97 (33%)
M12	9 (22.5%)	11 (16.4%)	9 (28.1%)	30 (19.3%)	59 (20%)
M13	21 (52.5%)	34 (50.7%)	11 (34.3%)	22 (14.2%)	88 (30%)
M14	9 (22.5%)	4 (6%)	1 (3.1%)	19 (12.2%)	33 (11.2%)
M15	31 (77.5%)	9 (13.4%)	20 (62.5%)	12 (7.7%)	72 (24.4%)
M16	15 (37.5%)	4 (6%)	15 (46.8%)	10 (6.5%)	44 (15%)
M17	35 (87.5%)	36 (53.7%)	27 (84.3%)	40 (25.8%)	138 (47%)

On the other hand, we see that significant number of students provided answers supporting all the remaining misconceptions. The two most prevalent misconceptions are M7 (68.3%) and M8 (66%). These misconceptions are related to the students' confusion between the concepts of upper bounds versus worst cases on one hand, and lower bounds versus best cases on the other hand. The evidence for both misconceptions are clear from answering Items (a) and (b) from Question 3 as True, and items (a) and (b) from Question 5 as True (see Appendix B). In addition, we see that 47.6% of the students demonstrated M9 which

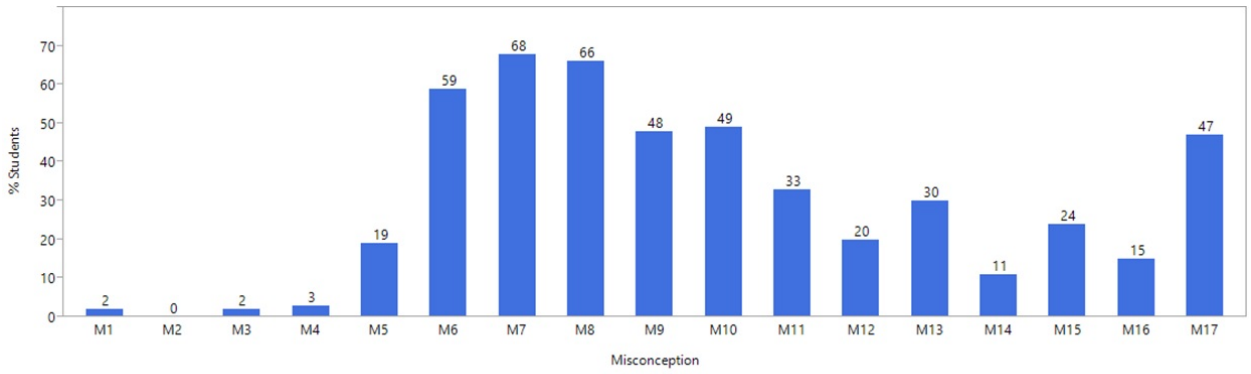


Figure 4.3: Results from the pilot AACI administrations as a post-test

is related to the students confusion between the concept of tight bound and average case. This is clear from the students answers to Item 3(f).

A significant fraction of students provided answers that demonstrated M6, which is related to confusion in applying upper bounds, lower bounds, and tight bounds notations. This is clear from student answers to Questions 7 and 9 related to relative growth rates. They used the incorrect notation to indicate the relationship between various growth rates.

We also see that the answers of 49.3% of the students provided evidence for M10, related to the students' confusion about the effect of the size of the input on whether an algorithm can run in its best or worst cases. This is clear from the answers to items 4(b) and 4(c). Students indicated that the best case of sequential search will occur when the array size is only a single element, while the worst case will occur when the array size is very large.

M17 is also one of the misconceptions that was significantly expressed in student answers (47%). This misconception is related to students' confusion about the best, average, and worst cases of a specific algorithm. This misconception was tested with Items 4(a), 4(d) and Question 10. In items 4(a) and 4(d), students were confused between the best case and worst case of sequential search. In Question 10, students were unable to determine the best sorting algorithm to use in a given use-case scenario. However, from the student answers, it seems like they would have answered the questions correctly if they had memorized the running time of the algorithms in its best and worst cases. For example, in Question 10, if the students knew that the worst case running time of heapsort is better than quicksort and it requires less memory than mergesort, then probably they would have answered the question correctly and the misconception wouldn't have been detected. The same for Items 4(a) and 4(d). If the students knew that the best case for sequential search is when the target element is located in the first position of the array, and the worst case is when the target element is located at the end of the array or not found, then they probably would have answered the items correctly and M17 would not have been detected.

Based on this analysis, we believe that our pilot AACI did a good job of detecting student misconceptions as defined in our list. However, we believe that we need better items for misconception M17 not based on memorization to be able to test whether this misconception is held by CS3 students.

4.7 Item Response Theory

Having evidence that the pilot AACI is reliable and valid, we now move to the evaluation of the AACI based on its items. Item Response Theory (IRT) is an approach to instrument analysis based on individual items rather than overall test scores. In this section, we present the results from IRT analysis of our AACI based on Item Characteristics Curves (ICCs) and the Test Information Function (TIF). ICCs plot the probability of a student to answer an item correctly (the y-axis) given an estimate of his overall ability (the x-axis). The TIF shows how reliable the test is at distinguishing students with different abilities. The results here only include those similar 23 items among all four AACI administrations. Figure 4.4 shows the ICC of each item, while Figure 4.5 shows the overall TIF. Table 4.10 shows the difficulty and discrimination coefficients of each item along with interpretation. All these results were calculated using JMP's item analysis support [†] by applying a Two-Parameter Logistic (2PL) IRT model [5]. The model's equation is given by:

$$p(\theta) = \frac{1}{1 + e^{-a(\theta - b)}}$$

This equation calculates the probability a student with ability θ can answer an item with discrimination a, and difficulty b.

The TIF in Figure 4.5 shows that the peak amount of information the whole test provides is around 0 ability. This indicates that the test did a good job in distinguishing between those students with above average ability (> 0) and those with below average ability (< 0).

The ICCs in Figure 4.4 show that not all items have this perfect S-shaped curve with an inflection point (intersecting the red line) around 0. Most of the curves have their inflection point below 0. This indicates that most of the test items are easy for students. This is not a problem as a CI is not intended to be hard. Its main purpose is to discriminate between those students with misconceptions and those without misconceptions.

[†] `http://www.jmp.com/support/help/Item_Analysis_Platform_Overview.shtml`

Table 4.10: Difficulty and Discrimination Coefficients of AACI items. Items with low discrimination indices are italicized.

Item	Difficulty	Discrimination	Interpretation
Item 1	-1.31	0.62	Easy item with low discrimination
Item 3(a)	0.50	1.39	Moderate item with high discrimination
Item 3(b)	0.40	1.29	Moderate item with Moderate discrimination
Item 3(c)	-1.26	0.60	Easy item with low discrimination
Item 3(d)	-6.72	0.20	Very easy item with very low discrimination
Item 3(e)	-1.99	0.68	Easy item with moderate discrimination
Item 3(f)	0.28	1.07	Moderate item with moderate discrimination
Item 3(g)	0.44	1.03	Moderate item with moderate discrimination
Item 4(a)	-3.28	0.75	Very easy item with moderate discrimination
Item 4(b)	-0.02	1.25	Moderate item with moderate discrimination
Item 4(c)	-1.54	1.30	Easy item with moderate discrimination
Item 4(d)	-3.85	0.66	Very easy item with moderate discrimination
Item 6(a)	-2.18	0.80	Easy item with moderate discrimination
Item 6(b)	-0.79	0.49	Moderate item with low discrimination
Item 6(c)	-0.44	1.19	Moderate item with moderate discrimination
Item 7(a)	-0.84	2.74	Moderate item with very high discrimination
Item 7(b)	-0.71	2.46	Moderate item with very high discrimination
Item 7(c)	-0.61	1.92	Moderate item with very high discrimination
Item 7(d)	-1.09	1.66	Moderate item with high discrimination
Item 8	1.05	-0.17	Above average item with very low discrimination
Item 9(a)	-0.17	8.62	Moderate item with very high discrimination
Item 9(b)	-0.24	4.53	Moderate item with very high discrimination
Item 9(c)	-0.04	4.55	Moderate item with very high discrimination

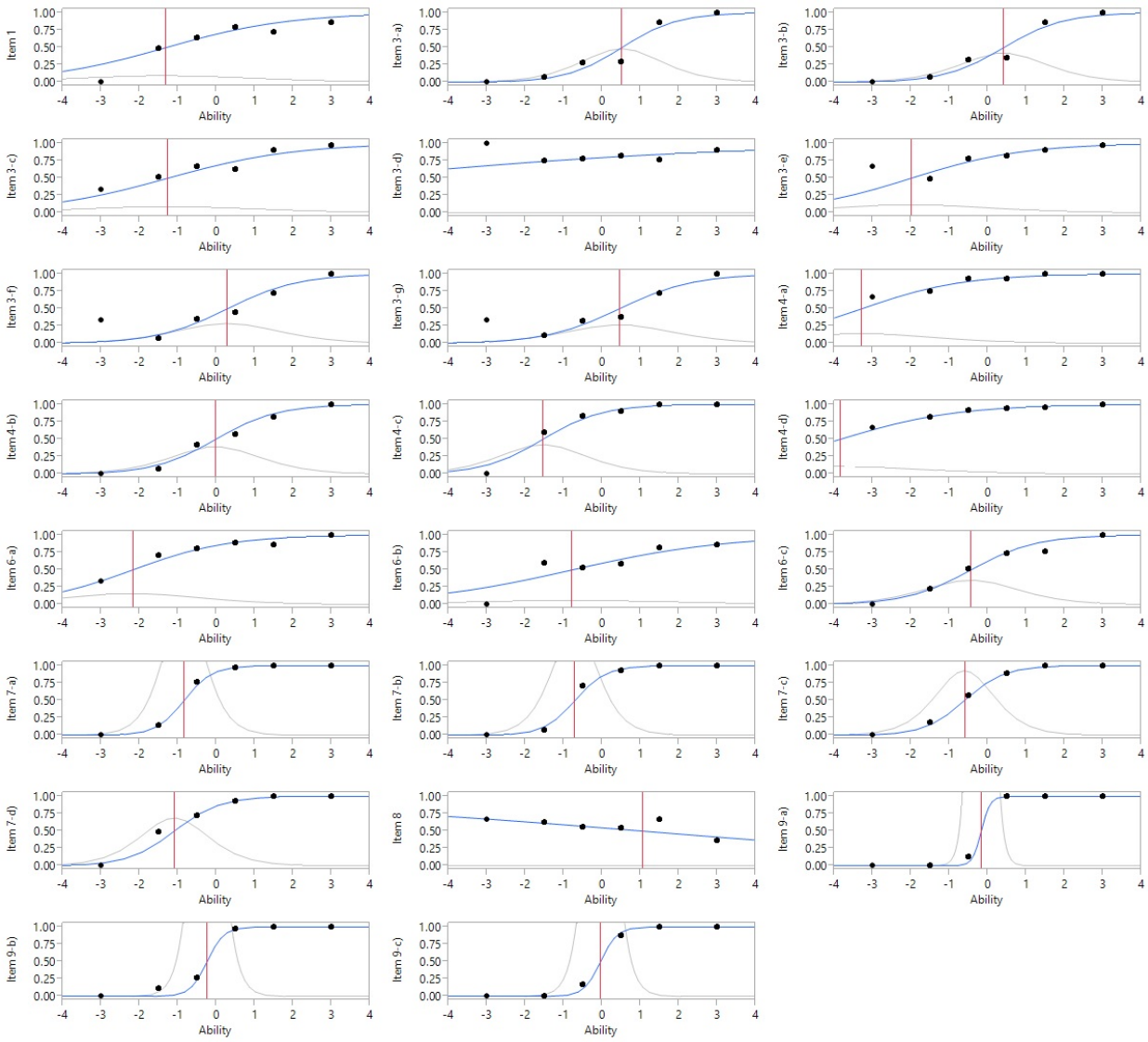


Figure 4.4: Item Characteristic Curves of AACI items

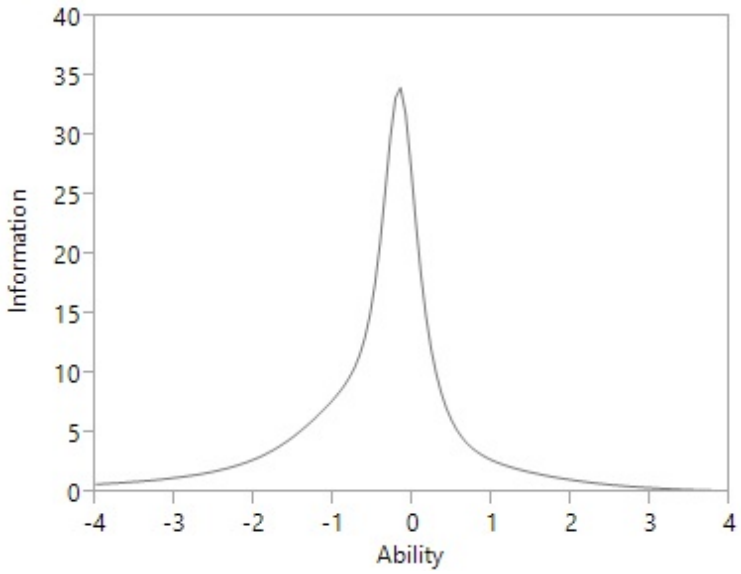


Figure 4.5: Test Information Function of the AACI

Table 4.10 provides estimations of ICC difficulty and discrimination coefficients along with interpretations. These interpretations are defined based on coefficient cut-off values defined in [5]. As we see, most of the items except items 8 and 3(d) are either easy or moderate items as they have their difficulty coefficients around or slightly above or below 0. Item 8 is slightly hard as it has a difficulty coefficient of 1.05, and item 3(d) is very easy as it has a difficulty coefficient of -6.72. Regarding discrimination, most of the items have discrimination coefficients above 0.64. According to [5], these items have moderate to high discrimination. Good discrimination is desirable in any CI as each item targets one or more misconception. Accordingly, if the item sufficiently discriminates between students, then this item will have a good contribution to the overall CI as a tool to group students into two groups: those who think in accordance to common conceptions, and those who think in accordance to common misconceptions within a topic. Only items 1, 3(c), 3(d), 6(b), and 8 have poor discrimination. In addition, item 8 has an interesting feature. It has a negative discrimination. This means that students with low ability tend to solve this item correctly better than those students with high ability. However, the discrimination is very low. Accordingly, this item is clearly a candidate for removal or update in future AACI versions in addition to items 1, 3(c), 3(d), and 6(b) as they have poor discrimination coefficients.

Chapter 5

Evaluation of Algorithm Analysis Visualizations

5.1 Introduction

In this chapter we describe our efforts to evaluate the effectiveness of AAVs when embedded in OpenDSA modules to replace the algorithm analysis content previously presented traditionally as text and static images. As described in Section 3.2, students using OpenDSA as their main textbook for CS3-level courses have been shown to not engage with the algorithm analysis content as traditionally presented. Student surveys include many comments stating the opinion that it would be helpful to Students if the content is presented in a more engaging way, as algorithm dynamics content is presented in OpenDSA. They suggested replacing the content with interactive visualizations. Accordingly, some of the original algorithm analysis content in OpenDSA was replaced with AAVs. This was done for the sorting chapter and some of the material in the introductory algorithm analysis chapter. Our evaluation relies on three facets:

1. **Evaluating student engagement:** For an intervention to be successful, it should be engaging enough to students so that they will be motivated to learn [18]. Accordingly, this is our first step in evaluating AAVs. The main focus here is to see whether students are more engaged with AAVs than with the traditional algorithm analysis content as presented previously in OpenDSA. The total time spent interacting with AAVs for each student in an entire semester was used as a proxy for engagement. This was calculated by mining the OpenDSA student interaction logs.
2. **Collecting student feedback about AAVs:** In order to support our results from student interaction logs, we surveyed the students to collect their feedback about AAVs in terms of whether they were engaging and helpful to them in understanding the algorithm analysis abstract concepts.

3. **Evaluating student performance** Engagement is a necessary but insufficient prerequisite for learning [28]. Accordingly, the last step in our evaluation is to evaluate student performance on tests. The algorithm analysis questions used on the final exam included our pilot AACI derived in the previous chapter.

The evaluation experiment was performed during Fall 2015 and Spring 2016 for a semester-long CS3-level course, CS3114 at Virginia Tech. Fall 2015 students were used as our control group (without AAVs), and Spring 2016 students were used as our intervention group (with AAVs). For both groups, we performed three evaluation activities. First, we mined OpenDSA interaction logs to compare student engagement. Second, we surveyed the students to collect their feedback on the analysis content without AAVs (Fall 2015) and with AAVs (Spring 2016). Finally, we compared the student performance on the final (for algorithm analysis questions only).

5.2 Evaluation Protocol

In this section, we outline our protocol for AAV evaluation in terms of student engagement, satisfaction, and performance.

1. **Hypotheses**
   We have three main hypotheses that we have tested through the evaluation.
   (a) Students in the intervention group will be more engaged with AAVs than the control group students were engaged with traditional algorithm analysis content.
   (b) Most students from the intervention group will express positive feedback regarding their experience with AAVs.
   (c) The performance of the intervention group on a set of algorithm analysis items offered on their final will be higher than the performance of the control group students on the same set of questions.
2. **Participants**
   Our subjects are 67 students who took CS3114 at Virginia Tech during Fall 2015 and 155 students who took the same course during Spring 2016. Only one section from Fall 2015 was selected, as this is the only section that used OpenDSA as the main textbook. For Spring 2016, both sections were included.
3. **Materials**
   (a) Both Fall 2015 and Spring 2016 groups used OpenDSA as their main textbook. The only difference was in the presentation of some of the algorithm analysis material in two chapters, the sorting and the algorithm analysis introduction chapters. All the algorithm analysis content in Fall 2015 are based on textual presentation. Some of the algorithm analysis content during Spring 2016 was presented interactively using AAVs.
   (b) In order to evaluate AAVs in terms of student learning, both groups were given a

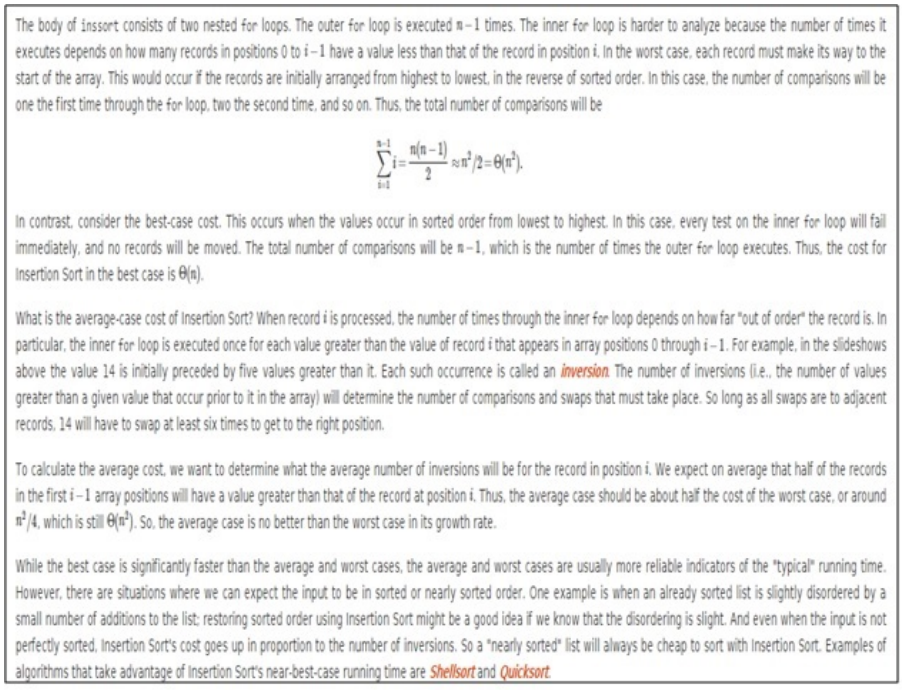
set of algorithm analysis questions as part of their final exam. The questions were part of our pilot AACI as described in the previous chapter (see Appendix B). The algorithm analysis part of the test for Fall 2015 was composed of 9 items; some were divided into sub-items for a total of 28 items. A similar test was offered in Spring 2016 with 11 items; some are divided into sub-items for a total of 29 items. Only 27 questions were similar in both tests. These questions form the basis of our performance comparison between both groups.

(c) In order to investigate student opinion about AAVs, we offered a survey at the end of Spring 2016 asking them to evaluate AAVs in terms of how useful they were in helping them understand the algorithm analysis concepts presented in the course. The survey is presented in Appendix C.

4. **Procedure**
   In order to evaluate student engagement, the total time spent on algorithm analysis content (a textual discussion for the control group, or an AAV for the intervention group) was used as a proxy for engagement. The total time was calculated by analyzing OpenDSA student interaction logs. In order to evaluate student performance, both groups were compared on the basis of the same set of algorithm analysis questions given as part of their final exam. Exams were graded by the same person for both groups in order to minimize inter-rater reliability problems.
5. **Analysis**
   In order to make inferences about our hypotheses, the control and intervention groups were compared according to the time spent in algorithm analysis material, and the exam grade for only the algorithm analysis part. We decided to use non-parametric tests (Mann-Whitney) with a 5% significance level. We chose this test to avoid the normality assumption that should be met if we would apply a parametric test (ANOVA), and to mitigate the effect of unbalanced sample sizes since Spring 2016 students are from two sections while Fall 2015 students are from one section. It was stated in [57], that Mann-Whitney test is robust to unbalanced sample sizes.

5.3 Evaluating Student Engagement

One way of evaluating student engagement is to mine student interaction logs from past course offerings [28]. Here we present our results from mining OpenDSA student interaction logs to compare the total time spent by CS3114 students on algorithm analysis material from both Fall 2015 and Spring 2016. The algorithm analysis content during Fall 2015 was presented as text and static images. Some of this content was replaced during Spring 2016 with AAVs. This was mainly done for the sorting modules and the algorithm analysis introduction modules. Figure 5.1 shows an example of the difference between the content for the two groups.



The body of `inssort` consists of two nested `for` loops. The outer `for` loop is executed $n-1$ times. The inner `for` loop is harder to analyze because the number of times it executes depends on how many records in positions 0 to $i-1$ have a value less than that of the record in position i. In the worst case, each record must make its way to the start of the array. This would occur if the records are initially arranged from highest to lowest, in the reverse of sorted order. In this case, the number of comparisons will be one the first time through the `for` loop, two the second time, and so on. Thus, the total number of comparisons will be

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx n^2/2 = \Theta(n^2).$$

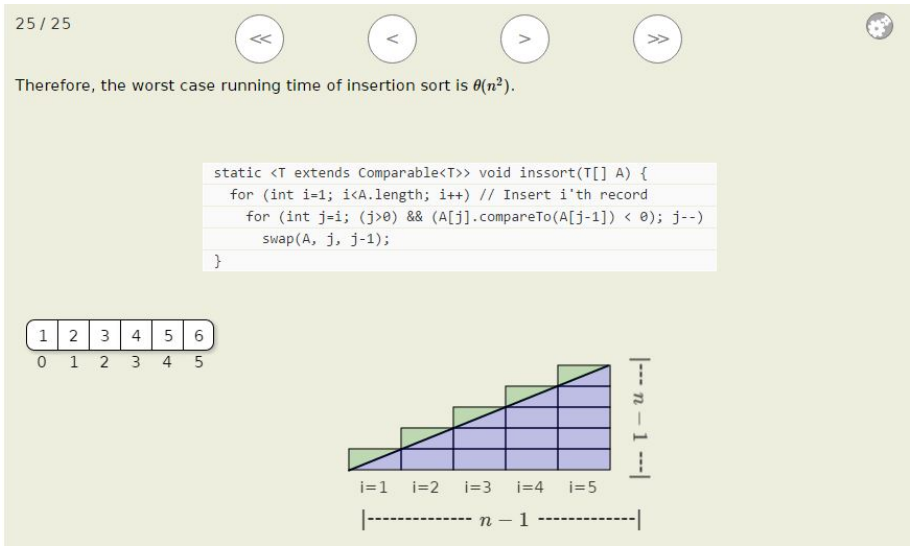
In contrast, consider the best-case cost. This occurs when the values occur in sorted order from lowest to highest. In this case, every test on the inner `for` loop will fail immediately, and no records will be moved. The total number of comparisons will be $n-1$, which is the number of times the outer `for` loop executes. Thus, the cost for Insertion Sort in the best case is $\Theta(n)$.

What is the average-case cost of Insertion Sort? When record i is processed, the number of times through the inner `for` loop depends on how far "out of order" the record is. In particular, the inner `for` loop is executed once for each value greater than the value of record i that appears in array positions 0 through $i-1$. For example, in the slideshows above the value 14 is initially preceded by five values greater than it. Each such occurrence is called an *inversion*. The number of inversions (i.e., the number of values greater than a given value that occur prior to it in the array) will determine the number of comparisons and swaps that must take place. So long as all swaps are to adjacent records, 14 will have to swap at least six times to get to the right position.

To calculate the average cost, we want to determine what the average number of inversions will be for the record in position i. We expect on average that half of the records in the first $i-1$ array positions will have a value greater than that of the record at position i. Thus, the average case should be about half the cost of the worst case, or around $n^2/4$, which is still $\Theta(n^2)$. So, the average case is no better than the worst case in its growth rate.

While the best case is significantly faster than the average and worst cases, the average and worst cases are usually more reliable indicators of the "typical" running time. However, there are situations where we can expect the input to be in sorted or nearly sorted order. One example is when an already sorted list is slightly disordered by a small number of additions to the list; restoring sorted order using Insertion Sort might be a good idea if we know that the disordering is slight. And even when the input is not perfectly sorted, Insertion Sort's cost goes up in proportion to the number of inversions. So a "nearly sorted" list will always be cheap to sort with Insertion Sort. Examples of algorithms that take advantage of Insertion Sort's near-best-case running time are *Shellsort* and *Quicksort*.

(a) Insertionsort analysis as presented during Fall 2015



(b) Insertionsort worst-case analysis as presented during spring 2016

Figure 5.1: Difference in the presentation of algorithm analysis content between Fall 2015 and Spring 2016

For the control group in Fall 2015, OpenDSA interaction logs were analyzed to find an estimate of the total time a student spent in the text-based algorithm analysis material in the sorting modules. Not all of the material in these modules is related to algorithm analysis, as it contains detailed discussion of the procedural dynamics of how the sorting algorithm works and then the running-time analysis of the algorithm is presented. We are only interested in the algorithm analysis part of the modules. Accordingly, the total time was calculated as the sum of the times the student spent in each reading session of the algorithm analysis part. We define a reading session as the difference between the time when the student pressed the show/hide button as described in Section 3.2.2, and the time the student performs any other interaction in the module (i.e., pressed the hide button, started solving the next exercise, left the module, or refreshed the page).

Similarly, for the intervention group in Spring 2016, OpenDSA interaction logs were analyzed to find an estimate of the total time a student spent interacting with the corresponding AAV(s). Again, the total time was calculated as the sum of the times the student spent in each interaction session. We define an interaction session as the difference between the time a student first clicked on a control in the AAV, and the time of the last interaction with the AAV.

We did the same for OpenDSA algorithm analysis modules. However, as all of the content in these modules are actually related to algorithm analysis, a different way of measuring the time seemed appropriate. Accordingly, for these modules the total time spent in the entire module for both groups was compared. For Fall 2015 students, the modules are presented without AAVs, and for Spring 2016 the modules are presented enhanced with some AAVs for part of the discussion. The total time spent in a module was calculated as the sum of the time the student spent in each module session. We define a module session as the difference between the time the student loaded the module (i.e., the document ready event is issued) and the time the student leaves the module (i.e., a window unload or window blur events are issued).

For some students, we found large time estimates exceeding tens of thousands of seconds, which we don't believe is a valid time for a typical student to spend in the text or AAVs. Our explanation for these cases is that some students started to interact with the material (the starting time is now recorded), and then moved away from the computer, leaving the browser opened. After a relatively long time they come back and start to interact again with the material until finishing with it (the end time is recorded). In order to perform valid analysis, we should discard all the students with such behavior from our evaluation. In order to do that, we ran a simple outlier analysis using the JMP* software package to detect those unusual points. For the sorting chapter, we found that all points exceeding half an hour were identified as outliers. This was only found for the intervention group students. Regarding the control group, we didn't find any student who exceeded this time. For the algorithm analysis chapter, all students who exceeded one hour per module were identified as outliers

*http://www.jmp.com/en_us/home.html

and excluded from both the control and intervention groups.

After calculating the time for both groups, we performed a series of Mann-Whitney tests to compare the total time spent for each student in an algorithm analysis textual discussion from the control group versus its corresponding AAV(s) from the test group for OpenDSA sorting modules. In addition, we performed a series of tests to compare the number of visits for each group (the number of visits is simply the total number of sessions a student read the text or interacted with an AAV). For the algorithm analysis introduction modules, a series of tests were performed comparing the total time spent for each student in the modules without AAVs (Fall 2015 control group) to the total time spent in the modules with AAVs (Spring 2016 test group). Results are presented in Table 5.1 for sorting modules and in Table 5.2 for the algorithm analysis introduction modules. The "exchange sorting cost" and the "sorting lower bound" modules are part of the sorting chapter, but their content is entirely related to algorithm analysis. Accordingly, the time spent using them was calculated in the same way as the algorithm analysis introduction modules.

Table 5.1: Comparing the total time in seconds spent in a textual discussion versus the total time spent in corresponding AAV(s) for OpenDSA sorting modules using a Mann-Whitney test. N here denotes the number of students who interacted with the material. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean		Median		p-value	
			Time	Visits	Time	Visits	Time	Visits
Insertionsort	F15	30	76.77	2.2	13.5	2	< 0.0001	< 0.0001
	S16	123	189.26	6.25	127	6		
Bubblesort	F15	28	23.57	1.35	12	1	< 0.0001	< 0.0001
	S16	137	78.96	2.65	52	2		
Selectionsort	F15	22	37.81	1.45	21.5	1	< 0.0001	< 0.0001
	S16	135	104.14	2.69	76	2		
Mergesort	F15	28	23	1.35	5.5	1	< 0.0001	< 0.0001
	S16	134	93.02	2.35	60	2		
Quicksort	F15	28	76.64	1.75	14.5	1.5	< 0.0001	< 0.0001
	S16	133	194.98	6.51	122	6		
Heapsort	F15	26	20.88	1.42	6.5	1	< 0.0001	0.0014
	S16	121	95.22	2.33	46	2		
Radixsort	F15	21	25.23	1.42	4	1	< 0.0001	0.0411
	S16	123	135.63	2.02	52	2		
The Cost of Exchange Sorting	F15	29	148.03	6.94	42	5	< 0.0001	0.4223
	S16	117	610.59	7	438	5		
Sorting Lower Bound	F15	27	78.55	4.77	17	5	< 0.0001	0.07
	S16	103	610.41	5.1	495	3		

Table 5.2: Comparing the total time in seconds spent in algorithm analysis introduction modules without AAVs versus the total time spent in the same modules with AAVs using a Mann-Whitney test. N here denotes the number of students who interacted with the material. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean		Median		p-value	
			Time	Visits	Time	Visits	Time	Visits
Asymptotic Analysis and Upper Bounds	F15	35	181.65	9.74	56	8	< 0.0001	< 0.0006
	S16	122	970.95	6.77	649	4		
Best, Average, and Worst Cases	F15	38	215.8	4.89	38	3.5	0.0037	0.0071
	S16	88	412.94	3.68	186	2		
Lower Bounds and Theta Notation	F15	33	358.72	12.72	160	8	< 0.0001	0.0054
	S16	120	1090.02	7.10	861	5		
Algorithm Analysis Misunderstandings	F15	35	92.48	6.34	14	4	< 0.0001	< 0.0001
	S16	140	462.57	3.61	324	2		
Problems,Algorithms, and Programs	F15	34	203.52	9.35	21.5	7.5	< 0.0001	0.0031
	S16	136	806.64	7.38	458	4		
Analyzing Problems	F15	34	78.23	6.91	20.5	5	< 0.0001	< 0.0001
	S16	126	502.14	4.10	335.3	3		
Calculating Program Running time	F15	37	601.02	12.81	371	11	< 0.0001	0.3136
	S16	118	1175.22	12.56	900	9		

It is clear from Table 5.1 that Spring 2016 students spent significantly more time interacting with AAVs than Fall 2015 students spent reading the corresponding textual content for all OpenDSA sorting modules. The results from Fall 2015 are similar to what we found in Section 3.2.2 from Fall 2014 data when we motivated the use of AAVs. In addition, we also see that Spring 2016 students tend to visit AAVs more than Fall 2015 students did with the textual content. This also may indicate some sort of engagement as students will not interact a lot of times with something boring (unless there is a problem with it that makes them confused). Accordingly, from this analysis, we believe that students are more engaged with AAVs than the textual content. This is supported by the student survey results as we will see in the next section.

Similarly, it is clear from Table 5.2 that Spring 2016 students spent significantly more time in the algorithm analysis modules than Fall 2015 students did. However, in most analysis modules, Fall 2015 students tend to visit the modules more times than Spring 2016 students. We were expecting to see similar results as found in the sorting modules in Spring 2016, as the analysis modules are enhanced with some AAVs which may engage students to spend more time in the modules and visit it more. What we found is that students spent significantly more time in the modules with AAVs, but they visit the modules significantly less number of times than the modules without AAVs. Our explanation to this is that Fall 2015 students

may have found the modules without AAVs boring. Accordingly, when they opened it they spent small amount of time and then close it or go to another module, or just jump to the summary exercise without reading the content. This is clear from the small amount of time spent in each module from Table 5.2. But since students will be tested on this content, they have to come back again and study the module. This is done in small time sessions as the modules are boring. This explains why we found more student sessions for the modules without AAVs but less total time. On the other hand, from Table 5.1, we see that Spring 2016 students spent more time in sorting AAVs and also they had more visits to them than Fall 2015 students did with the corresponding textual content. So why did we find this when comparing AAVs to text and we found the opposite when comparing the whole analysis modules with AAVs to the corresponding modules without AAVs? Our answer to this question is that an AAV is an interactive component that can by itself attract students and engage them to understand the material, and accordingly, they visit it more and spend more time on it. For the analysis modules, an AAV is only part of the whole module. In a typical module, AAVs represents only 10%-20% of the module content, and the rest can be boring for students. When they get the point from AAVs, they don't visit the module more times again. Accordingly, we believe that Spring 2016 students when opening the enhanced modules with AAVs, they are mainly interacting with the AAVs and ignoring the other textual content. The results from a study performed before adding AAVs to OpenDSA supports this, as it was found that when students open an OpenDSA module, they tend to jump directly to the summary exercise at the end of the module to secure their credit and they are using the interactive content (slide shows and AVs) only to find the exercise's answers [19]. In addition, the algorithm analysis modules in OpenDSA are more conceptual than the sorting modules, and we think that once students get the point from the AAVs in the analysis modules, they will not visit it again. But this is not the case in the analysis modules without AAVs, as each time they visit it, they found it hard to understand the concepts presented by reading the long text, so they leave the module and come back to it more times again trying to understand the content.

In order to further test the effect of AAVs on student engagement and to be more convinced that the difference in the time spent in the algorithm analysis modules between Fall 2015 and Spring 2016 students is attributed to AAVs, we did a similar analysis for three algorithm analysis modules in which we did not yet provide AAVs. The results are presented in Table 5.3.

It is clear from the table that there is no significant difference in the total time spent in algorithm analysis modules without AAV support between Fall 2015 students and Spring 2016 students. If we look at the number of Spring 2016 students who opened each module, we find that it is very small compared to the number of students who opened the modules with the AAV support from Table 5.2. This clearly indicates that students are not engaged in the modules without AAV support, and that the significant time difference found in Table 5.2 is mainly attributed to AAVs. Results from Table 5.1, Table 5.2, and Table 5.3 support the first hypothesis from our evaluation protocol.

Table 5.3: Comparison for the total time spent on algorithm analysis introduction modules without AAVs in Fall 2015 versus the total time spent in the same modules in Spring 2016 using a Mann-Whitney test. N here denotes the number of students who interacted with the material. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean		Median		p-value	
			Time	Visits	Time	Visits	Time	Visits
Multiple Parameters	F15	31	60.96	2.38	7	2	0.1456	0.0446
	S16	50	64.58	1.56	14	1		
Space Bounds	F15	31	46.41	2.22	8	2	0.3262	0.0077
	S16	47	74.25	1.40	9	1		
Code Tuning and Empirical Analysis	F15	30	23.76	1.93	4.5	1	0.0727	0.1049
	S16	39	46.38	1.46	9	1		

Having evidence that the intervention group students were more engaged with AAVs than control group students were engaged with the traditional textual content, we were interested to see whether this is also true for sub-groups within both the control and intervention groups. We were interested in comparing the level of engagement of those students who spent less time in both contents. In addition, we were also interested in comparing the level of engagement of those students who spent more time in both contents. Accordingly, a similar analysis as presented in tables 5.1 and 5.2 was done to compare the time spent by students in the first quartile (those students whose total time spent falls below the 25^{th} percentile) and fourth quartile (those students whose total time spent falls above the 75^{th} percentile) in both the control and intervention groups. Results are shown in tables 5.4, 5.5, 5.6, and 5.7.

It is clear from the tables that intervention group students who used AAVs to learn the analytical material were significantly more engaged with AAVs than control group students who used traditional textual material. This is true for those students who spent less time in both contents (first quartile students), and students who spent more time in both contents (fourth quartile students). The only exception was for the "Best, Average, and Worst Cases" module for students from the fourth quartile. Although intervention group students spent more time than control group students as indicated from the mean and median times, this difference is not significant as indicated from the p-value. We believe that this insignificance may be attributed to the small sample size of the control group quartile.

Table 5.4: Comparing the total time in seconds spent in a textual discussion from the first quartile of the control group versus the total time spent in corresponding AAV(s) from the first quartile of the intervention group for OpenDSA sorting modules using a Mann-Whitney test. N here denotes the number of students who interacted with the material from the first quartile. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean	Median	p-value
Insertionsort	F15	7	3.57	4	< 0.0001
	S16	32	44.12	41.5	
Bubblesort	F15	8	1.37	1.5	< 0.0001
	S16	32	22.53	23	
Selectionsort	F15	5	2.4	2	0.0014
	S16	34	27.64	31	
Mergesort	F15	8	2.25	2.5	< 0.0001
	S16	34	21.17	23.5	
Quicksort	F15	10	3.2	3	< 0.0001
	S16	34	33.08	35	
Heapsort	F15	10	1.6	2	< 0.0001
	S16	33	14.39	14	
Radixsort	F15	8	1.87	2	< 0.0001
	S16	31	13.38	14	
The Cost of Exchange Sorting	F15	7	21.14	22	< 0.0001
	S16	29	181.44	197	
Sorting Lower Bound	F15	8	6	6	< 0.0001
	S16	26	82.65	74	

Table 5.5: Comparing the total time in seconds spent in a textual discussion from the fourth quartile of the control group versus the total time spent in corresponding AAV(s) from the fourth quartile of the intervention group for OpenDSA sorting modules using a Mann-Whitney test. N here denotes the number of students who interacted with the material from the fourth quartile. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean	Median	p-value
Insertionsort	F15	7	253.14	171	0.0041
	S16	32	502.68	527.5	
Bubblesort	F15	7	70.85	53	0.0031
	S16	35	180.05	134	
Selectionsort	F15	5	108.6	81	0.0046
	S16	34	225.35	211.5	
Mergesort	F15	7	75	52	0.0007
	S16	33	230.3	185	
Quicksort	F15	7	265.14	215	0.0028
	S16	33	470.21	460	
Heapsort	F15	6	62.5	59.5	0.0001
	S16	30	254.03	203	
Radixsort	F15	5	81.2	66	0.0004
	S16	31	377.74	323	
The Cost of Exchange Sorting	F15	7	443.75	210	0.0008
	S16	29	1305.58	1154	
Sorting Lower Bound	F15	7	266.42	59	< 0.0001
	S16	26	1325.38	1302	

Table 5.6: Comparing the total time in seconds spent in algorithm analysis introduction modules without AAVs from the first quartile of the control group versus the total time spent in the same modules with AAVs from the first quartile of the intervention group using a Mann-Whitney test. N here denotes the number of students who interacted with the material from the first quartile. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds.

Module	Group	N	Mean	Median	p-value
Asymptotic Analysis and Upper Bounds	F15	8	12	12.5	< 0.0001
	S16	29	100	103	
Best, Average, and Worst Cases	F15	9	3.44	3	< 0.0001
	S16	22	22.54	18.5	
Lower Bounds and Theta Notation	F15	8	14.5	15	< 0.0001
	S16	30	150	150	
Algorithm Analysis Misunderstandings	F15	10	5.7	5.5	< 0.0001
	S16	35	60.8	51	
Problems,Algorithms, and Programs	F15	8	7	7	< 0.0001
	S16	34	89.85	96.5	
Analyzing Problems	F15	11	9.54	9	< 0.0001
	S16	31	71.8	79	
Calculating Program Running time	F15	9	104.88	111	< 0.0001
	S16	30	378.53	412	

Table 5.7: Comparing the total time in seconds spent in algorithm analysis introduction modules without AAVs from the fourth quartile of the control group versus the total time spent in the same modules with AAVs from the fourth quartile of the intervention group using a Mann-Whitney test. N here denotes the number of students who interacted with the material from the fourth quartile. F15 denotes the Fall 2015 (control) group, and S16 denotes the Spring 2016 (intervention) group. Time is in seconds. Insignificant p-values are italicized.

Module	Group	N	Mean	Median	p-value
Asymptotic Analysis and Upper Bounds	F15	8	364.25	305.5	< 0.0001
	S16	29	1937.75	1940	
Best, Average, and Worst Cases	F15	9	728.33	512	*0.0856*
	S16	22	1214	810.5	
Lower Bounds and Theta Notation	F15	8	1052.62	834.5	< 0.0001
	S16	30	2408.8	2274.5	
Algorithm Analysis Misunderstandings	F15	9	326.11	233	0.0001
	S16	35	1121.45	930	
Problems,Algorithms, and Programs	F15	8	782.62	297.5	0.0011
	S16	34	2089.38	1938	
Analyzing Problems	F15	8	246.87	174	< 0.0001
	S16	31	1174.03	1086	
Calculating Program Running time	F15	9	1509.22	1366	0.0022
	S16	29	2386.93	2334	

5.4 Collecting Student Feedback about AAVs

In Section 3.2.3, we presented results from a student survey done at the end of Fall 2014 asking students for their feedback about the algorithm analysis material presented in OpenDSA at that time, which was mainly based on text and static images. The results from the survey motivated the use of AAVs as an alternate method of presenting the analysis content in a more engaging way. In this section, we present the results from surveying CS3114 students in Spring 2016 after enhancing OpenDSA with AAVs as an alternative to the traditional content. Surveying students was a necessary step to support the results from the student interaction logs, as engagement is hard to measure without observing and questioning students [28]. But before presenting the results from Spring 2016 after adding AAVs, we present results from a survey done in Fall 2015. The purpose of this survey is to confirm the results from Fall 2014, as both Fall 2014 and Fall 2015 students did not learn with AAVs, as they used the same textual material.

In Fall 2015, 43 students responded to a survey in which they were asked the same questions as in the Fall 2014 survey presented in Appendix C. We found that about 87% of the surveyed students found algorithm dynamics concepts (how an algorithm works) easier than algorithm analysis concepts. 58% of the students who are more comfortable with algorithm dynamics attributed this to the material itself as algorithm analysis is more abstract and requires some familiarity with mathematical notations. On the other hand, 29% of the students who are more comfortable with dynamics attributed this to how each concept is presented in OpenDSA. In Fall 2015, algorithm analysis concepts were presented in OpenDSA like any other traditional textbook using textual discussion supported by static images, while algorithm dynamics were presented interactively using Algorithm Visualizations. When the students were asked to provide suggestions to improve the presentation of the algorithm analysis material in OpenDSA, 72% of them indicated clearly that they believe visualizations will definitely help. Other suggestions were to include video lectures to further describe the content and to just add more analysis examples.

We see that the results from the Fall 2015 survey coincide with the results from the Fall 2014 survey presented in Section 3.2.3. This indicates that a significant proportion of students are suffering in understanding the analysis content in OpenDSA, and they are demanding a better presentation.

After introducing AAVs to OpenDSA sorting modules and some of the algorithm analysis introduction modules in Spring 2016, 90 students were surveyed. They were asked similar questions as in Fall 2014 and Fall 2015 surveys. In addition, they were asked clearly to express their opinion about AAVs (we made sure that students clearly understood the difference between AAVs and AVs by stating the difference in the question) whether it helped them to understand algorithm analysis abstract concepts, and to what extent they found AAVs useful. The Spring 2016 survey is available in Appendix C.

Results indicate that about 83% of the surveyed students found AAVs useful in their under-

standing of the algorithm analysis concepts presented in OpenDSA. Here are some quotes from their answers to the question:

- *"Yes, the visual representations of running times were helpful. I thought the block representations were helpful."*
- *"Yes, the visualizations were effective with the concept of unit of work and the total work being the summation of the total area."*
- *"Yes, I felt that the step by step visuals made a difference in my understanding"*
- *"Definitely yes. It helped me to better compare different algorithms."*
- *"Yes I thought it was great to see how algorithms running longer with more data shows. Represent the analysis better."*
- *"Yes in particular the visualization of the proof that no sorting algorithm could be more than* $n \log n$ *"*
- *"Yes very very easier than looking at proofs/numbers"*
- *"Yes they provided an easy to understand visualization of something that is difficult to explain conceptually."*
- *"Yes visualizations helped. Make it easier to see Theta(x) and O(x) and put in visualization or obvious but not hidden in a text."*

Of these 83% who found AAVs useful, 51% found that AAVs are good as is and they didn't provide any sort of improvement as they liked it the way it is designed. About 15% didn't know what to say about an improvement, but they didn't mention clearly that they liked the AAVs as is. 34% stated that although they found AAVs useful to them, they have some ideas for improvement. Here we present representative suggestions:

- Several students mentioned that they still want more visualizations. So far, we have only added 28 AAVs to OpenDSA, and we believe that there are additional analysis discussions that can be further visualized.
- One student mentioned about tagging a visualization slide so that he can remember coming back to it again. This may be a useful feature as we found from the previous section that students tend to visit AAVs more times than they visited the textual discussion and they may only be interested in a subset of the slides in a given AAV.
- Some students mentioned their need for an AAV feature that was available for most sorting AAVs, which is displaying and highlighting code while building the graphical shape representing the running time of the highlighted statement. We believe this to be an important feature of AAVs that is also found in AVs. Some sorting AAVs currently have this feature.
- Some students stated that some AAVs need more colors to better enhance specific details. Most sorting AAVs have this feature, but we also believe that some AAVs for the algorithm analysis chapter can be enhanced with more coloring.
- Some students mentioned that AAVs should be more detailed by having more steps (i.e., slides). We are concerned that AAVs are detailed enough, such that adding more slides to them may affect them negatively. On average, an AAV has about 10-15 slide which we believe is a reasonable number to hold students' attention.

Only 8% of the students expressed negative feedback about AAVs. Here are some quotes of student feedback when asked to express their opinion about whether AAVs helped them in understanding algorithm analysis concepts:

- *"No it didn't really stick well in my head".*
- *"No because the graph isn't way clear'.*
- *"I felt that these are less helpful than how an algorithm works. This is because they often relied on the difference between O, Omega., and Theta that I was not confident with."*
- *"No. Only for the algorithm itself."*
- *"The visualizations for algorithm analysis didn't really help me. I just understand the concept and was able to set it from the text."*
- *"Somewhat — these concepts were still very abstract and I had a hard time understanding them even with the graph visualizations. It would have been helpful for me to have a more concrete metaphorical comparison to understand the concept of bounds".*

As we see from the quotes, it may be obvious that the problem is not in AAVs per se, but it is mainly in the concept itself except for the student who stated that the graphs were not clear.

85% of the students still found that algorithm dynamics concepts are easier than algorithm analysis concepts. Accordingly, AAVs didn't change the opinion of students about the difficulty of algorithm analysis topics. 74% of those students more comfortable with dynamics attributed this to the material itself, while 26% of them attributed this to the presentation. Even after AAVs, 26% of the students still believe that the dynamics concepts are better presented than analysis concepts in OpenDSA. One explanation to this is what was stated in [10], that visualizations appear better able to explain a dynamic evolving process as they help viewers to track patterns and observe relationships. This is the case for procedural dynamics, as an algorithm is intrinsically a dynamic process that evolves over time and imposes some changes on a data structure. But for a visual proof, this is not exactly the case. A proof is not a dynamic process, but rather it is a sequence of arguments that collectively validate or refute a given hypothesis. This is why it was stated in Chapter 1 that designing AAVs is more challenging than AVs. Accordingly, we believe that despite the fact that AAVs are found by most of the students to be helpful, they will still find AVs better as they are depicting intrinsically dynamic processes. As most of the students gave positive feedback on AAVs, we believe that the second hypothesis from our evaluation protocol is supported.

5.5 Evaluating Student Performance

Measuring student learning gains made with technology is a difficult endeavor [18]. However, we believe that introducing AAVs to OpenDSA will help greatly in enhancing the effectiveness of the modules. The reason behind this belief is stated in Carroll's time-on-task

hypothesis, that the longer a student spends engaging with the learning material, the more opportunities the student has to learn [12]. In order to evaluate AAVs in terms of student performance, we offered the pilot AACI as described in Appendix B to both our control (Fall 2015 without AAVs) and intervention (Spring 2016 with AAVs) groups as part of their final exam. In our initial evaluation, we preferred not to adopt the traditional pre/post test approach for two reasons:

1. Offering a pre-test at the beginning of a class and a post-test at the end of the class has a high cost, as most instructors are reluctant to waste a significant portion of their lecture's time.
2. We don't believe that a post-test is an actual measure of student performance as the students typically take the post-test one or two weeks before the final and they may not be well prepared. In addition, typically the pre- and post- tests are very similar (in order to correctly calculate the learning gain). This may impose a problem as now the students may remember the questions from the pre-test and know the correct answers for them to be able to answer it correctly in the post-test.

Based on this, we evaluated the effect of AAVs based on student performance in the algorithm analysis part of the finals to avoid these two problems. The final in both Fall 2015 and Spring 2016 was divided into two sections, one for general CS3 concepts and the other dedicated to algorithm analysis. The whole test time was 2 hours for both groups.

Only similar items were compared in both groups, for a total of 27 items. For the purpose of this study, each question worth one point, as we seek a comparison in student performance within each concept addressed by the items. We believe that this will measure the actual student performance in the algorithm analysis concepts presented in the course as each item is designed to target one or more concepts based on student misconceptions as revealed from Chapter 4. Table 5.8 shows the results from comparing the performance of the Spring 2016 group on the algorithm analysis part of the final, to the performance of the Fall 2015 group taking the same set of questions. Note that the results from the table are to the level of concepts, as for some concepts (Analyzing loops, Relative Growth Rates) we did not provide any AAV in OpenDSA. Accordingly, we want to see the difference in performance in those concepts for which AAVs were provided and also for those concepts where AAVs were not provided. In addition, Table 5.9 shows the effect sizes of the differences found in Table 5.8, calculated using Cohen's d for different sample sizes [32] (also known as Hedges g).

It is clear from Table 5.8, that Spring 2016 students have significantly better scores on the whole set of questions than Fall 2015 students. In addition, for only those concepts presented using AAVs, Spring 2016 students have better performance. Both groups have similar performance in those concepts not supported by AAVs. Furthermore, results from Table 5.9 indicates large effect sizes of roughly one standard deviation for the difference between both groups supporting the results from Table 5.8. This supports the third hypothesis in the evaluation protocol.

Table 5.8: Comparing student grades in the algorithm analysis part of the final from Fall 2015 (N = 67) Vs. the student grades in the same part from Spring 2016 (N = 155) using a Mann-Whitney test. Concepts not supported by AAVs as well as insignificant p-values are italicized.

Concept(s)	#Items	Mean Score		Median Score		p-value
		Control	Test	Control	Test	
Relative Growth Rates	8	5.73	5.72	7	7	*0.7470*
Upper, Lower, and tight bounds	7	3.13	4.76	2	5	< 0.0001
Problem Upper and Lower Bounds	3	0.65	1.98	0	2	< 0.0001
Best, Average and Worst Cases	5	3.44	4.38	4	5	< 0.0001
Analyzing Loop Constructs	4	3.104	3.109	3	3	*0.6697*
All	27	16.07	19.97	16	20	< 0.0001

Table 5.9: Effect sizes of the difference in grades between Fall 2015 and Spring 2016 students calculated using Cohen's d for different sample sizes. Concepts not supported by AAVs are italicized.

Concept(s)	Effect Size
Relative Growth Rates	-0.005
Upper, Lower, and tight bounds	0.892
Problem Upper and Lower Bounds	1.239
Best, Average and Worst Cases	1.012
Analyzing Loop Constructs	0.005
All	0.964

On the surface, it might appear that these results refute Naps, *et al.*'s hypothesis regarding the "Viewing" versus the "no-viewing" levels of engagement [62]. Naps, *et al.* stated that, "*Viewing results in equivalent learning outcomes to no visualization*". From our engagement, satisfaction, and performance evaluations, we found that AAVs (the viewing level) provided higher engagement, better student learning experience, and better performance compared to traditional textual content (the no-viewing level). An important distinction to make here is that we are not really comparing learning from prose against learning from AAVs. We are actually comparing an eTextbook environment where students are offered prose descriptions versus an eTextbook environment where students are offered a visual presentation of the same material. We don't know what the results would be if the time spent with the textual discussion was similar to that spent with AAVs (i.e., both content were engaging to students). What we do know is that approximately 75% of students appear to skip viewing the prose version, while only approximately 30% of students appear to skip the visual presentation, and that the overall average time spent is greater for the visual presentation. Accordingly, we do not conclude from the results we have that the "no-viewing" versus "viewing" hypothesis does not hold. Instead, we conclude that AAVs are more engaging in practice to students

than the equivalent prose presentation.

5.6 Threats to Validity

As mentioned in the previous section, AAVs were evaluated in terms of student scores in their final exam for those questions related to algorithm analysis. One of the main threats to the validity of this evaluation is that we didn't account for any difference that may exist in the students' pre-knowledge between the control and intervention groups. Comparing the student scores in a pre-test for both groups may have served in overcoming this problem. However, we do not believe that there is any difference in the pre-knowledge between both groups as the students are taking the same course CS3114 and they are all required to take its pre-requisites CS2114 (Software Design and Data Structures) and CS2104 (Introduction to problem solving). Based on this we do not believe that there is anything different that may make one group have more pre-knowledge than the other group. In addition, we performed a detailed comparison between both groups based on some demographic information. We were able to collect demographic information from 150 students during Spring 2016 and 46 students during Fall 2015. Results are shown in Table 5.10.

Table 5.10: Comparing the demographic information of students from Fall 2015 (N = 46) to the same information from Spring 2016 (N = 150). The p-value is calculated from a two sample proportion test.

Demographics	N		Proportion		p-value
	Fall 2015	Spring 2016	Fall 2015	Spring 2016	
		Gender			
Male	39	125	0.847	0.83	0.8181
Female	7	24	15.3	16	0.8965
Other	0	1	0	0.0066	0.5754
		Ethnicity			
White	28	91	0.608	0.606	0.9840
Asian	16	45	0.347	0.3	0.5418
Hispanic	0	6	0	0.04	0.1675
African-American	1	5	0.021	0.033	0.6891
Other	1	3	0.021	0.02	0.9442
		First Generation Student			
Yes	3	27	0.066	0.18	0.0587
No	43	123	0.934	0.82	0.0587

As we see, it is clear from the table that both groups have similar proportions regarding the gender and ethnicity with no statistically significant difference. However, we see some

difference in the proportions for first student generation, however, this difference is not statistically significant.

In addition, we did conduct pre-tests for students during both Fall 2014 (identical content to Fall 2015) and Spring 2016 (intervention group). We found that the Fall 2014 and Spring 2016 groups had pre-test results that were not significantly different, based on a Mann-Whitney test ($p = 0.1727$). Based on this, we believe that Fall 2015 control group and Spring 2016 intervention group are similar in the sense that there is no reason to worry about any difference in student pre-knowledge between both groups. Since we gave a pre-test to the intervention group, but not the control group, a reasonable concern is that the act of taking the pre-test itself becomes an intervention that affects the final outcome. This appears not to be the case, since as described above, the intervention group's scores on the final exam were significantly improved only for the three topics that were addressed by AAVs, and not for the two topics not addressed by AAVs. If taking the pre-test had affected performance on the final exam, then all topics would have been affected to some degree.

Another issue of concern is whether other factors caused the differences between the groups in the time spent on the algorithm analysis material or performance on the final exam. One factor could be that different instructors taught the course in Fall 2015 and Spring 2016. However, they used the same material and the same textbook (i.e., OpenDSA) for the course. The only difference was using AAVs in Spring 2016. Accordingly, we do not believe that this had a major contribution in the difference between both groups. Our belief here is supported by two points. First, our performance evaluation results indicate that students in Spring 2016 have better exam scores only on those concepts presented with AAVs. If there is any contribution by instructors, then we would expect a difference in the other concepts as well. Second, the Spring 2016 instructor also taught the course during Fall 2014. In Fall 2014, the algorithm analysis material was identical to Fall 2015 material (no AAVs). We have conducted an analysis similar to the one presented in Section 5.3. We compared the level of engagement of Fall 2014 students in the algorithm analysis material to the level of engagement of Fall 2015 students. This was done for OpenDSA sorting and algorithm analysis introductory modules. We found that for all modules there is no significant difference ($p > 0.05$) in the total time spent in the algorithm analysis material between Fall 2014 and Fall 2015 students. The only exception was for the "Problems, Algorithms, and Programs" module in which we found that Fall 2014 students spent significantly more time reading the algorithm analysis material presented in the module ($p = 0.0004$). This indicates that if there is any effect attributed to the instructors, it may be only for this module. This module presents the basic definitions of a problem, algorithm, and a program. These definitions were not tested in the final exam used for our performance evaluation. Accordingly, we don't see an effect from this module on student scores. This indicates that AAVs are the major factor contributing to both greater student time spent and improved student performance.

Another concern is related to a change in OpenDSA infrastructure. Prior to Spring 2016, OpenDSA had little support to reduce the act of students gaming OpenDSA exercises. Examples of student gaming include, abusing the use of hints and trial-and-error to find the

correct answer to hard exercises. In addition, when faced with a hard exercise, students could just refresh the page to get another question. This process could be repeated until the student receive a question that he can answer. Starting from Spring 2016, the OpenDSA infrastructure was updated such that when a student refreshed a page containing an exercise, the exercise question is not changed until he answers it correctly. This may have an effect on the student's learning experience and most importantly, the time spent in OpenDSA modules. However, we believe that if this change has any effect on the time spent in modules, it will be only for those credit seekers. When a credit seeker student opens a module, he jumps directly to the module exercises to secure his credit without learning the content presented in the module. In the old infrastructure, the student can just refresh the page several times until he gets an easy question. But, with the new infrastructure available, credit seekers may be forced to read the content to be able to answer the exercise as it will never be changed with refreshing the page. However, those credit seekers can still use trial-and-error to know the correct answer for a hard exercise. They can memorize the correct answer they got from the first attempt and answer the question correctly when the KA framework picks the question again. Accordingly, we don't believe that this threat has any major effect on the time spent or student performance.

A final issue is related to excluding students with outlier time spent from both control and intervention groups as described in Section 5.3. It may be the case that students with such behavior did some meaningful interaction with the content first, and then left the browser opened and went somewhere else. In the analysis done, we ignored the data from all students with such behavior. However, by doing so, we are also ignoring some true time spent (the time before the student left the browser). We don't believe that this is a major problem that can affect the results of the evaluation, as this behavior was detected from students in both the control and intervention groups with similar rates.

Chapter 6

Conclusions and Future Work

AAVs were found to be more engaging in that intervention group students spent more time interacting with them than control group students spent with the corresponding textual discussion. The difference was significant for all OpenDSA modules containing AAVs. The majority of the surveyed students (83%) mentioned positive feedback about the ability of AAVs to help them understand the algorithm analysis topics. However, a significant proportion of students gave at least one idea for improvement.

Given the engagement evidence from analyzing OpenDSA log data and the satisfaction evidence from the survey, we expected to see improved exam scores for the intervention group on the algorithm analysis part of the final exam. The scores for the intervention group students were significantly higher than the scores for the control group students, with large effect sizes of roughly one full standard deviation. Equally important, we also found that this improvement is only found on those questions addressing topics that were presented using AAVs. On other algorithm analysis questions that were not related to content presented using AAVs, there was no statistically significant difference between exam scores for the two groups. Accordingly, we believe that AAVs were a major contributor to this improvement.

Most students surveyed from the intervention group (85%) indicated that algorithm dynamics concepts are easier for them to understand than algorithm analysis concepts despite the use of AAVs. Control group students reported the same, in the same proportion. So AAVs did not change the students' opinion about the relative difficulty of algorithm analysis topics. Some students from Spring 2016 attributed the difficulty of algorithm analysis topics to the style of presentation. Even with AAVs, 26% of students still believe that the dynamics concepts are better presented than the analysis concepts in OpenDSA. As both are presented visually, this may indicate that students like AVs more than AAVs. One explanation for this is that visualizations appear better able to explain a dynamic evolving process as they help viewers to track patterns and observe relationships [10]. An algorithm is intrinsically a dynamic process that evolves over time and imposes some changes on a data structure. A run-time proof is not a dynamic process, but rather it is a sequence of statements that collectively

validate or refute a given hypothesis.

Here we describe some directions to further explore the effectiveness of AAVs and to further develop the AACI.

6.1 More AAVs

We developed AAVs to visually present run-time proofs for the main sorting algorithms and the basic introductory concepts of algorithm analysis. Having proved to be effective in terms of student engagement and performance, we believe that more AAVs should be implemented and evaluated for other topics. Other topics that we believe can be targeted by AAVs include:

- Relative growth rates of simple mathematical functions.
- Analyzing loop constructs.
- Binary-tree traversals.
- Graph traversals and shortest-path algorithms.

In addition, students' suggestions presented in Section 5.4 for improving AAVs as indicated from the survey should be carefully studied and addressed.

6.2 Evaluating AAVs based on student demographic information

We believe that further evaluation studies should be done for AAVs, taking into consideration the differences that may arise from student demographics (gender, ethnicity, first generation student) and majors (not all of the students taking CS3114 are CS majors). The reason behind conducting these evaluations is to identify those groups who get the highest benefit from AAVs regarding student engagement, satisfaction, and performance. A quasi-experiment similar to the one described in Chapter 5 should be conducted and comparisons should be done to the level of subgroups, similar to what was done in [51].

6.3 Evaluating AAVs at multiple levels of engagement

So far, all AAVs fall into the viewing level of student engagement according to the engagement level taxonomy defined in [62]. AAVs merely turn a textual discussion into a series of visual presentations that the user can view sequentially by using the forward and backward controls. The quasi-experiment described in Chapter 5 compared the student engagement

and performance between the no-viewing (no AAVs) and viewing (AAVs) levels. A careful investigation is required to see whether AAVs can be extended to further levels of engagement (i.e., Responding, Changing, and Constructing, defined in [62]) as was done for traditional AVs.

To support the Responding level of engagement, students should be interrupted with questions about the presented concept(s) while viewing the AAVs. We believe it may be relatively easy to support this level in AAVs. The JSAV library has support to create simple questions and integrate them with AAVs.

To support the Changing level, the student should be able to change the underlying data structure and run the AAV on it. This is challenging as AAVs do not simulate the procedural dynamics of an algorithm, but rather a proof. Accordingly, to support the changing level, the AAV should create a proof dynamically to work on the changed data structure. This can be relatively easy for those AAVs relying on the Area-to-Cost principle. For example, the Insertionsort worst-case analysis AAV presented in Figure 3.5(b) can be modified to allow the student to provide his own array input. The AAV then should create a similar shape by adding the right number of rectangles corresponding to the size of the array provided by the student.

To support the construction level, the student should be able to construct his own AAV. We believe that the JSAV library is handy for students to use and implement their own AAVs. Another way to support the Construction level is through Proficiency Exercises (PEs). In a PE, a student is presented with a visual depiction of a data structure state and he is required to interactively manipulate the data structure simulating the execution of a specific algorithm. Creating PEs for algorithm analysis is hard, as there is no dynamic process to simulate.

6.4 Alpha AACI

In Chapter 4, we described initial steps towards developing a CI for C3-level algorithm analysis topics. Our pilot AACI did a good job in revealing students' misconceptions expressed through their responses. For misconceptions M1 to M4 , we didn't find enough evidence from students' responses, but still some students are holding these misconceptions. We removed M2 from the misconceptions list. In addition, we believe that better items are required to detect misconception M17. This misconception is related to the ability of students to differentiate between the best, average, and worst cases of algorithms.

The pilot AACI items have now received face validity by our Delphi experts, and they did a good job in detecting various students' misconceptions in initial testing. In addition, the pilot AACI was found to be a reliable instrument as indicated from a Cronbach's-alpha reliability coefficient of 0.82. We are planning to administer the prospective alpha-AACI in several different universities to be able to validate it and test its reliability. In addition,

student interviews should be conducted for better validation of the misconceptions list.

Bibliography

[1] Ayonike Akingbade, Thomas Finley, Diana Jackson, Pretesh Patel, and Susan H Rodger. Jawaa: easy web-based animation from CS0 to advanced CS courses. In *ACM SIGCSE Bulletin*, volume 35, pages 162–166, 2003.

[2] Kirk Allen. The statistics concept inventory: Development and analysis of a cognitive assessment instrument in statistics. *Available at SSRN 2130143*, 2006.

[3] Mary J Allen and Wendy M Yen. *Introduction to measurement theory*. Waveland Press, 2001.

[4] Vicki L Almstrum, Peter B Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. Concept inventories in computer science for the topic discrete mathematics. In *ACM SIGCSE Bulletin*, volume 38, pages 132–145, 2006.

[5] Frank B Baker. *The basics of item response theory*. ERIC, 2001.

[6] Ryan S. Baker, Albert T. Corbett, and K.R. Koedinger. Detecting student misuse of intelligent tutoring systems. In J.C. Lester, R.M. Vicari, and F. Paraguaçu, editors, *Intelligent Tutoring Systems: Seventh International Conference*, pages 531–540. 2004.

[7] Ryan S. Baker, Albert T. Corbett, K.R. Koedinger, and A.Z. Wagner. Off-task behavior in the cognitive tutor classroom: When students "game the system". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 383–390, 2004.

[8] Don Blaheta. A visual proof of amortised-linear resizable arrays. *ACM SIGCSE Bulletin*, 41(3):338–338, 2009.

[9] Daniel A Breakiron. Evaluating the integration of online, interactive tutorials into a data structures and algorithms course. Master's thesis, Virginia Polytechnic Institute and State University, 2013.

[10] Michael D Byrne, Richard Catrambone, and John T Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & education*, 33(4):253–278, 1999.

[11] Ricardo Caceffo, Steve Wolfman, Kellogg S Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 364–369, 2016.

[12] John Carroll. A model of school learning. *The Teachers College Record*, 64(8):723–723, 1963.

[13] Pierluigi Crescenzi. Using AVs to explain NP-completeness. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 299–299, 2010.

[14] Pilu Crescenzi and Carlo Nocentini. Fully integrating algorithm visualization into a CS2 course: A two-year experience. In *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 296–300, 2007.

[15] Linda M Crocker, James Algina, et al. *Introduction to classical and modern test theory*, volume 6277. JSTOR, 1986.

[16] Norman Dalkey and Olaf Helmer. An experimental application of the delphi method to the use of experts. *Management science*, 9(3):458–467, 1963.

[17] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 21–26, 2012.

[18] Richard E Ferdig. Assessing technologies for teaching and learning: understanding the importance of technological pedagogical content knowledge. *British Journal of Educational Technology*, 37(5):749–760, 2006.

[19] Eric Fouh, Daniel A. Breakiron, Sally Hamouda, Mohammed F. Farghally, and Clifford A. Shaffer. Exploring students learning behavior with an interactive etextbook in computer science courses. *Computers in Human Behavior*, pages 478–485, December 2014.

[20] Eric Fouh, Ville Karavirta, Daniel A Breakiron, Sally Hamouda, Simin Hall, Thomas L Naps, and Clifford A Shaffer. Design and architecture of an interactive etextbook–The OpenDSA system. *Science of Computer Programming*, 88:22–40, 2014.

[21] Judith Gal-Ezer and Ela Zur. The efficiency of algorithms misconceptions. *Computers & Education*, 42(3):215–226, 2004.

[22] Tesfaye Getinet. Effect of instructional interventions on students learning gains: An experimental research. *Lat. Am. J. Phys. Educ. Vol*, 6(2):187, 2012.

[23] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C Loui, and Craig Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin*, 40(1):256–260, 2008.

[24] Michael T Goodrich and Roberto Tamassia. Teaching the analysis of algorithms with visual proofs. In *ACM SIGCSE Bulletin*, volume 30, pages 207–211, 1998.

[25] Michael T Goodrich, Roberto Tamassia, and Michael H Goldwasser. *Data structures and algorithms in Java*. John Wiley & Sons, 2014.

[26] Scott Grissom, Myles F McNally, and Tom Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94, 2003.

[27] Philip J. Guo, Juho Kim, and Rob Rubin. How video production affects student engagement: An empirical study of mooc videos. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 41–50, 2014.

[28] Philip J Guo, Juho Kim, and Rob Rubin. How video production affects student engagement: An empirical study of mooc videos. In *Proceedings of the first ACM conference on Learning@ scale conference*, pages 41–50, 2014.

[29] Thomas Hainey. Using games-based learning to teach requirements collection and analysis at tertiary education level. 2010.

[30] Richard H Hammack and David W Lyons. Alternating series convergence: a visual proof. *Teaching Mathematics and its Applications*, 25(2):58, 2006.

[31] Sally Hamouda. *Enhancing Learning of Recursion*. Doctoral dissertation, Virginia polytechnic Institute and State University, 2015.

[32] LV Hedges and I Olkin. Statistical methods for meta-analysis: Academic press. *Orlando, FL*, 1985.

[33] Geoffrey L Herman, Lisa Kaczmarczyk, Michael C Loui, and Craig Zilles. Proof by incomplete enumeration and other logical misconceptions. In *Proceedings of the Fourth international Workshop on Computing Education Research*, pages 59–70, 2008.

[34] Geoffrey L Herman, Michael C Loui, and Craig Zilles. Creating the digital logic concept inventory. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 102–106, 2010.

[35] Geoffrey L Herman, Craig Zilles, and Michael C Loui. Work in progress-students' misconceptions about state in digital systems. In *Frontiers in Education Conference, 2009. FIE'09. 39th IEEE*, pages 1–2, 2009.

[36] David Hestenes, Malcolm Wells, Gregg Swackhamer, et al. Force concept inventory. *The physics teacher*, 30(3):141–158, 1992.

[37] Simon Holland, Robert Griffiths, and Mark Woodman. Avoiding object misconceptions. In *ACM SIGCSE Bulletin*, volume 29, pages 131–134, 1997.

[38] Wen-Hao Huang. Evaluating learners motivational and cognitive processing in an online game-based learning environment. *Computers in Human Behavior*, 27(2):694–704, 2011.

[39] Douglas Huffman and Patricia Heller. What does the force concept inventory actually measure?. *Physics Teacher*, 33(3):138–43, 1995.

[40] Cristopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, June 2002.

[41] Anthony Jacobi, Jay Martin, John Mitchell, and Ty Newell. A concept inventory for heat transfer. In *Frontiers in Education, 2003. FIE 2003 33rd Annual*, volume 1, pages T3D–12, 2003.

[42] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 107–111, 2010.

[43] Ville Karavirta. Seamless merging of hypertext and algorithm animation. *ACM Transactions on Computing Education (TOCE)*, 9(2):10, 2009.

[44] Ville Karavirta, Ari Korhonen, Jussi Nikander, and Petri Tenhunen. Effortless creation of algorithm visualization. In *Proceedings of the Second Annual Finnish/Baltic Sea Conference on Computer Science Education*, pages 52–56, 2002.

[45] Ville Karavirta and Clifford Shaffer. Creating engaging online learning material with the JSAV javascript algorithm visualization library. *IEEE Transactions on Learning Technologies*, pages 171 – 183, 2015.

[46] Ville Karavirta and Clifford A Shaffer. JSAV: the javascript algorithm visualization library. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 159–164, 2013.

[47] Juko Kim, Philip J. Guo, Carrie J. Cai, Shang-wen W. Li, Krzysztof Z. Gajos, and Robert C. Miller. Data-driven interaction techniques for improving navigation of educational videos. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 563–572, 2014.

[48] Archie Korhonen, Lauri Malmi, Panu Silvasti, Jussi Nikander, Harri Salonen, and Ville Karavirta. TRAKLA2. `http://www.cs.hut.fi/Research/TRAKLA2/`, 2003.

[49] Markus Krebs, Tobias Lauer, Thomas Ottmann, and Stephan Trahasch. Student-built algorithm visualizations for assessment: flexible generation, feedback and grading. In *ACM SIGCSE Bulletin*, volume 37, pages 281–285, 2005.

[50] Amruth N Kumar. Results from the evaluation of the effectiveness of an online tutor on expression evaluation. In *ACM SIGCSE Bulletin*, volume 37, pages 216–220, 2005.

[51] Amruth N Kumar. The effectiveness of visualization for learning expression evaluation. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 362–367, 2015.

[52] Teemu Rajala Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Effectiveness of program visualization: A case study with the ville tool. *Journal of Information Technology Education*, 7, 2008.

[53] Tobias Lauer. Learner interaction with algorithm visualizations: viewing vs. changing vs. constructing. In *ACM SIGCSE Bulletin*, volume 38, pages 202–206, 2006.

[54] Julie C Libarkin and Steven W Anderson. Assessment of learning in entry-level geoscience courses: Results from the geoscience concept inventory. *Journal of Geoscience Education*, 53(4):394, 2005.

[55] Nabanita Maji. An interactive tutorial for NP-completeness. Master's thesis, Virginia Polytechnic Institute and State University, 2015.

[56] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, September 2004.

[57] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[58] Jay Martin, John Mitchell, and Ty Newell. Development of a concept inventory for fluid mechanics. In *Frontiers in Education, 2003. FIE 2003 33rd Annual*, volume 1, pages T3D–23. IEEE, 2003.

[59] Richard E. Mayer. Cognitive theory and the design of multimedia instruction: An example of the two-way street between cognition and instruction. *New Directions for Teaching and Learning*, 89:55–71, Spring 2002.

[60] Thomas Naps, Stephen Cooper, Boris Koldehofe, Charles Leska, Guido Rößling, Wanda Dann, Ari Korhonen, Lauri Malmi, Jarmo Rantakokko, Rockford J Ross, et al. Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin*, 35(4):124–136, 2003.

[61] Thomas L Naps, James R Eagan, and Laura L Norton. Jhavéan environment to actively engage students in web-based algorithm visualizations. *ACM SIGCSE Bulletin*, 32(1):109–113, 2000.

[62] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *ACM SIGCSE Bulletin*, volume 35, pages 131–152. ACM, 2002.

[63] Thomas L Naps and Brian Swander. An object-oriented approach to algorithm visualizationeasy, extensible, and dynamic. *ACM SIGCSE Bulletin*, 26(1):46–50, 1994.

[64] Mary A Nelson, Monica R Geist, Ronald L Miller, Ruth A Streveler, and Barbara M Olds. How to create a concept inventory: The thermal and transport concept inventory. In *Annual Conference of the American Educational Research Association, Chicago, IL*, 2007.

[65] Branislav M Notaros. Concept inventory assessment instruments for electromagnetics education. In *Antennas and Propagation Society International Symposium, 2002. IEEE*, volume 1, pages 684–687, 2002.

[66] Jum Nunnally. Psychometric theory, 2nd. edition. *McGraw-Hill, New York*, 1978.

[67] Nesrin Özdener. A comparison of the misconceptions about the time-efficiency of algorithms by various profiles of computer-programming students. *Computers & Education*, 51(3):1094–1102, 2008.

[68] Marina Papastergiou. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education*, 52(1):1–12, 2009.

[69] Miranda Parker and Colleen Lewis. What makes big-o analysis difficult: understanding how students understand runtime analysis. *Journal of Computing Sciences in Colleges*, 29(4):164–174, 2014.

[70] Leo Porter, Saturnino Garcia, Hung-Wei Tseng, and Daniel Zingaro. Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 279–284, 2013.

[71] Daniel Roberge, Anthony Rojas, and Ryan S. Baker. Does the length of time off-task matter? In *Proceedings of the Second International Conference on Learning Analytics and Knowledge*, LAK '12, pages 234–237, 2012.

[72] Susan H Rodger, Eric Wiebe, Kyung Min Lee, Chris Morgan, Kareem Omar, and Jonathan Su. Increasing engagement in automata theory with jflap. In *ACM SIGCSE Bulletin*, volume 41, pages 403–407, 2009.

[73] Guido Rößling, Markus Schüer, and Bernd Freisleben. The ANIMAL algorithm animation tool. In *Proceedings of the 5th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pages 37–40, 2000.

[74] Purvi Saraiya, Clifford A Shaffer, D Scott McCrickard, and Chris North. Effective features of algorithm visualizations. 36(1), 2004.

[75] Antti Savinainen and Philip Scott. The force concept inventory: a tool for monitoring student learning. *Physics Education*, 37(1):45, 2002.

[76] Clifford A Shaffer. *Data Structures & Algorithm Analysis in Java*. Courier Corporation, 2011.

[77] Clifford A. Shaffer. OpenDSA CS3114 eTextbook. `http://algoviz.org/OpenDSA/Books/CS3114/html/AnalMisunderstanding.html`, 2014.

[78] Clifford A Shaffer, Matthew L Cooper, Alexander Joel D Alon, Monika Akbar, Michael Stewart, Sean Ponce, and Stephen H Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)*, 10(3):9, 2010.

[79] Clifford A Shaffer, Lenwood S Heath, and Jun Yang. Using the swan data structure visualization system for computer science education. In *ACM SIGCSE Bulletin*, volume 28, pages 140–144, 1996.

[80] David B Sher. A visual proof for an average case of list searching. *ACM SIGCSE Bulletin*, 40(2):74–78, 2008.

[81] Paul S Steif and John A Dantzler. A statics concept inventory: Development and psychometric analysis. *Journal of Engineering Education*, 94(4):363, 2005.

[82] Cynthia Taylor, D Zingaro, L Porter, KC Webb, CB Lee, and M Clancy. Computer science concept inventories: past and future. *Computer Science Education*, 24(4):253–276, 2014.

[83] Allison Elliott Tew and Mark Guzdial. Developing a validated assessment of fundamental cs1 concepts. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 97–101, 2010.

[84] Allison Elliott Tew and Mark Guzdial. The fcs1: a language independent assessment of cs1 knowledge. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 111–116, 2011.

[85] Hussein Thompson and Pranay Chadhuri. An alternative visual analysis of the build heap algorithm. *ACM Inroads*, 2(3):31–32, 2011.

[86] Jaime Urquiza-Fuentes and J Ángel Velázquez-Iturbide. A survey of successful evaluations of program visualization and algorithm animation systems. *ACM Transactions on Computing Education (TOCE)*, 9(2):9, 2009.

[87] Jaime Urquiza-Fuentes and J Angel Velázquez-Iturbide. A long-term evaluation of educational animations of functional programs. In *Advanced Learning Technologies (ICALT), 2012 IEEE 12th International Conference on*, pages 26–30, 2012.

[88] Kathleen E Wage, John R Buck, Cameron HG Wright, and Thad B Welch. The signals and systems concept inventory. *Education, IEEE Transactions on*, 48(3):448–461, 2005.

[89] Kevin C Webb and Cynthia Taylor. Developing a pre-and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 103–108, 2014.

Appendix A

Algorithm Analysis Visualizations

A.1 General Algorithm Analysis AAVs

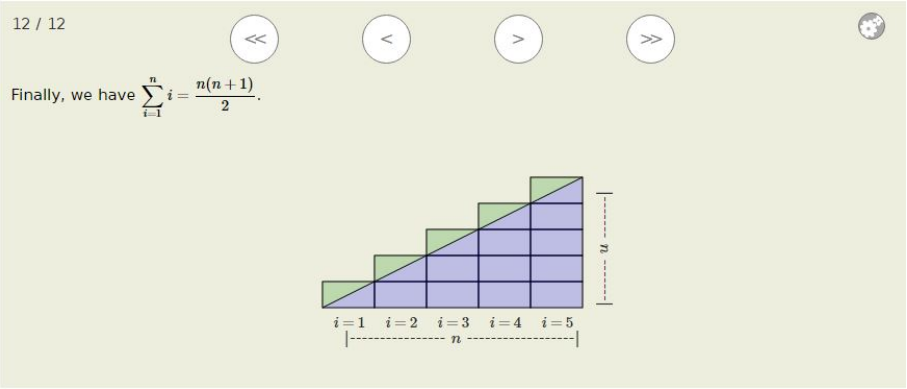


Figure A.1: Closed form solution for summation 1 to N.

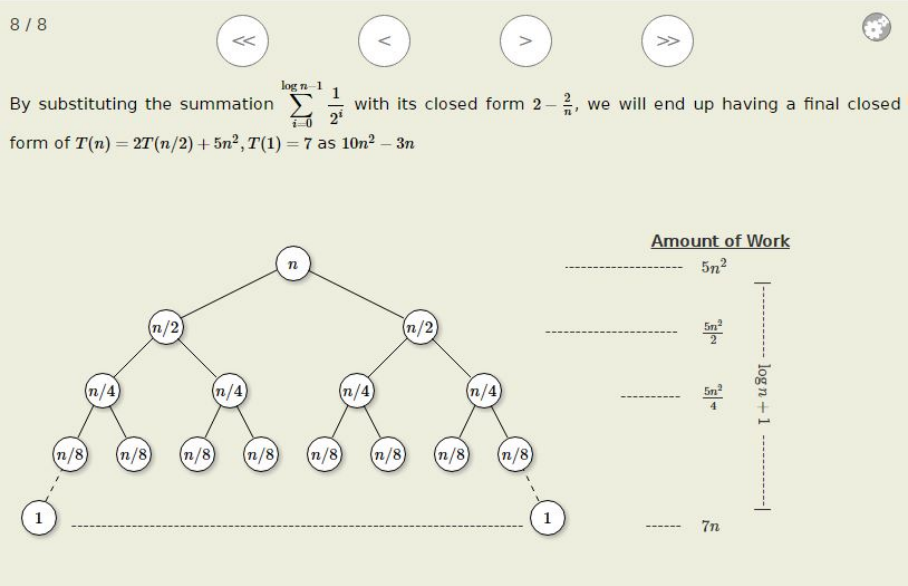


Figure A.2: Unrolling the Recurrence $T(n) = 2T(n/2) + 5n^2$.

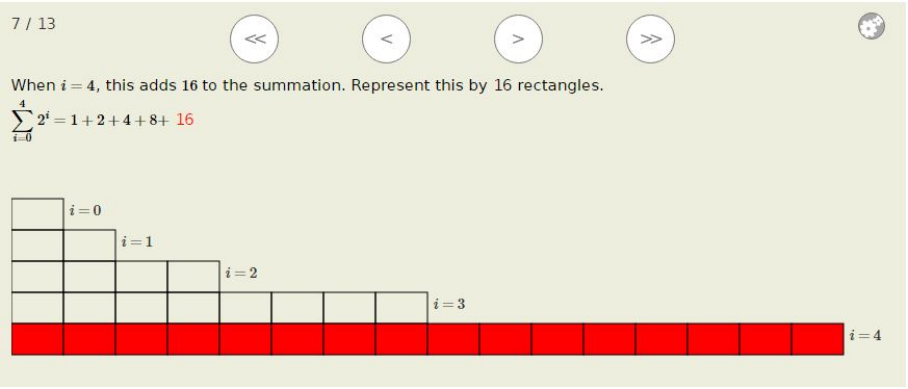


Figure A.3: Closed form solution for the summation $\sum_{i=0}^{n} 2^i$

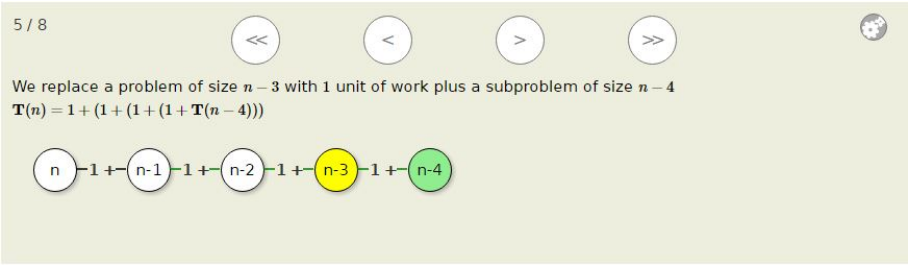


Figure A.4: Unrolling the recurrence $T(n) = T(n-1) + 1$

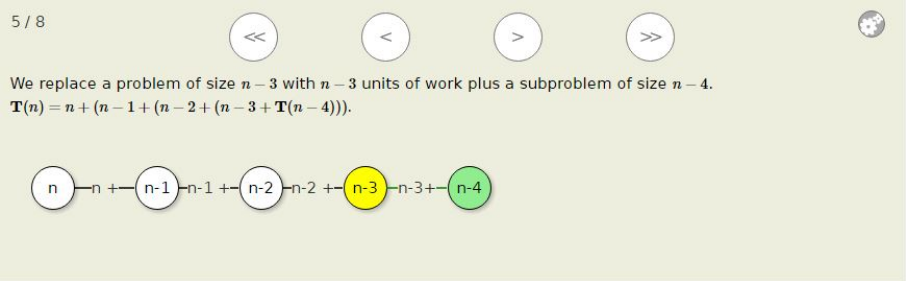


Figure A.5: Unrolling the recurrence $T(n) = T(n-1) + n$

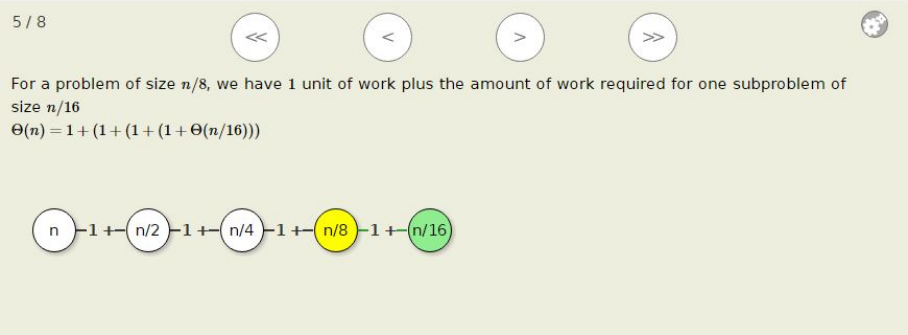


Figure A.6: Unrolling the recurrence $T(n) = T(n/2) + 1$

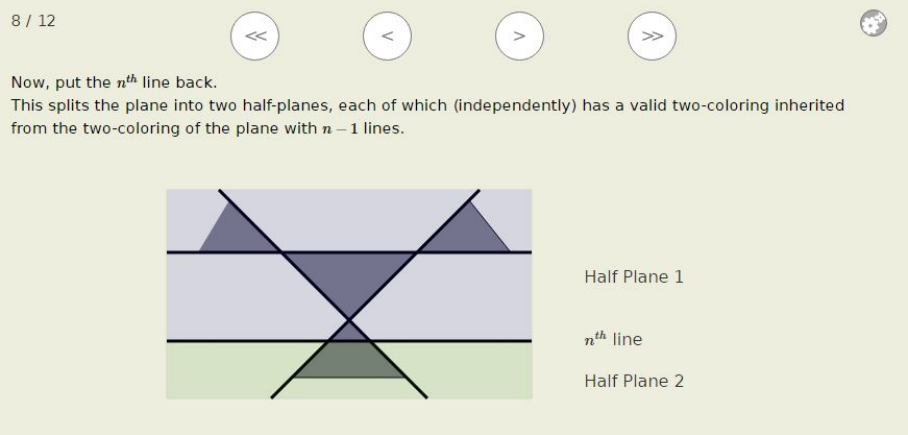


Figure A.7: Proof by Induction example

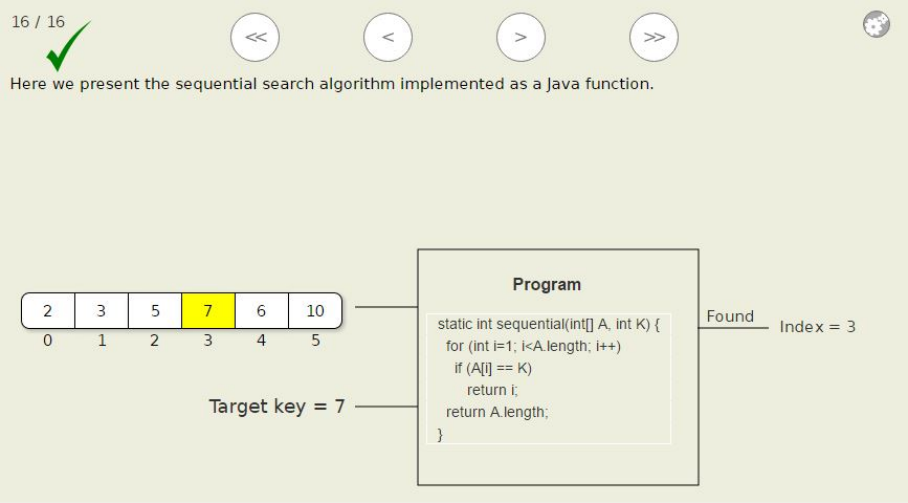


Figure A.8: Problems, Algorithms, and Programs

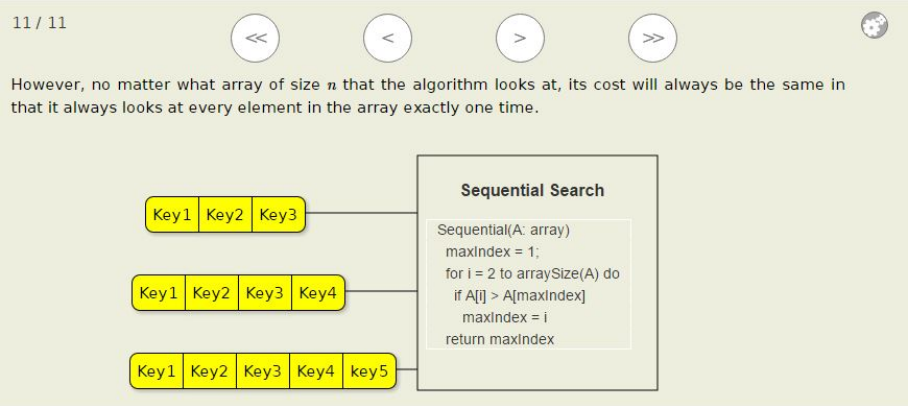


Figure A.9: Best, Average, and Worst Cases 1

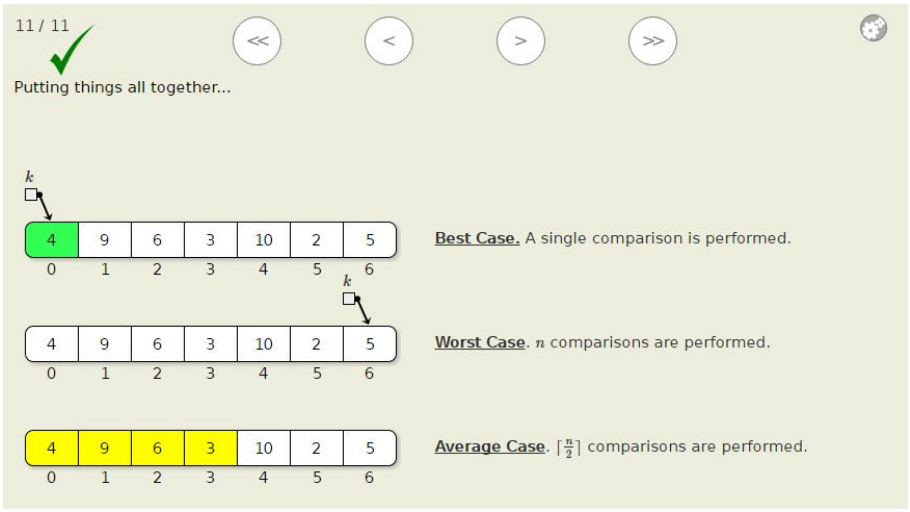


Figure A.10: Best, Average, and Worst Cases 2

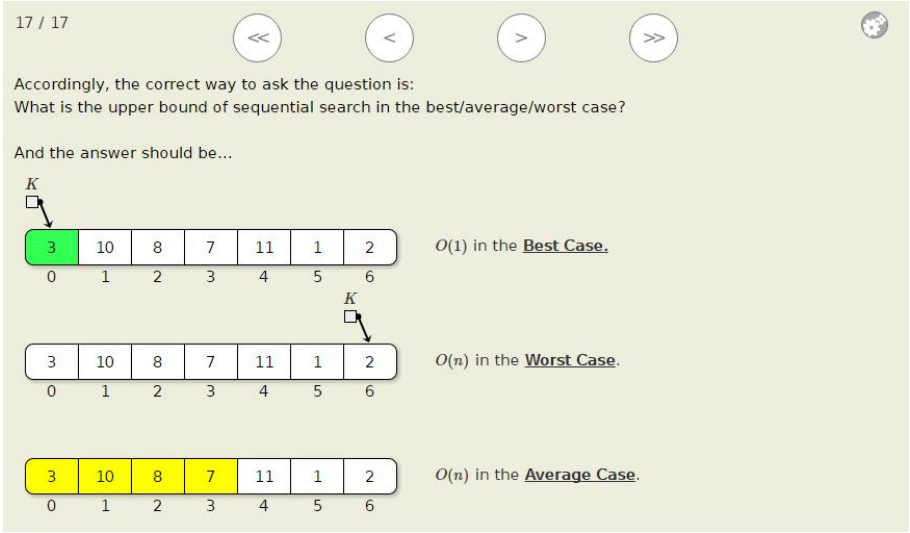


Figure A.11: Upper Bounds Definition

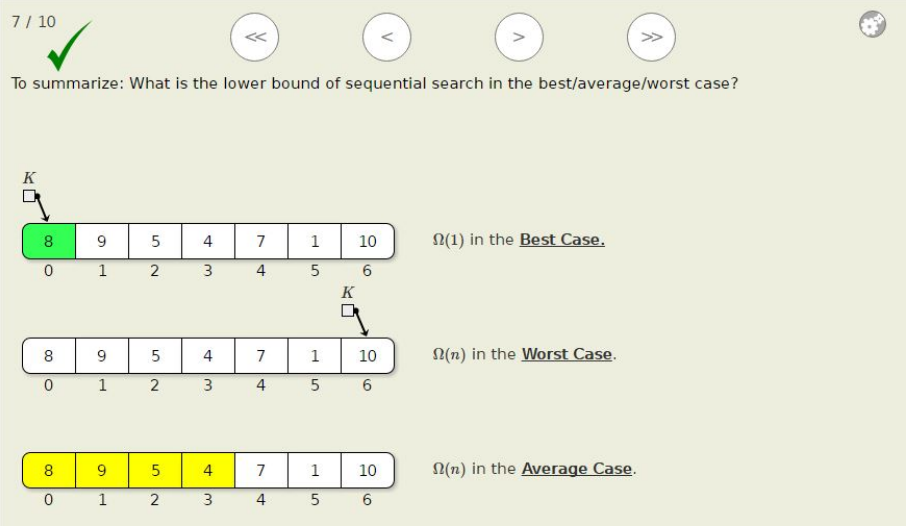


Figure A.12: Lower Bounds Definition

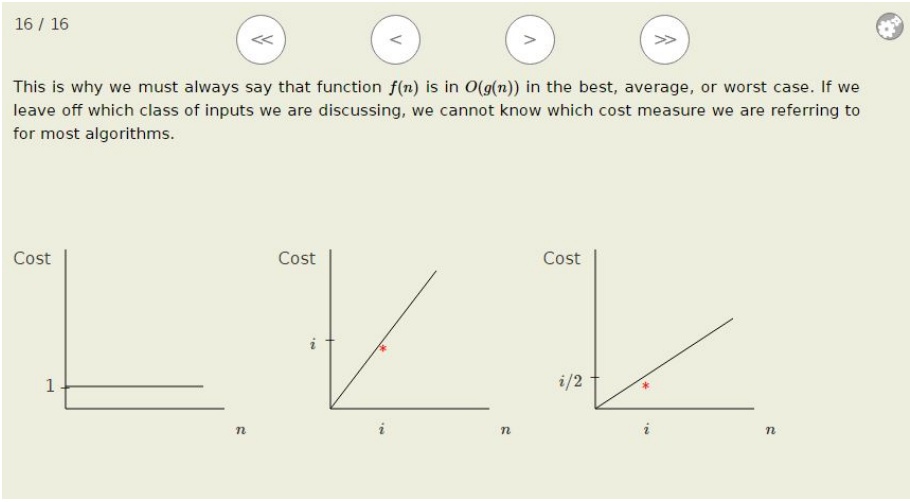


Figure A.13: Cases Versus Bounds

A.2 Sorting AAVs

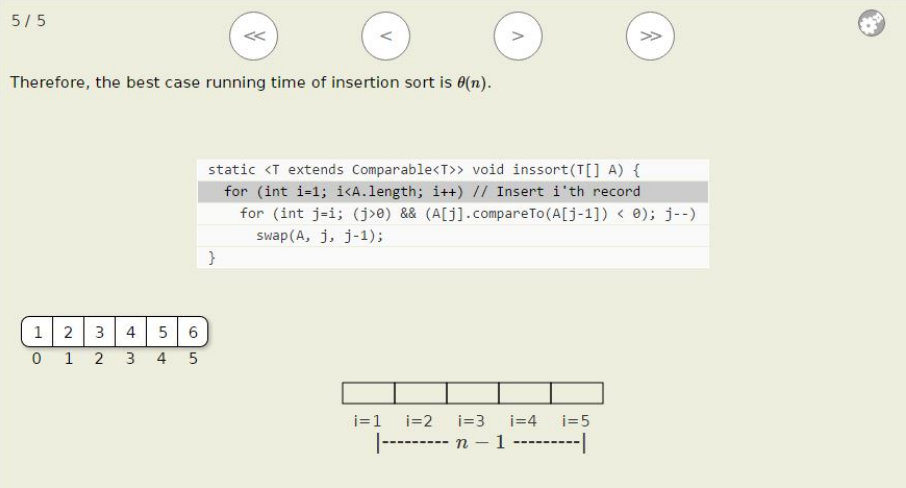


Figure A.14: Insertionsort Best-case running-time analysis

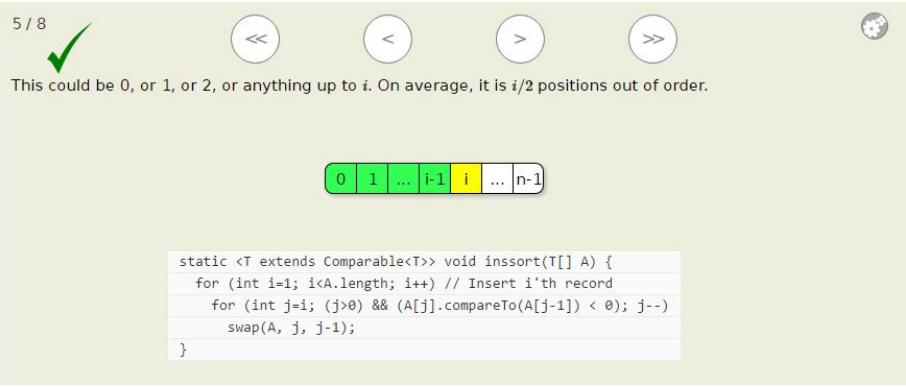


Figure A.15: Insertionsort Average-case running-time analysis

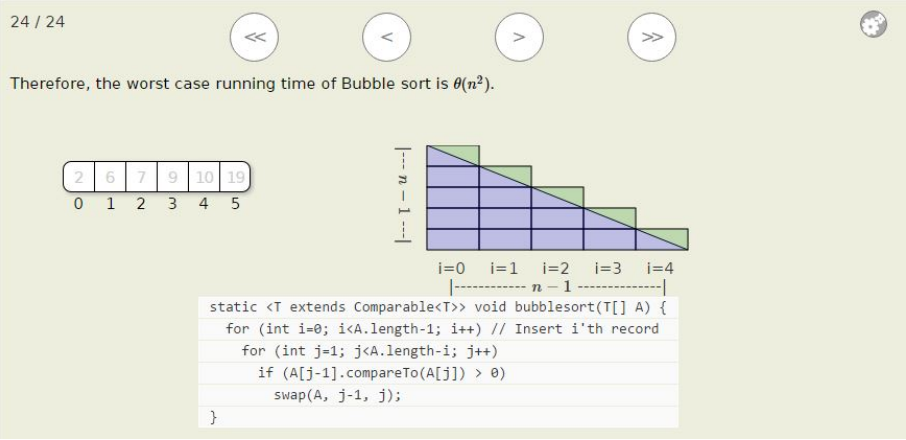


Figure A.16: Bubblesort running-time analysis

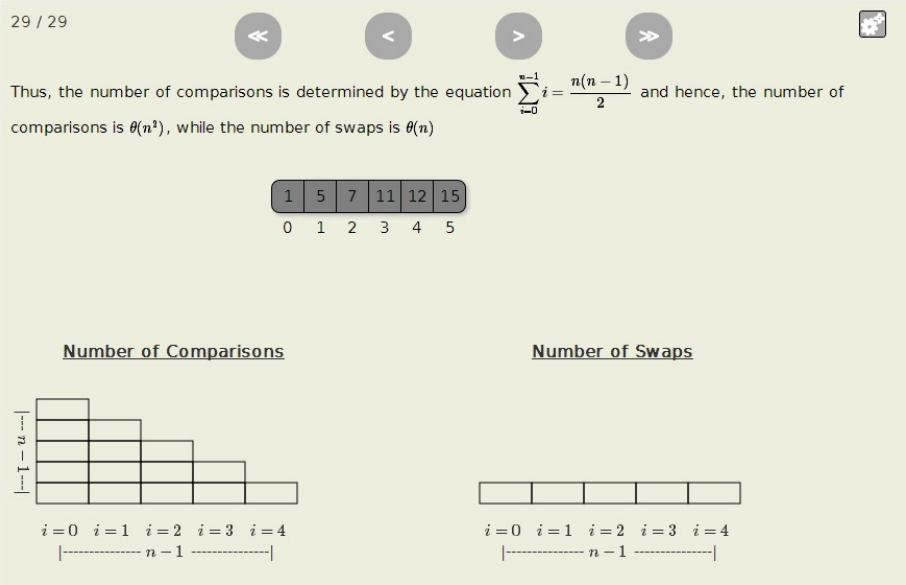


Figure A.17: Selectionsort running-time analysis

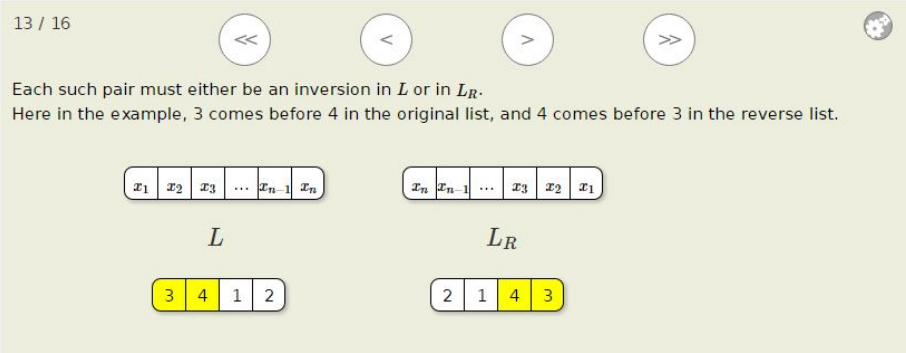


Figure A.18: The cost of Exchange Sorting analysis

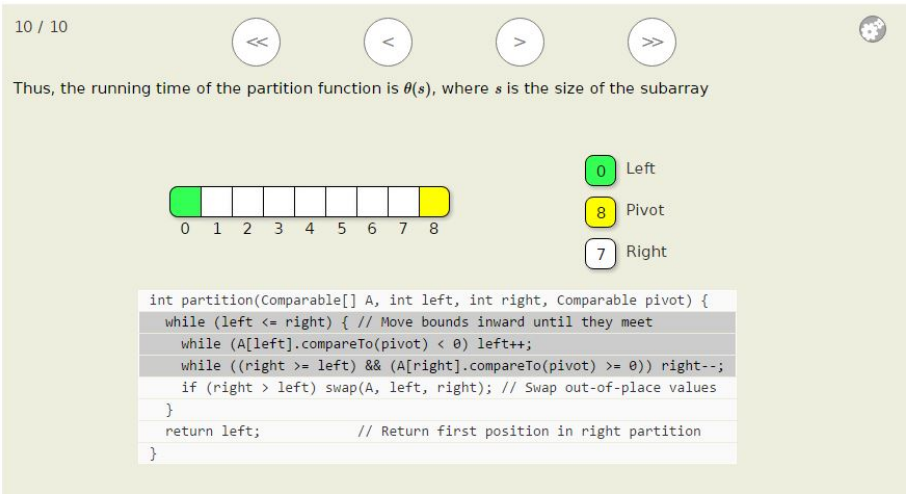


Figure A.19: Quicksort Partition running-time analysis

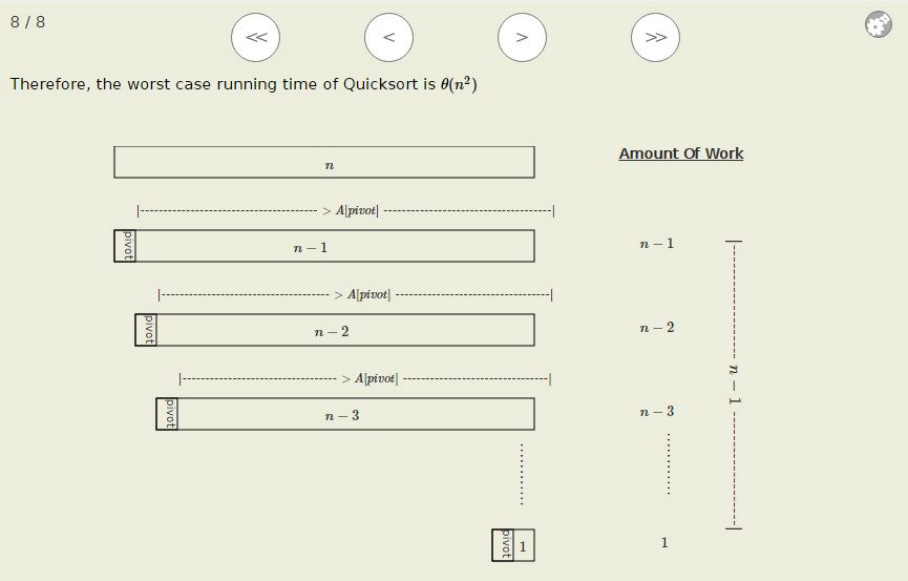


Figure A.20: Quicksort Worst-case running-time analysis

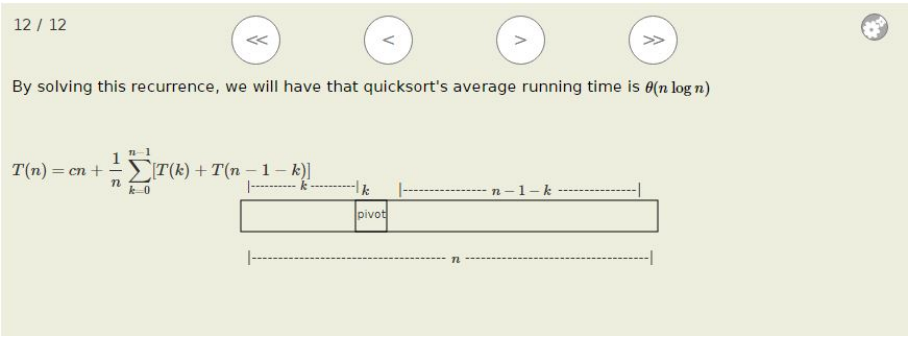


Figure A.21: Quicksort Average-case running-time analysis

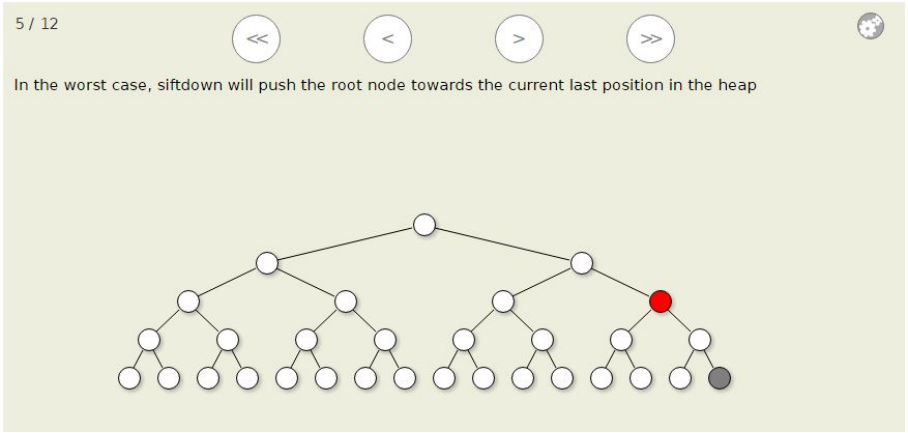


Figure A.22: Heapsort running-time analysis

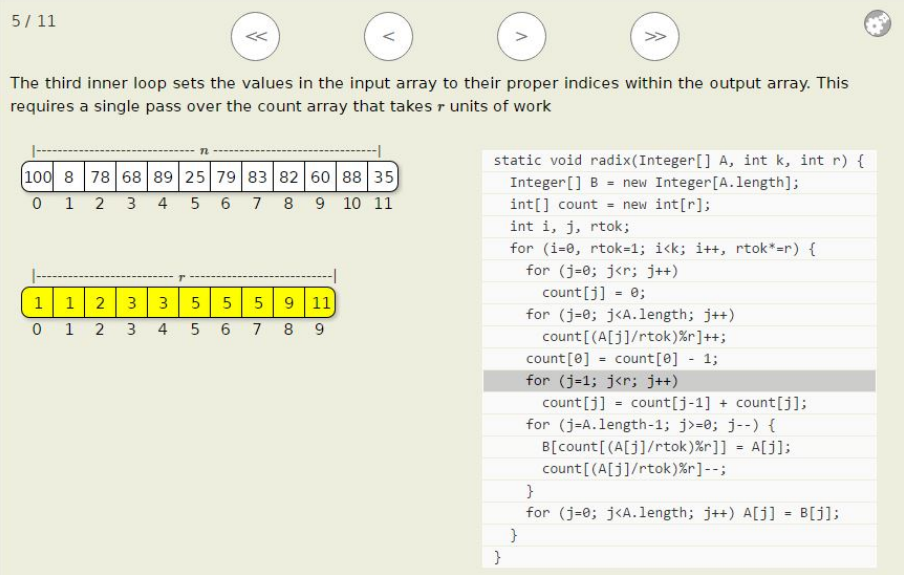


Figure A.23: Radixsort running-time analysis

Appendix B

The Algorithm Analysis Concept Inventory Items

Here, we present out initial AACI items as administered in Virginia Tech (VT) and Christopher New Port (CNP) during Fall 2015 and Spring 2016. The concept each question targets is displayed in parenthesis next to the question number. Each question's rubric is displayed below indicating the misconceptions associated with each wrong answer.

B.1 CNP Fall 2015, VT Fall 2015, and CNP Spring 2016 administrations

- **Item1: (C22)**
  Below are two Java functions to compute $(x^y)^z$.

  Function A

```
public int TwoPowers(int x, int y,
int z) {
  int i, result = 1;
  for (i = 1; i <= y * z; i++) {
    result *= x;
  }
  return result;
}
```

  Function B

```
public int TwoPowers(int x, int y,
int z) {
  int i, result = 1, finalResult = 1;
  for (i = 1; i <= y; i++) {
    result *= x;
  }
  for (i = 1; i <= z; i++) {
    finalResult *= result;
  }
  return finalResult;
}
```

Which function, A or B, do you think usually runs faster? Justify your answer.

(a) Function A usually runs faster.
(b) Function B usually runs faster.
(c) Both functions take about the same amount of time.

Table B.1: Item 1 Rubric.

Answer	**Justification**	**Misconception**
(a)	Algorithm A is simpler and has less code	(M1)
	Algorithm A has fewer variables than Algorithm B	(M2)
	Algorithm A has a single loop, while Algorithm B has two loops	(M16)
(b)	Algorithm B runs in $\mathcal{O}(y + z)$, while Algorithm A runs in $\mathcal{O}(y * z)$	correct
(c)	Both algorithms are doing the same task	(M4)

- **Item2: (C22)**
  Below are two functions to raise the product of the elements of an array of integers to the power of y.

Function A

```
public int RaiseByY(int [] arr,
int y) {
  int result = 1, i, j, mul;
  for(i = 0; i < arr.length; i++) {
    mul = 1;
    for(j = 0; j < y; j++) {
      mul *= arr[i];
    }
    result *= mul;
  }
  return result;
}
```

Function B

```
public int RaiseByY(int [] arr,
int y) {
  int result = 1, i, j, mul;
  mul = 1;
  for(i = 0; i < arr.length; i++) {
    mul *= arr[i];
  }
  for(j = 0; j < y; j++) {
      result *= mul;
  }
  return result;
}
```

Which function, A or B, do you think usually runs faster? Justify your answer.
(a) Function A usually runs faster.
(b) Function B usually runs faster.
(c) Both functions take about the same amount of time.

Table B.2: Item 2 Rubric.

Answer	**Justification**	**Misconception**
(a)	...	other
(b)	Algorithm A has two nested loops with an overall running time complexity of $\mathcal{O}(n * y)$, while Algorithm B has an overall running time complexity of $\mathcal{O}(n + y)$	correct
(c)	Both Algorithms have the same statements or number of statements	(M3)
	Both Algorithms are doing the same task	(M4)
	Both algorithms have the same number of loops	(M16)

- **Item3: (C11) (C12) (C13)**
  Mark all statements below about the upper, lower, and tight bounds of an algorithm as True or False. For each false statement, either correct it or state why it is false.
  (a) The upper bound of an algorithm is the growth rate that the algorithm has in its worst case. (C11)
  (b) The lower bound of an algorithm is the growth rate that the algorithm has in its best case. (C12)
  (c) When we know the exact running time for an algorithm in the worst case, then the worst case upper and lower bounds for the algorithm will always be the same. (C13)
  (d) An upper bound to the running time of an algorithm in the worst case is any function that is always equal to or greater than the running time of that algorithm in the worst case. (C11)
  (e) A lower bound is the least amount of a time that the algorithm needs for an input of size n. This can be found for the algorithm's best, average, or worst cases. (C12)
  (f) Θ is the notation used to describe the amount of a time required by the algorithm in the average case. (C13)
  (g) We use the Big-O, and Big-Ω notations to model the running time of an algorithm in its worst, and best cases respectively. (C11)(C12)

Table B.3: Item 3 Rubric.

Part	Answer	Correction	Misconception
(a)	True		(M7)
	False	Upper bound shouldn't be confused with worst case	correct
	False	other	other
(b)	True		(M8)
	False	Lower bound shouldn't be confused with best case	correct
	False	other	other
(c)	True		correct
	False		(M5) or other
(d)	True		correct
	False		(M5) or other
(e)	True		correct
	False		(M5) or other
(f)	True		(M9)
	False	Θ is used as a notation for tight bounds. The shouldn't be a confusion between tight bounds and average cases	correct
	False	other	other
(g)	True		(M7) and (M8)
	False	Big-$\mathcal{O}$ and Big-Ω are notations used for upper and lower bounds respectively.	correct
	False	other	other

- **Item4: (C21)**
  The **sequential search algorithm** takes an array of integers and a target value. It searches for the target value within the array and returns its index if it is found, or returns the size of the array if not found. For each case below write whether that case is the best or worst case of the algorithm. You can answer "None" if you feel that the example is neither the best nor worst case for the algorithm. Justify your answer.
  (a) The target value is located near the front of the array.
  (b) The input array contains only a single element.
  (c) The input array is very large.
  (d) The target value is located near the end of the array.

Table B.4: Item 4 Rubric.

Part	**Answer**	**Misconception**
(a)	Best Case	correct
	Worst Case	(M17)
	None	(M17)
(b)	Best Case	(M10)
	Worst Case	(M17)
	None	correct
(c)	Best Case	(M17)
	Worst Case	(M10)
	None	correct
(d)	Best Case	(M17)
	Worst Case	correct
	None	(M17)

- **Item5: (C16) (C17) (C18)**
  Mark all statements below about the upper and lower bounds of problems as True or False.
  Justify your answer.
  (a) The upper bound for the sorting problem is defined as the cost of the best algorithm we know in the worst case. (C16)
  (b) The lower bound of the sorting problem is defined as the cost of the best algorithm we know in the best case. (C17)
  (c) For the sorting problem, the upper bound is defined as the running time of the best algorithm we know, while the lower bound is defined as the least amount of time that any algorithm could require. (C16) (C17)
  (d) The lower bound for the sorting problem can be defined as the best growth rate of one of the algorithms that solve the problem. C17 (C18)

Table B.5: Item 5 Rubric.

Part	Answer	Correction	Misconception
(a)	True		(M7)
	False	Problem upper bound is defined as the cost of the best algorithm solves the problem we know about. Upper bounds shouldn't be confused with worst cases	correct
	False	other	other
(b)	True		(M8)
	False	Problem lower bound is defined as the least cost that any algorithm solves the problem could reach. Lower bounds shouldn't be confused with best cases	correct
	False	other	other
(c)	True		correct
	False		(M5)
(d)	True		(M11)
	False	Problem lower bound shouldn't be confused with algorithm lower bound	correct
	False	other	other

- **Item6: (C22)**
  Provide the running time for the following code snippets in terms of n.
  (a)
```
for(i = 1; i <= n; i++)
  for(j = 1; j <= n/2; j++)
    System.out.print(i*j);
```

  (b)
```
for(i = 1; i <= n; i++)
  for(j = 1; j <= i; j++)
    System.out.print(i*j);
```

  (c)
```
for(i = 1; i <= n; i++)
  for(j = 1; j <= n; j=j*2)
    System.out.print(i*j);
```

- **Item7**: (C10) (C11) (C12) (C13) Suppose that $f(n) = \sqrt{n}\log n$, $g(n) = n\sqrt{n}$, and $h(n) = 2^n$. Mark all the statements below as True or False. Justify your answer.
  (a) $f(n) \in O(g(n))$ (C7) (C10) (C11)
  (b) $g(n) \in O(f(n))$ (C7) (C10) (C11)
  (c) $h(n) \in \Omega(g(n))$ (C7) (C10) (C12)
  (d) $g(n) \in \Theta(f(n))$ (C7) (C10) (C13)

Table B.6: Item 6 Rubric.

Part	**Answer**	**Misconception**
(a)	$n^2/2$	correct
	$n^n/2$	(M14) or (other)
	$n \log n$	New Misconception. They think that running the loop to half of the input size will contribute to a logarithmic running time
(b)	$\frac{n(n+1)}{2}$ or $\mathcal{O}(n^2)$	correct
	$n!$	(M14) or (other)
(c)	$n \log n$	correct
	n^2	(M15)

Table B.7: Item 7 Rubric.

Part	**Answer**	**Justification**	**Misconception**
(a)	True	$f(n) \leq cg(n) \forall n \geq 4$ and $c = 1/2$	correct
	True	other	(M6)
	False	other	(M6)
(b)	True	other	(M6)
	False	You cannot find n_0 and c such that $g(n) \leq cf(n) \forall n \geq n_0$	correct
	False	other	(M6)
(c)	True	$h(n) \geq cg(n) \forall n \geq 4$ and $c = 2$	correct
	False	You cannot find n_0 and c such that $h(n) \geq cg(n) \forall n \geq n_0$	(M6)
	False	other	(M6)
(d)	True	other	(M6)
	False	We don't have both $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$	correct
	False	other	(M6)

- **Item8: (C7) (C10)**
  Which of the following statements is true?
  (I) The ratio $n/n\log_2 n$ is the same as $n\log_2 n/n^2$.
  (II) The ratio $\log_2 n/n$ is the same as $n/2^n$.
  (a) (I) Only.
  (b) (II) Only.
  (c) (I) and (II).
  (d) Neither (I) nor (II).

Table B.8: Item 8 Rubric.

Part	**Answer**	**Misconception**
(a)	True	(M12) and (M13)
	False	correct
(b)	True	correct
	False	(M12) or (M13)
(c)	True	(M12)
	False	correct
(d)	True	(M13)
	False	correct

- **Item9**: (C7) (C10) (C11) (C12) Look at the following figure and then fill in the blanks with the appropriate asymptotic notation (O, Ω, or Θ).
  (a) $f(x) \in$ _____$(g(x))$
  (b) $f(x) \in$ _____$(h(x))$
  (c) $h(x) \in$ _____$(g(x))$

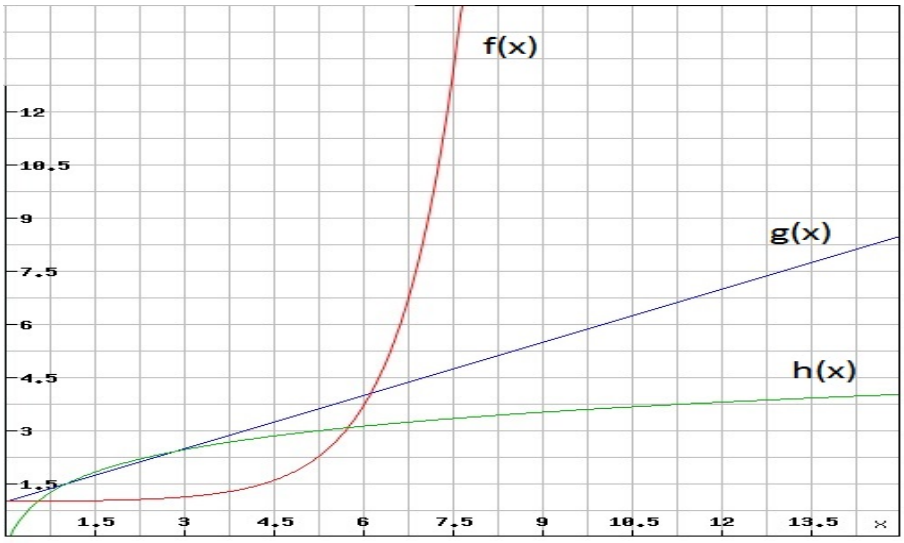


Figure B.1: Functions f(x), g(x), and h(x)

Table B.9: Item 9 Rubric.

Part	Answer	Misconception
(a)	$f(x) = \Omega(g(x))$	correct
	$f(x) = \mathcal{O}(g(x))$	(M5) or (M6)
(b)	$f(x) = \Omega(h(x))$	correct
	$f(x) = \mathcal{O}(h(x))$	(M5) or (M6)
(c)	$h(x) = \mathcal{O}(g(x))$	correct
	$h(x) = \Omega(g(x))$	(M5) or (M6)

- **Item10**: (C24) (C26)
  Suppose that you have an array of n records to sort. Which of the following sorting algorithms is the best choice of the four:
  (a) Insertionsort, (b) Heapsort, (c) Quicksort, (d) Mergesort

  Given the following two considerations:
  (i) The array is so big that it nearly fits your available memory (so you are not allowed to use a lot of extra space), and
  (ii) You want the algorithm with the fastest worst case running time.
  Justify your answer.

Table B.10: Item 10 Rubric.

Part	Justification	Misconception
(a)	——	M17 or other
(b)	Heapsort has an nlogn worst case running time which is faster than both quicksort's and insertionsort's worst case running time of n^2. We cannot use mergesort here since the array is nearly fitting the memory and mergesort needs an auxiliary array of size n	correct
	other	M17 or other
(c)	——	M17 or other
(d)	——	M17 or other

B.2 VT Spring 2016 administration

In VT Spring 2016 administration, the same questions described in the previous section were used. However, there were some changes to some of the questions. Here we show

only those questions found in VT Spring 2016 administration that were not found in other administrations.

- **Item5: (C16) (C17) (C18)**
  Mark all statements below about the upper and lower bounds of problems as True or False.
  Justify your answer.
  (a) The upper bound for the sorting problem is defined as the cost of the best algorithm that we know. (C16)
  (b) The lower bound of the sorting problem is defined as the cost of the best algorithm that we know .
- **Item10**: (C24) (C26)
  Consider these choices for algorithms:
  (i) Insertionsort, (ii) Quicksort, (iii) Mergesort
  (a) Which is the fastest in the best case.
  (b) Which is the fastest in the worst case.
  (c) Which is the fastest in the average case.

Appendix C

Student Surveys

Here we include OpenDSA student surveys given at the end of Fall 2014, Fall 2015 (No AAVs), and Spring 2016 (AAVs). We include only the parts of the surveys related to Algorithm Analysis.

C.1 Fall 2014, 2015 Survey

.

CS3114 Data Structures and Algorithms
Fall 2014, 2015 Survey

We are conducting a study of your experiences and opinions regarding various pedagogical innovations (such as OpenDSA) used in the course this semester. The principal investigators for this study are Dr. Cliff Shaffer and Dr. Stephen Edwards. There is no risk for you to participate in this study. The survey is conducted anonymously, and all data reporting will be in aggregate form. You may ask questions about this research by contacting Dr. Shaffer at (540) 231-4354 or shaffer@cs.vt.edu. For questions about your rights as a research participant, contact Virginia Techs Office of Research Administration at (540) 231-6866. Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. You must be 18 or older to take part in this research. Thank you for your participation.

1. I understand algorithm analysis concepts presented in the course.
   (a) Not at all confident
   (b) Not confident
   (c) Neutral
   (d) Confident
   (e) Very confident

2. Did you find OpenDSA useful for learning algorithm analysis concepts? Please explain.

3. Which is easier for you: understanding how an algorithm works, or determining the asymptotic running time of an algorithm that you understand? Do you think the difference in difficulty is intrinsic to the material, or driven by the style of presentation?

4. What do you think could be a good presentation technique that we can use to enhance OpenDSA content on algorithm analysis? Do you think visualizations can help?

C.2 Spring 2016 Survey

CS3114 Data Structures and Algorithms
Spring 2016 Survey

We are conducting a study of your experiences and opinions regarding various pedagogical innovations (such as OpenDSA) used in the course this semester. The principal investigators for this study are Dr. Cliff Shaffer and Dr. Stephen Edwards. There is no risk for you to participate in this study. The survey is conducted anonymously, and all data reporting will be in aggregate form. You may ask questions about this research by contacting Dr. Shaffer at (540) 231-4354 or shaffer@cs.vt.edu. For questions about your rights as a research participant, contact Dr. Moore, Associate Vice President for Research Compliance, at (540) 231-4991 or moored@vt.edu. Your participation is voluntary. If you do not wish to participate, simply do not fill out the survey. You must be 18 or older to take part in this research. Thank you for your participation.

1. How do you rate your proficiency in Mathematics necessary for your data structures course? This would include things like Discrete Math topics, induction proofs, logarithms, etc.
   (a) very high
   (b) high
   (c) moderate
   (d) low
   (e) very low

2. How confident are you about your understanding of the algorithm analysis concepts presented in the course(i.e., upper bounds, lower bounds, Big-O, bounds for problems).
   (a) Not at all confident
   (b) Not confident
   (c) Neutral
   (d) Confident
   (e) Very confident

3. Did you find the algorithm analysis visualizations (those related to the running time proofs, rather than the ones about how the algorithm works) useful for learning the algorithm analysis concepts? Please explain.

4. Which is easier for you: understanding how an algorithm works, or determining the asymptotic running time of an algorithm that you understand? Do you think the difference in difficulty is intrinsic to the material, or driven by the style of presentation?

5. What do you think could be a good presentation technique that we can use to enhance OpenDSA content on algorithm analysis? Do you think that the visualizations helped?

Appendix D

Student Interview Protocol

Individual Interview protocol for CS3114 Fall 2014

Introduction

- Signed consent form
- We are conducting these interviews to get perspectives from students at differing levels of performance in the class. You were selected from the pool of students for whom we still had Midterm 1 (so that we are able to discuss it with you).

Interview Questions

1. What aspects you think are important to study related to algorithm analysis?
2. What is your level of confidence about reading an algorithm and determining its asymptotic running time?
3. What do you think about when asked to determine the best algorithm to use in a particular situation?
4. Do you know the difference between the upper and lower bounds of an algorithm versus the upper and lower bounds of a problem?
5. Do you find any difficultly in understanding the algorithm analysis sections in OpenDSA? If yes, what features might give you a better understanding?
6. What questions do you feel were hard for you?
7. What was your confidence level about your responses?
8. Do you think that you know this material any better now (since we are near the end of the semester)?

Appendix E

List of Publications

E.1 Papers

- **Mohammed F. Farghally**, Kyuhan Koh, Hossameldin Shahin, and Clifford A. Shaffer. "Evaluating the Effectiveness of Algorithm Analysis Visualizations." Accepted at the 48^{th} ACM Technical Symposium on Computer Science Education, 2017.

- **Mohammed F. Farghally**, Kyuhan Koh, Jeremy V. Ernst and Clifford A. Shaffer. Towards a Concept Inventory for Algorithm Analysis Topics." Accepted at the 48^{th} ACM Technical Symposium on Computer Science Education, 2017.

- Fouh, Eric, **Mohammed F. Farghally**, Sally Hamouda, Kyu Han Koh, and Clifford A. Shaffer."Investigating Difficult Topics in a Data Structures Course Using Item Response Theory and Logged Data Analysis." In Proceedings of the 9^{th} International Conference on Educational Data Mining, Raleigh, NC, 2016, pages 370-375.

- Fouh, Eric, Daniel A. Breakiron, Sally Hamouda, **Mohammed F. Farghally**, and Clifford A.Shaffer. "Exploring students learning behavior with an interactive etextbook in computer science courses". Computers in Human Behavior 41 (2014): 478-485.

E.2 Book Chapters

- E. Fouh, S. Hamouda, **M.F. Farghally**, and C.A. Shaffer, "Automating Learner Feedback in an eTextbook for Data Structures and Algorithms Courses". In Challenges in ICT Education: Formative Assessment, Learning Data Analytics and Gamification, Santi Caball and Robert Claris, eds., Elsevier, 2016, 135-165.

E.3 Posters

- **Mohammed F. Farghally**, Eric Fouh, Sally Hamouda, Kyu Han Koh, and Clifford A. Shaffer. "Visualizing Algorithm Analysis Topics". In Proceedings of the 47^{th} ACM Technical Symposium on Computer Science Education, pp. 687-687, 2016.

E.4 Papers to appear

- Kyuhan Koh, Eric Fouh, **Mohammed F. Farghally**, Hossameldin Shahin, and Clifford A. Shaffer, Experience: Learner Analytics Data Quality for an eTextbook System. Submitted to the ACM Journal of Data and Information Quality, 2017.