Evaluating the Integration of Online, Interactive Tutorials into a Data Structures and Algorithms Course

Daniel A. Breakiron

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Clifford A. Shaffer, Chair
Stephen H. Edwards
Tahereh S. Hall

May 2, 2013
Blacksburg, Virginia

Keywords: OpenDSA, Electronic Textbook, Algorithm Visualization, Automatic Assessment

Evaluating the Integration of Online, Interactive Tutorials into a Data Structures and Algorithms Course

Daniel A. Breakiron

(ABSTRACT)

OpenDSA is a collection of open source tutorials for teaching data structures and algorithms. It was created with the goals of visualizing complex, abstract topics; increasing the amount of practice material available to students; and providing immediate feedback and incremental assessment. In this thesis, I first describe aspects of the OpenDSA architecture relevant to collecting user interaction data. I then present an analysis of the interaction log data gathered from three classes during Spring 2013. The analysis focuses on determining the time distribution of student activity, determining the time required for assignment completion, and exploring "credit-seeking" behaviors and behavior related to non-required exercises.

We identified clusters of students based on when they completed exercises, verified the reliability of estimated time requirements for exercises, provided evidence that a majority of students do not read the text, discovered a measurement that could be used to identify exercises that require additional development, and found evidence that students complete exercises after obtaining credit. Furthermore, we determined that slideshow usage was fairly high (even when credit was not offered), and skipping to the end of slideshows was more common when credit was offered but also occurred when it was not.

This work was supported in part by NSF grants DUE-1139861 and IIS-1258571.

Acknowledgments

While this project would not have been possible without the efforts of many people, there are several individuals who deserve special thanks.

First, I would like to thank my advisor, Dr. Clifford A. Shaffer, both for his guidance and his work managing and creating content for the OpenDSA project, and Drs. Stephen H. Edwards and T. Simin Hall for serving on my thesis committee.

Next, I would like to recognize Eric Fouh, a PhD student at Virginia Tech, for his work on the OpenDSA Data Collection Server and database system, and Dr. Ville Karavirta from Aalto University in Finland for creating the JavaScript Algorithm Visualization (JSAV) library upon which many of the OpenDSA exercises are based.

I would also like to show my appreciate to the National Science Foundation who partially supported my work through grants DUE-1139861 and IIS-1258571, and Virginia Tech for providing me a Graduate Research Assistantship.

Thanks to my friends and family for their support, and special thanks to my girlfriend Caitlyn Kingry, not only for her support but also for her help as a data analysis consultant.

Finally, I would like to extend my thanks to Dwight Barnette, Dr. Yang Cao, and Dr. Hicham Elmongui for incorporating OpenDSA into their classes, and to all the students who used OpenDSA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of education as a whole is resistant to change and slow to adopt new technologies. In this chapter we will describe several shortcomings with the traditional models for university courses and textbooks. We propose a solution, OpenDSA, to address these shortcomings. In the remainder of the paper we provide a deeper understanding of the goals and capabilities of OpenDSA. The major contribution of this thesis is an analysis of data collected from OpenDSA's deployment in three classes during the Spring 2013 semester.

1.1 Traditional Course Model

Traditionally university courses center around lectures given by a professor with assigned reading from a textbook, practice problems for homework, and several tests administered to assess student understanding. While this paradigm has obviously been effective, it is not without limitations. While students may be given practice problems to reinforce material learned in class, the time consuming nature of many of these assignments, both for students and for graders, mean they can only cover a small amount of the material that students are expected to understand. Students require feedback on their performance to either verify their comprehension or identify where they made mistakes. Identifying and addressing student's misconceptions early is vital to ensuring students have a strong foundation on which to build future knowledge and will help reduce their frustration.

Unfortunately, under a traditional model based on paper textbooks, homework has been graded manually with feedback provided by the professor or teaching assistants (TAs). Due to this manually intensive process, it can take several weeks for students to receive feedback, if they receive any at all. When they do, the quality of the feedback may vary between professors or even between TAs within the same course. Students who receive more helpful information from one TA are certainly in a better position to succeed than students in the same class who receive less helpful feedback from another TA. As class sizes grow, this model

quickly becomes unsustainable.

Another limitation is the finite set of examples and practice problems available to students under this model. Some students rely heavily on examples to learn material and the limited number presented in class or in the textbook may not be sufficient for students to gain comprehension, especially when instructors use the same textbook examples in their lectures. Likewise, whether homework problems come from the instructor or a textbook, the limited number of problems covering a particular topic in turn limits the amount a student is able to practice. Taking into consideration the fact that students will likely only receive feedback for the assigned problems, the amount of practice from which a student can gain useful experience is further diminished.

1.2 Electronic Textbooks

Like the traditional course model, textbooks have also been slow to embrace the benefits of digital technology. Publishers now routinely offer electronic versions of textbooks as PDFs or popular E-reader formats such as those used by the Amazon Kindle or Barnes and Noble's Nook. However, these static, electronic textbooks, which we refer to as e-texts, offer few advantages over their paper equivalents. While cheaper, easier to distribute, and easier to access, they generally provide content identical to their physical counterparts and can still be expensive. These e-texts are a step in the right direction, but they do not take full advantage of the digital medium. Rather than static images, e-texts could include videos, and instead of static text, e-texts could easily be updated to correct errors or improve information. Unfortunately, many of these capabilities are either impractical or not supported by E-reader-style books.

1.3 Overview of the Problem Space

We identify the following problems with the current standard course and textbook models:

- Too few examples and practice problems
- Practice problems are not sufficiently comprehensive
- Students do not receive helpful feedback in a timely manner
- Manual grading and feedback techniques do not scale to larger class sizes
- Textbooks are expensive, static and not engaging
- Textbooks do a poor job of conveying dynamic processes (like algorithms)

1.4 OpenDSA

The OpenDSA project [12, 13] focuses on addressing the shortcomings discussed above for courses in the area of data structures and algorithms. OpenDSA is a collection of online, open-source tutorials which combine textbook-quality text with algorithm visualizations and randomly generated instances of interactive examples and exercises to provide students with unlimited practice. The exercises provide immediate feedback on a per-step or per-question basis and display an indicator when the student has achieved proficiency with the exercise as a whole. While the list of topics currently covered by practice problems is relatively small, the framework supports adding exercises on any topic, making it possible for instructors to create more comprehensive assignments. Since feedback is handled by the exercise itself and grading is handled automatically by the OpenDSA framework, this technique can easily scale to classes of any size. The online, open-source nature of the tutorials means that OpenDSA is free and available to anyone with an internet connection. Existing instances of OpenDSA can be rebuilt at any time to incorporate new or updated content.

Another advantage of OpenDSA over traditional textbooks is its ability to collect data about how it is used. During the Spring 2013 semester, we collected and analyzed usage data from three classes as part of an observational case study. Chapter 6 presents the results of this study as it pertains to our research questions about the order and time distribution of student activity, time required for assignment completion, "credit-seeking" behaviors, and behavior related to non-required exercises.

Chapter 2

Background

This chapter presents some previous work that inspired OpenDSA, the motivating forces behind its creation, and key results from our earlier quasi-experimental pilot study conducted in Fall 2012.

2.1 TRAKLA2

2.1.1 Overview

TRAKLA2 [5] is a framework for presenting interactive algorithm simulation exercises. It provided much of the inspiration for OpenDSA's proficiency exercise infrastructure. In TRAKLA2, exercises are implemented as Java applets which allow students to interact with a visual representation of a data structure in order to simulate the step-by-step operation of an algorithm. Automatic assessment and immediate feedback are provided in the form of a "Grade" button, which compares the state of the student's data structure with that of the model answer. When they are finished, students can click the "Submit" button to submit their solution and see the number of points they received out of the total available. TRAKLA2 does not impose a limit on the number of times each exercise can be attempted and submitted, however, a specific instance of an exercise can only be submitted once. Exercises also include a visual model answer which walks students step-by-step through the execution of the algorithm, though by viewing the model answer students forfeit the ability to submit their solution to the given exercise instance. Textual content centers around and is linked to from specific exercises. Finally, TRAKLA2 possesses a data logging feature which records information about how students use the applet, focusing especially on GUI operations.

OpenDSA replicates much of TRAKLA2's functionality while making some improvements. Unlike TRAKLA2, OpenDSA exercises are written using HTML5, CSS and JavaScript which

are more widely supported and arguably easier to develop than Java applets. Support for automatic assessment and immediate feedback is provided by the JavaScript Algorithm Visualization (JSAV) library [4, 12] which was created and is maintained by one of the TRAKLA2 developers. While OpenDSA exercises still sport a "Grade" button, its functionality has been entirely replaced by a persistent score display which displays a student's current score throughout the exercise. To reduce the number of steps involved and keep the interface minimal, OpenDSA exercises forgo a "Submit" button, opting instead to automatically submit a student's attempt when he or she completes an exercise. Additionally, TRAKLA2's feedback is limited to the student's final score once they complete an exercise, while OpenDSA is typically configured to provide feedback on a step-by-step basis. [4]

Like TRAKLA2, exercises can be attempted an unlimited number of times. Since exercise instances are randomly generated, the likelihood of a repeated instance is negligible. Unlike TRAKLA2, OpenDSA is content-centric with exercises acting as a supplement to the text. OpenDSA provides a similar model answer capability, including the same restriction on receiving credit. All OpenDSA exercises are tied into an application-wide data logging mechanism that is described in depth in Section 3.2.

A problem encountered with TRAKLA2 was that, as a Java applet, it required browsers to support Java. Also, some firewalls blocked communication between the applet and the server. [5] OpenDSA does not suffer from these issues since content is rendered using HTML5, CSS, and JavaScript, and all server communications use HTTP/S which are only blocked by the most stringent of firewalls.

2.1.2 Evaluation

A study by Silvasti *et al.* [14] evaluated the data collected by the TRAKLA2 logging mechanism in order to better understand students' learning processes. They used the number of exercise initializations and model answer views to measure the difficulty of an exercise. Then they determined how many attempts it took students to obtain the maximum number of points for an exercise. They also used the number of times each model answer step was viewed to determine the most difficult steps within an exercise. While their study is similar to ours, the two investigate different aspects of student behavior and provide a good compliment to each other. As discussed in Section 7.2.2, we hope to use similar techniques in the future to identify confusing concepts.

2.2 Origins of OpenDSA

The original motivation for OpenDSA grew out of research into Algorithm Visualization (AV), a technique for improving Computer Science instruction by creating visual representations of data structures and the processes that manipulate them. [11] Studies have shown

that despite strong support for AVs expressed by instructors, adoption rates remain low. Survey results indicate instructors have difficulty finding good materials, lack time in classes or lack time to change the classes, and lack the understanding to use AVs effectively [7, 10]. It was believed that a collection of high-quality, online course materials could provide the solution to these issues. However, at the time, "there exist[ed] no complete electronic textbook, tightly integrated with AVs, that could be used as the primary learning resource in a semester-long computer science course in data structures and algorithms". [12] Shaffer *et al.* [12] determined that lack of practice exercises posed an even bigger problem than lack of dynamic AVs. They proposed that automatic assessment was necessary to provide sufficient practice. OpenDSA was created to fulfill these requirements.

2.2.1 Open Education

The Open Education movement seeks to leverage technology to improve the quality and availability of learning materials. In particular, followers believe educational resources should be interactive, engaging, continually updated, corrected, and improved, and available online for free, without limitations. [1] We are strong believers in this philosophy, and these tenants exist at the heart of OpenDSA. [12]

All OpenDSA content is released under an MIT open source license, which grants users full permission including the ability to use, copy and modify the content as long as the copyright and permission notice are maintained. [6] All aspects of the OpenDSA framework are open-source and available on GitHub. The OpenDSA content and client-side framework can be found at `http://github.com/cashaffer/OpenDSA`, while the server is located at `https://github.com/cashaffer/Aalto--`.

Part of the OpenDSA philosophy is that the effort required to create the envisioned eTextbook is huge, and an open-source project is the best way to enlist the community to accomplished the task. [12]

2.3 Pilot Study

OpenDSA was first deployed in a pilot study to one of Virginia Tech's two CS3114 Data Structures and Algorithms sections offered during Fall 2012. This class acted as the treatment group in a quasi-experiment, while the other acted as a control. Both groups spent 3 weeks covering sorting and hashing topics, but the control group received traditional lectures and used a traditional textbook while the treatment group used OpenDSA and received a combination of lecture and class time devoted to completion of OpenDSA content. An identical midterm exam was administered to both classes at the end of the experiment, in addition to a pre- and post-survey which assessed students' attitudes towards using an electronic textbook to learn data structures and algorithm topics. [3]

While the treatment group performed slightly better overall on the midterm, the results were not statistically significant. The pre-survey showed that most students in both sections had previous experience with online courseware and an overall positive attitude towards its potential despite expressing dislike for certain previous experiences. The post-survey results of the treatment group indicated in increase in positive attitude towards online tutorials. [3]

During interviews with three students from the treatment group, all expressed positive feedback and indicated that OpenDSA had helped them better retain what they learned. Other students expressed their disappointment that OpenDSA was only used for the sorting and hashing sections. The evidence collected indicates students spent roughly the same amount of time and effort on OpenDSA as in a traditional class but that they prefer and are even excited to use OpenDSA. [3]

The pilot study dealt primarily with qualitative data obtained from surveys and discussions with students while only touching the surface of the quantitative data available for analysis. This thesis seeks to expand upon the results of the pilot by performing a quantitative analysis of student interaction with OpenDSA.

Chapter 3

OpenDSA

When I became involved with the OpenDSA project, some content had already been created and a mechanism was in place to compile the book from the various source files. However, the system lacked a mechanism to collect and store student scores and interaction data to a central location which would be required in order for OpenDSA to be used in a classroom environment. Since OpenDSA is a web application, a client-server architecture was chosen for this purpose, and the design and implementation of the client-side portion of this infrastructure became my primary focus. Below is an overview of how OpenDSA works and details about the client-side framework that I developed.

3.1 Overview

3.1.1 Content Structure

Content within OpenDSA is organized into modules, each of which focuses on a specific topic such as Shellsort or Bucket Hashing. All modules are written using reStructuredText [9], a plaintext markup language, and compiled into HTML using Sphinx [2], a tool originally designed to generate Python documentation. These modules are grouped into chapters, and the main navigation page currently resembles the table of contents of a standard textbook, as seen in Figure 3.1. Each module is composed of a combination of text, images, and interactive activities which include mini-slideshows, full AVs, proficiency exercises and Khan Academy-style exercises [8]. All of these interactive features will collectively be referred to as exercises for the remainder of this paper, but occasionally they will be categorized by type with "SS" referring to both mini-slideshows and AVs, "PE" referring to proficiency exercises, and "KA" referring to Khan Academy-style exercises.

All exercises are written using HTML5, CSS, and JavaScript with extensive use of jQuery. Visualizations are written using the JavaScript Algorithm Visualization (JSAV) library writ-

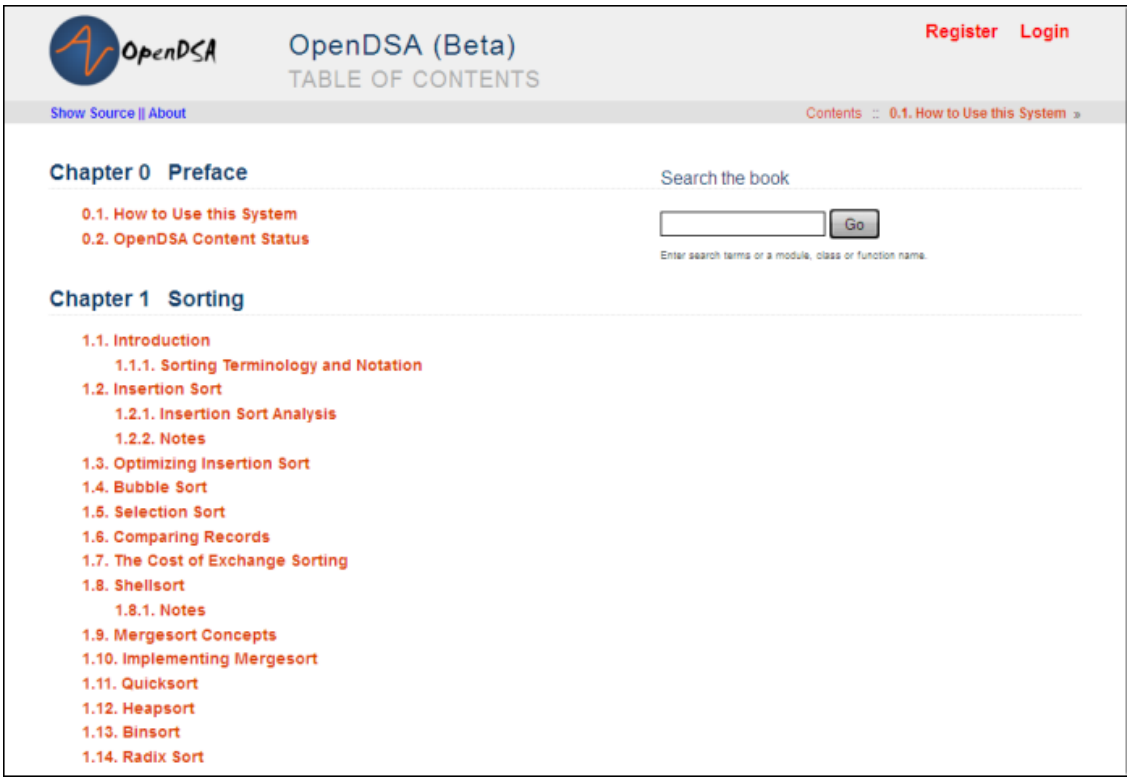


Figure 3.1: OpenDSA Navigation Page: The Table of Contents

ten by Dr. Ville Karavirta from Aalto University in Finland [4]. AVs, proficiency exercises, and to a lesser extent the Khan Academy-style exercises were designed to be self contained so that they are able to run independently or in conjunction with OpenDSA (through iframe embedding). This ensures maximum return on effort invested in AV and exercise creation since they can be used individually regardless of whether OpenDSA as a whole is adopted. The Khan Academy-style questions utilize an open source portion of the Khan Academy framework [8] to provide support for multiple choice, True / False and simple proficiency style exercises. Additionally, as part of their stand-alone nature, all exercises are capable of self-grading and providing some form of feedback to the user. AVs display the current slide number as well as the total number of slides. Proficiency exercises display the student's current points, the total points, the number of points still available, and the number of points lost. Khan Academy exercises show either a smiling or a frowning face to indicate a correct or incorrect answer. Khan Academy exercises also provide a status bar displaying a student's progress since that type of exercise requires student's to do several questions to complete. Examples of each exercise type can be seen below in Figures 3.2, 3.3, 3.4, and 3.5.

Unlike the traditional A-F grading scale, OpenDSA uses mastery-based grading. This is a boolean system with the student receiving no credit until he or she masters the material presented in an exercise, at which time they receive credit and are considered "proficient".

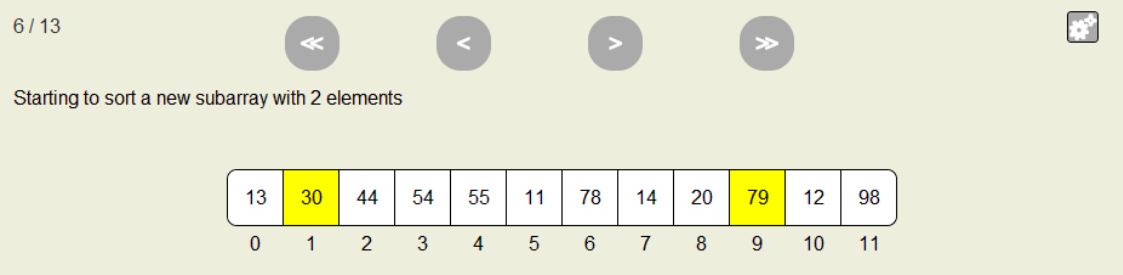


Figure 3.2: A "mini-slideshow": A Shellsort mini-slideshow demonstrating how sublists are sorting using Insertion Sort

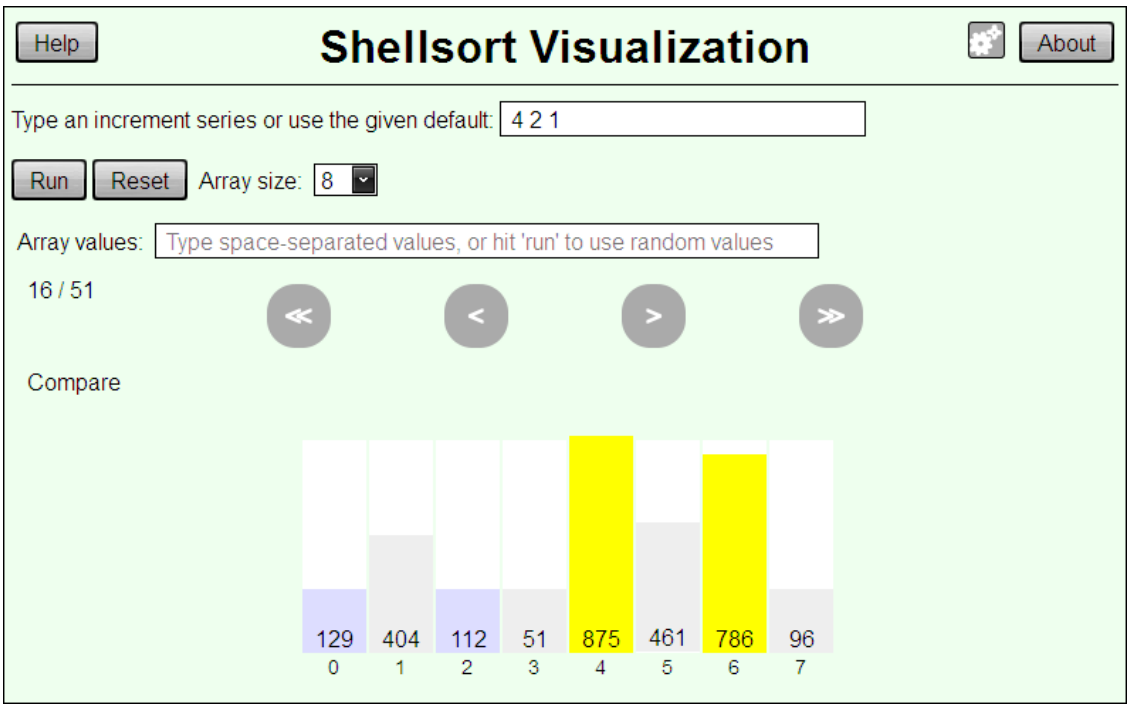


Figure 3.3: An "AV": The Shellsort Algorithm Visualization. Note that most "full AVs" take the form of slideshows

The criteria for awarding proficiency differs between exercises with slideshows and some simple proficiency exercises requiring completion, while other proficiency exercises require the student's score on the exercise to be above a certain threshold. Khan Academy-style exercises require the student to accumulate a certain number of points where a correct answer gains one point, an incorrect answer loses one point, and using a hint allows the student to answer the question without penalty or reward.

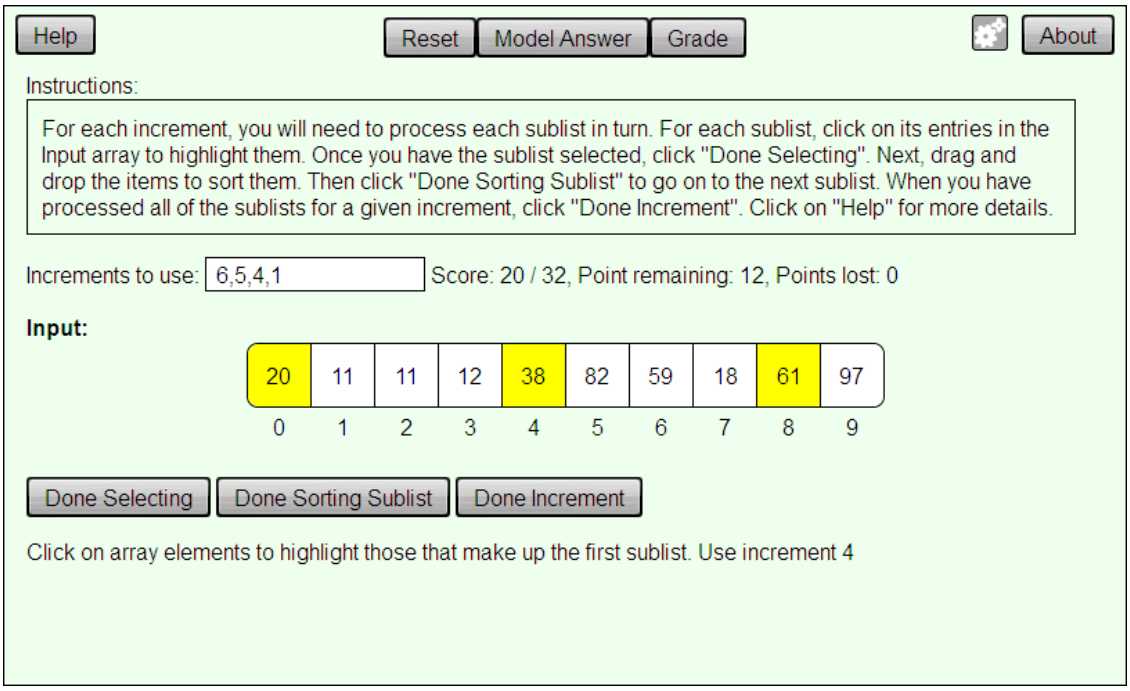


Figure 3.4: A "Proficiency Exercise": The Shellsort Proficiency Exercise

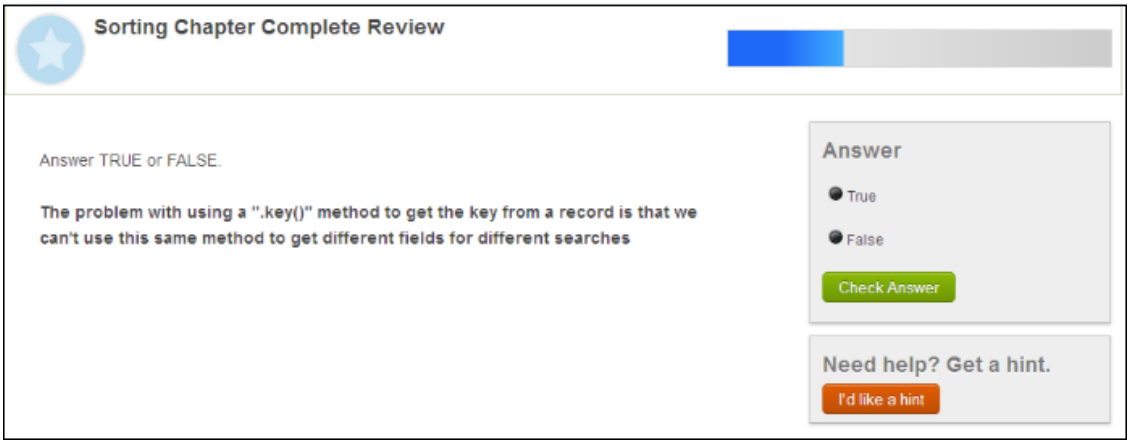


Figure 3.5: A "Khan-Academy-style Exercise": Sorting Chapter Review KA Exercise

3.1.2 Infrastructure

There are three main components to the OpenDSA infrastructure including an OpenDSA web server, a client machine, and a Data Collection Server (DCS). The OpenDSA web server hosts the tutorial content, a web browser running on the student's client machine allows them to view and interact with the content, and the DCS handles all persistent user information including user accounts, active sessions, and storing score and interaction data. While the

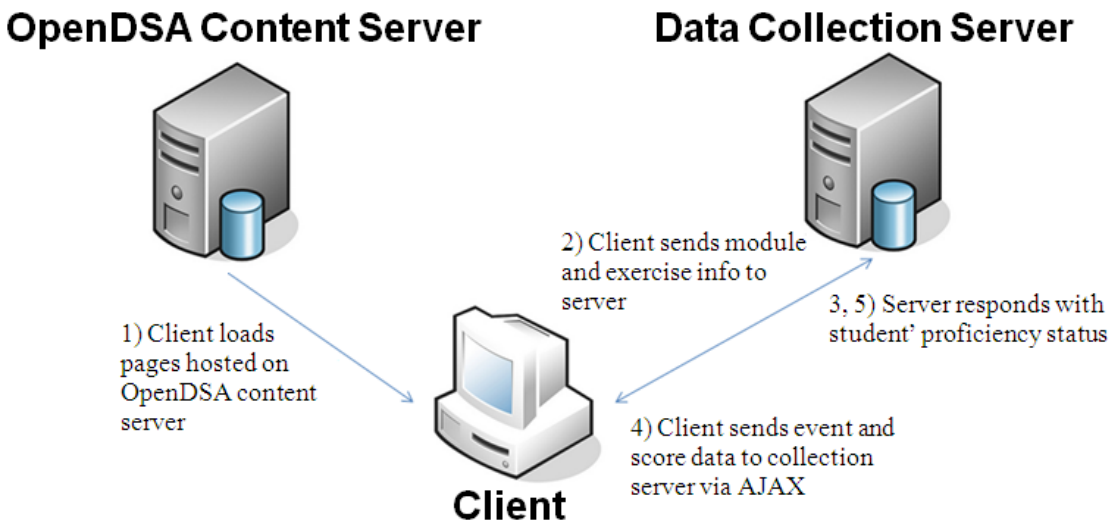


Figure 3.6: OpenDSA Architecture: This figure illustrates the relationship between the OpenDSA server, the client-side framework and the DCS. The OpenDSA client-side framework is responsible for: collecting event data locally and sending it in batches to the DCS, caching and submitting score data from exercises, and checking user proficiency status and displaying appropriate indicators. This last responsibility can be further divided into: obtaining status verification from the DCS for logged in users, determining the proficiency of guest users (allows OpenDSA to remain independent from the collection system), and caching proficiency status to increase responsiveness and support guest users

DCS is required for centralized score and interaction data collection, it is not necessary for OpenDSA to function. This architecture was selected in order to allow OpenDSA to reach the widest possible audience. While it is possible to host the OpenDSA webserver and the DCS on the same machine, they were intentionally developed as separate systems (and at our site they are hosted on separate VMs) so as to allow instructors to provide OpenDSA content to their class without necessitating the extra effort required to set up the DCS if the instructor was not interested in those capabilities. Another option which would require such flexibility is if an instructor wanted to set up their own content server and make arrangements with another university to use an existing DCS. A high-level view of this architecture can be seen in Figure 3.6.

The DCS was written by Eric Fouh, a PhD student from Virginia Tech, using the Python-based Django framework. It was designed as a plugin to Aalto+, a course management framework developed at Aalto University, in order to take advantage of existing capabilities, rather than creating our own system from scratch. The DCS provides a web service for OpenDSA applications that creates and manages user accounts, receives and stores score and event data, and verifies users' proficiency.

3.2 Client-side Framework

The development of the client-side framework is my major contribution to the OpenDSA infrastructure. This framework is written in JavaScript and is separated into 3 primary files. The first, `odsaMOD.js`, contains code which is specific and common to all modules while the second, `odsaAV.js`, contains code specific and common to AVs and proficiency exercises. The third, `odsaUtils.js`, contains several utility functions as well as the interaction data logging functions used by the first two files. Overall, the responsibilities of the client-side framework include:

- Providing an interface for users to login, logout, and register new accounts
- Sending the information necessary to store a new exercise in the database
- Managing a user's intermediate score during an exercise (Automatically buffering and sending score data to the DCS when a user completes an exercise)
- Managing a user's proficiency
  - Determining whether a user obtains proficiency with a module or exercise
  - Caching the user's proficiency status locally
  - Displaying the appropriate proficiency indicators
  - Making sure the local proficiency cache remains in sync with the server
- Keeping multiple OpenDSA pages in sync: Ensure that actions such as logging in, logging out, or gaining proficiency are reflected across all OpenDSA pages open within the browser
- Collecting and transmitting user interaction data to the DCS

All but the last of these responsibilities are handled by `odsaMOD.js`, effectively making module pages the "brains" of the client-side framework. In order to make OpenDSA function independently from the DCS, the client-side framework had to contain all the configuration information necessary to determine when a student obtains proficiency with each exercise and how many points they earn. This configuration information is specific to an instance of an OpenDSA textbook and is inserted during compile time by a custom configuration process which I developed in parallel with the client-side framework. While the reStructuredText modules already had to be compiled by Sphinx, exercises are written in HTML, CSS and JavaScript and do not require any special development tools. In order to keep the barrier of entry for AV development low, the decision was made to only include the configuration information on module pages. The portion of the framework that runs on exercise pages automatically handles sending the score from the embedded exercise to the parent module

page using HTML postMessage. This is believed to be an appropriate solution because exercises do not intrinsically have points or a proficiency threshold, rather these are properties of the context in which the exercise is viewed, i.e. they are book dependent. Therefore it is logical to have exercise scores evaluated and sent to the DCS by the parent module.

3.2.1 Login, Logout and Registration

Each module page contains a "Register" and a "Login" link, both of which cause a form to appear. Filling out the form and clicking the submit button causes an AJAX (Asynchronous JavaScript and XML) message to be sent to the DCS. If registration is successful, a new user account is created on the DCS and the student is automatically logged in. The login operation creates a new session with the DCS which returns a session key that the framework stores and sends with all future messages to identify the student. When the user logs in, the "Register" and "Login" links are replaced with a "Logout" link and the student's username which shows who is logged in and serves as a link to the Student Gradebook page. While the logout operation also triggers an AJAX message to the DCS, upon successful logout the framework refreshes the page in order to reset the state of all exercises.

Errors returned by the DCS are handled gracefully with messages being displayed to indicate things like an incorrect username / password combination, the inability to contact the DCS, or an invalid session. Since the DCS can only handle one session per user at a time, if the student logs in anywhere else, his or her first session is invalidated. When any communication to the DCS is sent using the old, invalid session key, the server returns an HTTP 401 error which causes the framework to delete the invalid session information, inform the student that his or her session is no longer valid and that they must log in again, then refreshes the page after the student closes the message. This causes the login box to pop up.

If the client-side framework is not configured to use a DCS, it removes the registration and login links since they serve no purpose given this configuration and would serve only to confuse students. Whether or not a DCS is configured, when a user is anonymous (not logged in), his or her actions are attributed to a local guest account.

3.2.2 Dynamically Loading Exercises

Besides independence from the DCS, another advantage to having all the configuration information for modules and exercises available on the client is that it provides an easy way to load new modules and exercises into the DCS database. When a module page loads, a function called `loadModule()` is called which handles several conditions. If a student is logged in, an AJAX message is sent to the DCS with the configuration information for the module and all exercises it contains. The DCS updates the database as necessary and responds with information about the student's proficiency with the module, each exercise in

the module, and progress information for the KA exercises. The local proficiency cache is updated based on the information in the response, which keeps the client in sync with the server. If no user is logged in when the module loads, then the guest account information stored in the local proficiency cache is used to initialize the proficiency indicators on the module page.

3.2.3 Score Management

The module portion of the framework implements two listeners relevant to score management. One listens for events generated by the JSAV-based mini-slideshows included on most module pages, while the second listens for HTML5 postMessages from embedded AVs or Khan Academy exercises. Both listeners send the data they receive to `processEventData(data)` which processes the events to ensure all additional non-standardized event properties are logged properly and calls `storeExerciseScore(exercise, score, totalTime)` under three circumstances: (i) if the event type is `odsa-award-credit` (ii) if the user reached the end of a slideshow (iii) if the event type is `jsav-exercise-grade-change` and the final step in the exercise was just completed . If a student is logged in or the system is configured to assign anonymous score data to the next student who logs in, then `storeExerciseScore()` will create a score object and store it in the browser's local storage. If the score is above the proficiency threshold and either no DCS is configured or no student is logged in, then the guest account is awarded proficiency and the appropriate proficiency indicator is displayed.

Near the end of `processEventData()`, `flushStoredData()` is called which in turn calls `sendExerciseScores()` and `sendEventData()`. `sendExerciseScores()` creates a copy of the current student's list of score objects related to the current book and deletes the original list. It then loops through the copied list, calling `sendExerciseScore()` for each score object. `sendExerciseScore()` sends the specified score object to the DCS and updates the student's proficiency status for the exercise based on the server's response. If an error occurs, the score object is added back to the buffer in local storage to minimize data loss. Ideally, the score data could be read from local storage, sent to the DCS, and only removed from local storage upon successful transmission. However, since reading from and writing to local storage is not an atomic operation, if the score data was deleted after receiving the server's response any data written to local storage since it was read would be lost. By using a copy of the list, the original can be deleted immediately, allowing a new list to be written to as safely as possible.

3.2.4 Proficiency Management

The module portion of the framework is also in charge of determining a user's proficiency with an exercise or module, caching this proficiency status in local storage, displaying the appropriate proficiency indicator for each exercise, and making sure the local proficiency

cache stays in sync with the server. For each book, for each user, the client stores the status of each exercise with which the user obtains proficiency. The status can be one of several states:

- `SUBMITTED` - indicates the student has obtained local proficiency and his or her score has been sent to the server
- `STORED` - indicates the student has obtained local proficiency and the server has successfully stored it
- `ERROR` - indicates the student has obtained local proficiency and the score was sent to the server, but it was not stored successfully

If an exercise does not appear in a student's proficiency cache, that student has not obtained proficiency.

Local Proficiency Cache

The primary purpose of the local proficiency cache is to allow users who are not logged in to maintain their progress and to allow OpenDSA to function without a DCS, but a secondary benefit is that it makes pages more responsive for students who are logged in. While proficiency information is synced on every page load for these users, keeping a local copy allows the page to immediately display the proper proficiency indicators rather than waiting for a response from the server.

The proficiency cache uses the browser's local storage to store an object for the guest user and each user who logs in. Each user object contains one or more book objects, and each book object contains exercise names mapped to the user's exercise status.

Proficiency Displays

Proficiency for mini-slideshows is indicated by the appearance of a green checkmark on the right side of the slideshow container. If the status is `SUBMITTED`, a "Saving..." message will appear beneath the checkmark that will be removed once the status changes to `STORED`. If the status is set to `ERROR`, a warning indicator will appear (to draw the user's attention to the exercise) and the saving message will be replaced by an error message and a "Resubmit" link which allows the student to resend his or her score data without re-completing the exercise.

Proficiency for embedded exercises is indicated by the color of the button used to show or hide the exercise. Red indicates that the student is not proficient, yellow indicates that the student's score has been submitted or an error occurred, and green indicates that the student's proficiency has been verified by the DCS.

When a student obtains proficiency for all the required exercises in a module, the words "Module Complete" appear in green at the top of the module. If "Module Complete" appears in yellow, the student has obtained local proficiency with all the required exercises but one or more of them have not yet been successfully verified by the server. In general, to obtain module completion a student must complete all exercises marked as "required" in the configuration file. If the module does not contain any required exercises, module completion cannot be obtained unless the instructor specifies a configuration file option, explicitly allowing proficiency to be granted.

On the table of contents page, a small green checkmark appears next to modules a student has completed. On the Student Gradebook page, the score for exercises and modules with which the student is proficient are highlighted in green. At this time, there is no concept of chapter completion.

Syncing with the Server

As described in Section 3.2.2, on page load, if a student is logged in then `loadModule()` sends a message to the DCS and the response contains information about the student's proficiency with the module and each exercise in the module.

The Contents and Gradebook pages call `syncProficiency()`, which initiates an AJAX request to the DCS, which in turn responds with the proficiency for all modules and exercises.

In both cases, the information returned by the server is used to update the local proficiency cache.

Determining Proficiency Status

A user's proficiency status is determined differently in different situations. If no DCS is configured or no user is logged in (meaning the user is the guest account), the client is given the authority to determine whether or not the user is proficient with an exercise or module. Exercise proficiency is awarded if the user's score on an exercise is greater than or equal to the proficiency threshold for that exercise. Module proficiency is awarded when a user has obtained proficiency with all exercises in a module that are listed as `required` in the configuration file. Since there is no server involved in the process, the only valid status for the guest user is `STORED`.

A DCS is required to verify proficiency of all logged-in users and two additional statuses are added to handle interaction with the server. When a logged-in user's exercise score is sent to the server, if the client determines that he or she is proficient, then his or her status for the given exercise is set to `SUBMITTED`. When the server responds to the AJAX request, the response contains a boolean indicating whether or not the user is proficient with the given exercise. If the server determines that the user is proficient, his or her status for the exercise

is set to `STORED`. However, if the server responds with "`success`": `false` or an HTTP error occurs, the status is set to `ERROR`.

When the status of a required exercise is set to `STORED`, `checkProficiency()` is called to check for module proficiency. `checkProficiency()` begins by calling `updateProfDisplay()`, which updates the proficiency displays for the given exercise or module based on the contents of the local proficiency cache and returns the status. If the status is `STORED`, then `checkProficiency()` returns immediately. If the status is not `STORED` but a user is logged in, then the framework will send an AJAX request to the DCS asking if the user is proficient with the exercise or module and update the proficiency cache appropriately when it receives a response. If the status is not `STORED`, no user is logged in and the request is for module proficiency, `checkProficiency()` will loop through an internal list of exercises and determine if the guest user has proficiency with all required exercises. If so, the guest account is awarded module proficiency and the cache is updated. If a single required exercise is found that the guest user is not proficient with, the loop breaks and the function returns.

A user's proficiency status can also be updated by the synchronization functions described in Section 3.2.2.

3.2.5 Keeping Pages in Sync

Consider the situation where a user logs in to OpenDSA and then opens modules in multiple tabs. Since a user is logged in, each tab will display the logged in user's name in the top right hand corner. Later, the user logs out and another user logs in on one of the pages. Without a system to sync pages, it would appear as if two users are logged in at the same time, which could potentially be very confusing. To rectify this situation, the module portion of the framework implements an `updateLogin()` function which is called any time the window receives focus. This function determines whether the current user appears to be logged in and updates the display if this is not the case. If another user has logged in since the page was loaded, the former user's name is replaced with the current user's name and if no user is logged in, the logout link and former user's name are replaced with the default "Register" and "Login" links. If any change is made, `loadModule()` is called to ensure the proficiency displays match the current user's progress. Since the function is called when the window receives focus, updates will be made as soon as the user navigates to a tab.

3.2.6 Interaction Data Collection and Transmission

The client-side framework collects data about how users interact with OpenDSA for two reasons:

1. To continually improve OpenDSA

2. For pedagogical research purposes

As a user interacts with OpenDSA, a variety of events are generated. If a DCS is configured, then information is recorded about these events, buffered in local storage, and sent to the server when a flush is triggered. If a user is logged in, then the event data is sent with his or her session key, effectively tying interaction data to a specific user. If no user is logged-in the data is sent anonymously. This ensures that we are able to collect as much interaction data as possible.

Unfortunately, this event data is stored and transmitted in much the same way as the score data discussed in Section 3.2.3, which means it suffers from the same lack of atomic operations on local storage. Event data is generated much more frequently than score data, increasing the likelihood that new data will be written to local storage between the time when event data is read and the empty event list is written back to local storage, resulting in lost data. It is believed that this limitation is partially responsible for the missing event data we observed and discuss in Section 7.1.1. We are currently investigating alternative storage techniques and a local storage locking mechanism which we hope will address this problem.

3.3 Data Model

The client-side framework is responsible for collecting and sending two types of data to the DCS including: score and event data. The following sections describe what information is recorded by each data type.

3.3.1 Score Data

Score data is buffered in local storage as a list of score objects related to a specific user and instance of OpenDSA. Each score object contains:

- `exercise` - the name of the exercise the student completed
- `module` - the module with which the event is associated
- `score` - a float between 0 and 1.0 that indicates the student's score on the exercise
- `steps_fixed` - an integer counting the number of steps which were automatically corrected by the exercise
- `submit_time` - a timestamp indicating when the exercise was completed

- `total_time` - the time, in milliseconds, the user took to complete the exercise
- `uiid` - the unique instance identifier which allows the score to be tied to a specific instance of an exercise or a specific load of a module page

When score data is transmitted to the DCS, additional `key` and `book` properties are added which contain the student's session key and the OpenDSA instance identifer, respectively.

3.3.2 Interaction / Event Data

User interaction data is buffered as a list of event objects associated with a book. Each event object contains:

- `av` - the name of the exercise with which the event is associated (the null string is used for module-level events)
- `desc` - a human readable description or stringified JSON object containing additional event-specific information
- `module` - the module with which the event is associated
- `tstamp` - a timestamp indicating when the event occurred
- `type` - the type of event
- `uiid` - the unique instance identifier which allows an event to be tied to a specific instance of an exercise or a specific load of a module page

When event data is transmitted, a new object is created that contains the `key` and `book` properties described above and a third property, `actions`, which contains the entire list of event objects.

Chapter 4

Experiment

Building on our previous experiment in Fall 2012, we expanded our Spring 2013 study from one class to three. These three classes included both offerings of Virginia Tech's CS3114 Data Structures and Algorithms and CS223 Data Structures II, offered by Alexandria University. To distinguish them, we will refer to the two CS3114 classes as CS3114A and CS3114B for the remainder of this paper.

4.1 CS223

OpenDSA was used as the primary resource for hashing instruction in CS223. Dr. Hicham Elmongui, the instructor of CS223, used OpenDSA in class instead of traditional PowerPoint slides and assigned OpenDSA modules for students to complete for a homework grade. This study lasted roughly one week, from March 4th to March 10th. A midterm exam was administered on March 31st.

4.2 CS3114A

The class designated as CS3114A used OpenDSA as the primary resource to cover both sorting and hashing. During class the instructor provided a traditional lecture with PowerPoint slides but replaced the example slides with interactive examples from OpenDSA. Students were required to read and complete the Sorting and Hashing chapters of OpenDSA outside of class as a homework assignment worth 5% of the class grade. This study lasted approximately three weeks, from March 19th to April 12th with a midterm exam given on April 11th. Students in this class were not required to and did not receive credit for viewing slideshows, however, they still received feedback from the system, in the form of a green checkmark, when they completed a slideshow.

4.3 CS3114B

Like CS3114A, CS3114B used OpenDSA to cover sorting and hashing, however, OpenDSA was not used as the primary resource. The instructor taught the sorting section using his standard lectures, administered an exam, and then assigned homework from OpenDSA. For the hashing section, the instructor continued to use his standard lectures, but assigned homework from OpenDSA prior to the exam. This study lasted five weeks with students completing the sorting material between March 8th to March 31st, while having until April 12th to complete the hashing material. Students in this class were both required to complete and received credit for completing slideshows. These students also received a green checkmark as feedback from completing slideshows.

Chapter 5

Research Questions and Analytic Approach

As researchers and developers we continually strive to improve OpenDSA, but in order to best meet the needs of students, we must first understand how students interact with the current implementation of OpenDSA. Once we have this knowledge, we can begin to make improvements to the system. Until now, our research topic has been broadly defined as characterizing student use of OpenDSA. Here we refine our overall topic by identifying the specific subtopics and related questions we seek to answer in this study. Each section below explains a specific research focus and describes our analytic approach.

5.1 Time Distribution of Activity

For our first subtopic, we propose the existence of student clusters which are distinguished by the time when members begin to seriously complete exercises. In particular, we expected to observe three separate groups. Members of the first group, which we call the "proactive" group, would begin completing exercises near the beginning of the given time range. Initial activity by members in the second group, which we refer to as the "normal" group, would occur near the middle of the time range. Finally, initial activity by members of the third group, which we refer to as the "procrastinator" group, would occur near the end of the given time range.

To determine whether such clusters existed, we examined the timestamps related to when each student obtained proficiency with each exercise. The first time when proficiency was awarded to a student in the class became the base time for that class. Proficient attempts were assigned to bins based on their calculated offset from the base time. For instance, using a bin size of 12 hours, an attempt where proficiency was awarded 15.3 hours after the base time would be assigned to the second bin, which contains times between 12 and 24 hours.

Separation by class was performed because each used OpenDSA within a different time range and under different conditions. We believed viewing each class's data individually would reduce the effect of those differences and better reveal trends that might be obscured by viewing the data as a whole.

5.2 Time Required for Proficiency

With traditional courses there can be a disconnect between the amount of time an instructor expects an assignment to take and the time actually required for the majority of students to complete it. Our next subtopic is an analysis of how long it took students to obtain proficiency on exercises. Given this information, instructors will gain a better understanding of the real-world time requirements for OpenDSA's sorting and hashing exercises, allowing them to better gauge the workload they assign. Students will benefit not only from instructor's more realistic expectations, but also from better time management facilitated by more accurate estimates of time required to complete assignments. Developers could also use this data to discover what exercises students struggle with in order to improve them.

The first step of data processing was to determine each student's total time to proficiency with each exercise. This was calculated by summing the time required for each exercise instance that the student completed before becoming proficient with that particular exercise. These times were categorized by exercise type, and once all of a student's times had been calculated, the median of each category was found and recorded. The times were also categorized by exercise, and the median across all students was found and recorded.

By analyzing the data with regards to exercise, we obtained an estimated length of time required to complete each exercise, which can be applied in the ways described above. When performing this analysis we separated the students by class to account for class-specific conditions such as the instructor of CS3114B covering sorting and administering the exam on it prior to students using OpenDSA. Analysis was performed on each exercise type independently since the types are disparate, and a combined analysis could not be expected to deliver meaningful results.

5.3 Credit Seeking vs Learning Behavior

In this section we investigate several behaviors which we collectively term as "credit-seeking behavior". We believe this is different from (and possibly in conflict with) "learning behavior." The specific behaviors we were interested in include jumping straight to the exercises without reading the text, clicking through slideshows as quickly as possible, skipping to the end of slideshows for credit, and using AVs to complete exercises.

5.3.1 Not Reading

We propose there are two general patterns of behavior related to reading the text. In the first, students would read through most or all of the text, completing exercises as they encounter them within the module. In the second, students would not read a majority of the text and instead skip directly to the exercises. Since the mini-slideshows are considered to be content, and CS3114A students were not required to complete slideshows, we chose to consider only proficiency and Khan Academy-style exercises for this analysis. Both of these exercise types are hidden by default, requiring students to click a button to reveal the exercise. To determine whether separate behaviors existed, we calculated the total time a module was open before a button was clicked to reveal an exercise. When determining the total time, we accounted for students opening and closing modules multiple times before completing exercises. This calculation was performed for every module and every student, accounting for cases where the module was closed and reopened later. For simplicity, we assumed that if a student revealed an exercise, they attempted it immediately. These calculated times were grouped into bins where each bin represents a range of times. We experimented with different size bins in order to determine an optimal trade-off between data resolution and clustering.

If the two types of behavior existed, we would expect a histogram to contain two clusters, one near the lower end of the spectrum representing modules on which students skipped directly to the exercises, and another cluster higher on the spectrum representing modules on which students took more time, presumably due to reading the text. If these clusters were detected, we could create a threshold to separate the two clusters. Using this threshold, we could count the number of modules for which each student skipped directly to the exercises.

5.3.2 Clicking Through Slideshows

We also propose two patterns of behavior related to completing slideshows. In the first, which we refer to as "learning" behavior, students would use the slideshows as a learning resource by reading slide descriptions and navigating backward and forward as necessary to re-examine operations. In the second, which we refer to as "rushing" behavior, students would click through the slides as quickly as possible in order to obtain credit, without paying attention to the material. To determine whether students rushed through slideshows in order to receive credit, we examined the mean time per slide on slideshow instances where students first obtained proficiency. As with our analysis of reading behavior described in Section 5.3.1, we placed the mean slide times in bins which represent time ranges. Again, we expect two clusters on be visible on a histogram, allowing us to select a threshold which could be applied to identify the number of slideshows each student has rushed through. While it is not addressed in this thesis, we encourage future research to investigate whether students who rush through slideshows return later and exhibit learning behavior after obtaining proficiency.

5.3.3 Skipping to the End of Slideshows

The grading system in place during the Fall 2012 experiment contained a bug which would allow students to obtain credit for slideshows without viewing all the slides. They could click the button that navigates to the end of the slideshow, then perform some variation of advancing a single step to get credit. Based on the data collected from that experiment, we were able to determine that some students discovered and took advantage of this fault. While we now have a solution to this bug, we intentionally left the existing mechanism in place during our experiment in order to quantify student discovery and use of this bug. Since the most common method for exploiting this bug was to jump to the end of the slideshow then click forward, we refer to this behavior as "skipping to the end of a slideshow" or simply "skipping".

To determine the number of slideshows each student skipped, all slideshow events were examined. The first time an event was encountered that belonged to the instance where a student first obtained proficiency with the slideshow, we performed a look-ahead operation to ensure events existed for each step in the slideshow. If so, the student viewed each step and it was considered a valid attempt. If not, a counter tracking the number of times the student skipped was incremented. Note that for this particular question, we were not concerned with the order slides were viewed, only whether or not each slide was viewed at least once. Again, we encourage future research to explore whether students return to exhibit learning behavior after obtaining proficiency.

5.3.4 Using AVs to Complete Exercises

For the majority of the JSAV-based proficiency exercises, students are expected to perform actions that closely resemble those demonstrated by a related AV. We know from the Fall 2012 experiment that some students ran the AVs using the exercise input and mimicked the AV output in order to complete the exercises. Therefore the third aspect of credit-seeking behavior that we chose to investigate involves quantifying how much students used the AVs for assistance when completing proficiency exercises.

Whenever a proficiency exercise is generated or an AV is run, the initial state is logged. Therefore in order to determine if a student used AV assistance we matched the initial state of a proficiency exercise with the initial state of its related AV (if one exists), that is timestamped before the student completes the proficiency exercise. Criteria for completing a proficiency exercise include completing every step of the exercise, clicking the "Reset" button, or refreshing the page. At the time of the experiment, only three proficiency exercises had matching AVs which a student could use for assistance, so our analysis was limited to "mergesortProficiency", "ShellsortProficiency" and "quicksortProficiency". Since CS223 students only used the hashing chapter, they do not appear in our analysis.

5.4 Completion of Non-Required Exercises

Our final subtopic investigates the effect of exercise requirement on student behavior, namely whether students view exercises as a useful learning resource or simply as a requirement to be completed. In order to address this question, students in CS3114A were not required to complete and did not receive credit for slideshows, whereas slideshows were required for credit in CS3114B.

We determined the total number of slideshows started and the total number completed for each student in both CS3114A and CS3114B, then computed usage statistics. Since CS3114A students did not receive credit for completing slideshows, we believe if they chose to use slideshows at all they most likely did so for the learning benefit. Therefore we expected to see little to no credit-seeking behavior on slideshows from this group. However, we do note that students still receive a green checkmark for completing a slideshow, and this might have affected their behavior.

Chapter 6

Results and Discussion

This chapter presents results relevant to each of the research questions discussed in Chapter 5. Individual student results are identified only by an arbitrary number in order to protect student privacy, and median values are used both to effectively compress the data and further protect student identity. While the identification numbers are arbitrary, they are consistent (with regards to each class) throughout this document, allowing patterns across multiple research topics to be assessed.

Unfortunately due to some technical problems which are explained at length in Section 7.1, some event data was not recorded and some raw score data times were inaccurate. The inaccurate score times were reconstructed to the best of our ability based on other data we collected and used in our analysis. In several cases it was necessary to eliminate certain outlying data caused by missing event data and in each case is clearly noted. Where possible, we indicate how missing event data could affect the results presented. Another prominent source of outliers were the abandoned user accounts discussed in Section 7.1.4. All data related to these accounts was eliminated from our analysis and therefore will not be individually noted when discussing each research topic. After eliminating these accounts, CS223 contained 21 students, CS3114A contained 51, and CS3114B contained 65.

Module analysis includes only those modules which appear in the sorting and hashing chapters as they are the only ones students may complete for credit and any others would only be a distraction. Additionally, `hashAV` does not appear in our analysis of exercises because it does not possess proficiency criteria and cannot be "completed."

6.1 Time Distribution of Activity

In order to determine whether clusters of students existed, we grouped exercise completion times into bins and examined their distribution. We found the bin sizes had to be adjusted

in order to provide an optimal trade-off between resolution and clustering. If the bins did not cover a sufficiently large range, we ended up with an excessive number of small bins, and if they covered too large a range, the bins did not provide a suitable amount of detail. In both cases, no obvious clusters could be observed. We suspect the optimal bin size is influenced by both the number of students in a class and the length of time they are given to complete the material, but we do not have sufficient data to determine what relationship may exist.

When examining the CS223 distribution, we found small clusters started to occur with a bin size of one hour (the smallest value we tried), but the clusters did not coalesce into clear groups until a bin size of 14 hours. Since group resolution was lost with bin sizes greater than 14 hours, we selected this value for our bin size as it provides the maximum amount of clustering, while providing the necessary resolution. Clusters begin to appear in the CS3114A data when using a bin size of 20 hours and disappear with sizes larger than 28 hours. While clusters in the CS3114B data begin to appear with a bin size of 14 and disappear after 24 hours, they are most clear with a bin size between 18 and 22 hours. We selected bin sizes of 22 hours for both CS3114 and CS3114B. Each distribution begins with the base time described in Section 5.1, and bins appear as an offset (in hours) from this time.

Contrary to our expectation, the time range for CS223 contained only two groups and neither occurred at the end. The first group appeared at the beginning of the time range where we predicted a proactive group could appear, and the second appeared near the middle of the time range where we predicted a normal group would appear. A third group was observed, but it occurred at 630 hours, well after the OpenDSA material was due. While this group was originally omitted and does not appear in Figure 6.1, further investigation revealed this activity occurred immediately prior to the exam. This provides evidence that students use OpenDSA material to review for exams, a behavior that we hope to investigate further in the future.

CS3114A contained three obvious clusters, as seen in Figure 6.2. The first occurred at the beginning of the time range, ending at an offset of 66 hours. The second cluster appeared between 88 and 264 hours, ending just before the middle of the time range. The third cluster extended from 286 to 572 hours, covering the entire second half of the time range. However, a trough appears at 418 hours, roughly three-quarters of the total time range, which divides this cluster into two slightly less obvious clusters, resulting in four total clusters. The first cluster appears at the very beginning of the time range, meeting the criteria for our speculated proactive group. The second and third clusters occupy the middle of the time range where we expected a normal group to appear. This indicates a possible differentiation within the normal group we proposed, where a more proactive and a less proactive subgroup appear. The fourth cluster occurs at the end and contains a larger amount of activity, meeting the criteria of our procrastinator group. The increased activity level is due to the members of the procrastinator group completing all of their required work within a single cluster, while students from other clusters finish up their remaining work prior to the deadline.

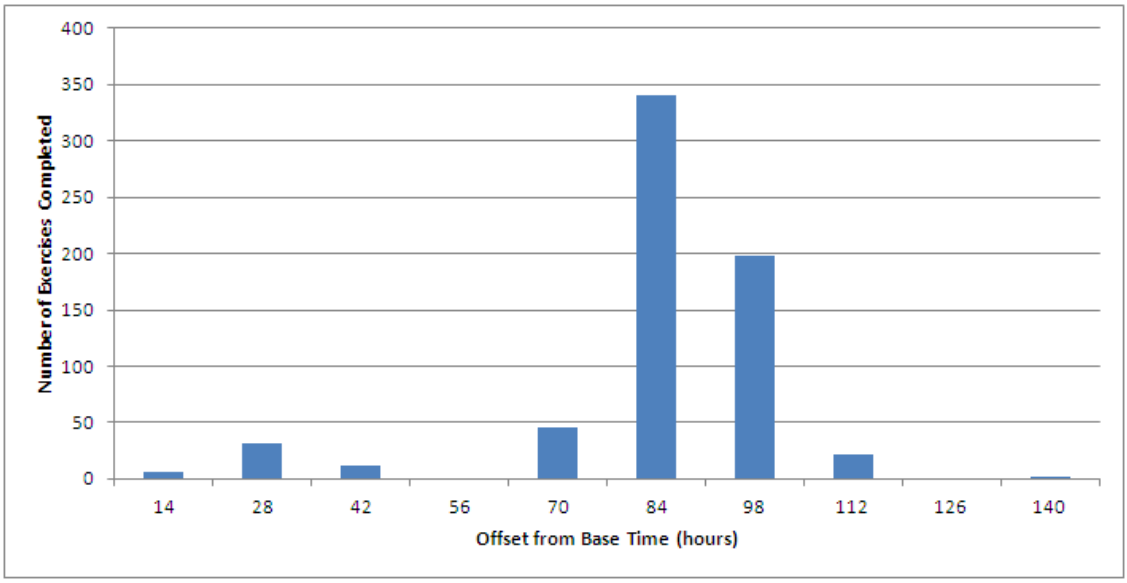


Figure 6.1: Time Distribution of Activity, CS223: The number of exercises completed in 14 hour offsets from the first time a student in the class obtained proficiency

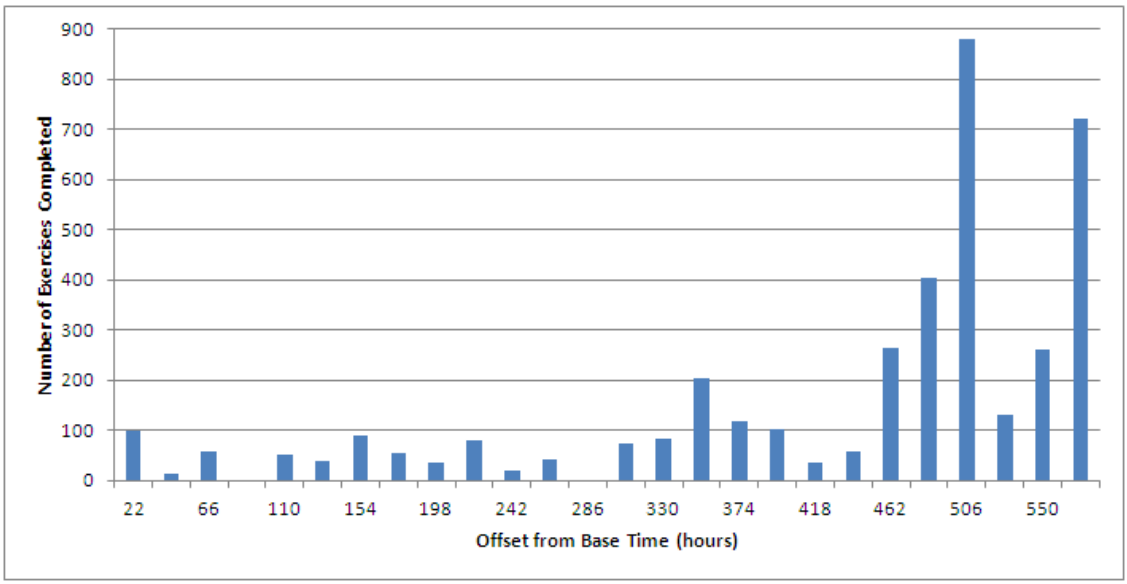


Figure 6.2: Time Distribution of Activity, CS3114A: The number of exercises completed in 22 hour offsets from the first time a student in the class obtained proficiency

The staggered deadline in CS3114B for the sorting and hashing chapters made it more difficult to identify clusters, as seen in Figure 6.3, so the chapters were isolated and analyzed independently. For reference, we calculated the deadline for the sorting material to be 533 hours after the sorting base time and the deadline for the hashing material to be 821 hours after the hashing base time. First consider the sorting chapter, seen in Figure 6.4, where

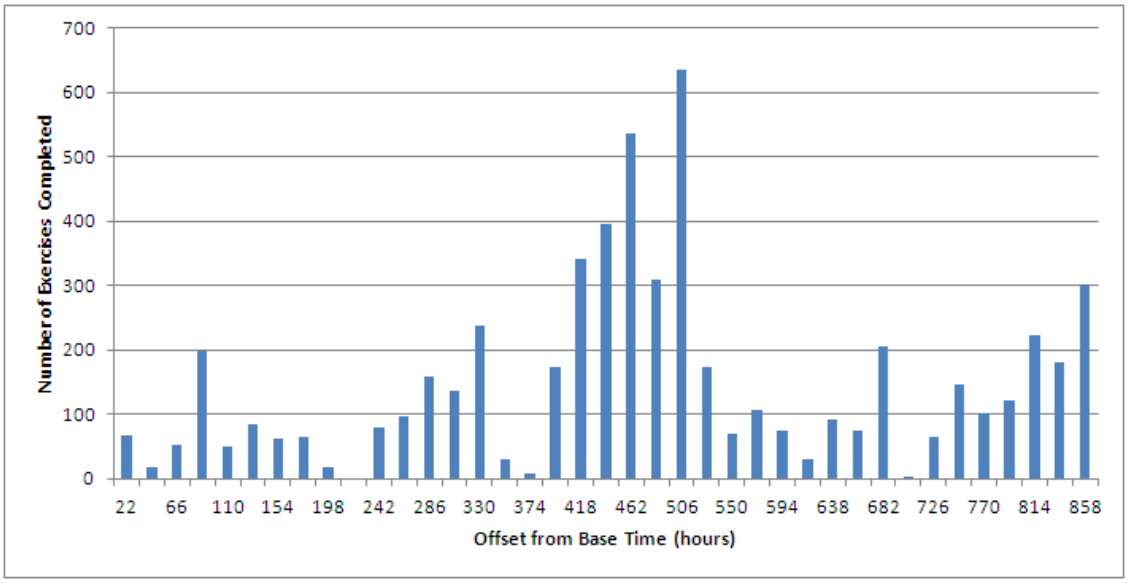


Figure 6.3: Time Distribution of Activity, CS3114B: The number of exercises completed in 22 hour offsets from the first time a student in the class obtained proficiency

three distinct clusters can be observed. The first from 0 to 198 hours, correlates to our proposed proactive group, while the second from 220 to 352 and the the third from 374 to 506, respectively correlate to the proposed normal and procrastinator groups. When considering the results from the hashing chapter, displayed in Figure 6.5, we were able to identify four clusters, two of which were obscured in the combined distribution. The first cluster extends from 0 to 66 hours, the second from 330 to 440, the third from 462 to 638, and the final cluster from 660 to 814.

One drawback of this technique is that it is based on the first proficiency event rather than the explicit start of a time range. While we encourage future work to use an explicit time range, we manually confirmed, based on additional information not provided in the figures, that the "proactive" group completions occurred at the beginning of the time range. Since the results were based on score data rather than event data, we are confident that they are reliable.

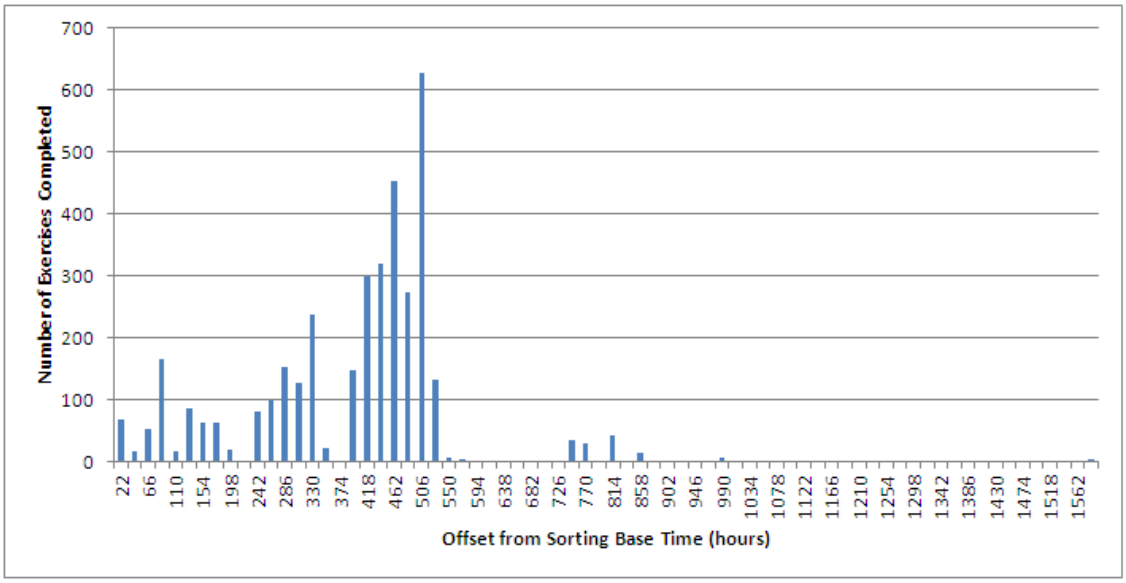


Figure 6.4: Time Distribution of Sorting Chapter Activity, CS3114B: The number of sorting exercises completed in 22 hour offsets from the first time a student in the class obtained proficiency with a sorting exercise

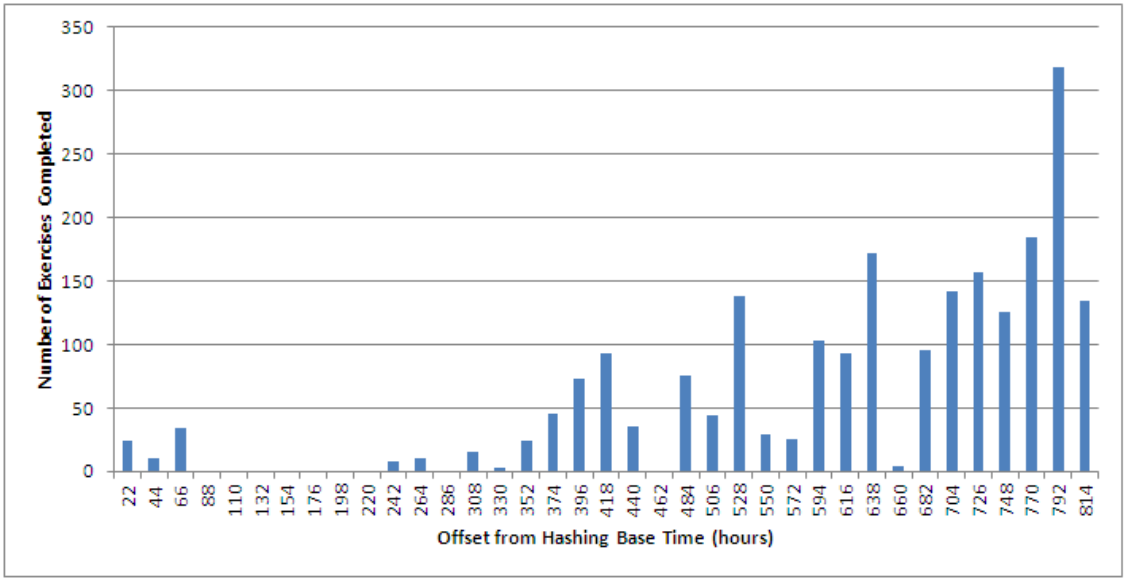


Figure 6.5: Time Distribution of Hashing Chapter Activity, CS3114B: The number of hashing exercises completed in 22 hour offsets from the first time a student in the class obtained proficiency with a hashing exercise

6.2 Time Required for Proficiency

In this section we will examine the amount of time required for proficiency with each of six exercise types. Until now we have only been concerned with slideshows, proficiency exercises and Khan Academy-style exercises. However, each of these types can be divided into two subtypes which are sufficiently different from each other to cause the time requirements to be vastly different. For this reason we analyze each type independently. Slideshows can be broken down into mini-slideshows and full AVs. Mini-slideshows tend to demonstrate a single operation within an algorithm and are usually much shorter than their counterparts, full AVs, which demonstrate the operation of an entire algorithm. Proficiency exercises can either be algorithm simulations where the student manually performs an algorithm's operations or an interactive equation, which we refer to as a calculator, that returns an answer calculated based on user input. Algorithm simulations tend to be long and involved, while calculators usually require little user interaction and tend to be quick. Khan-Academy-style exercises can either be mini-proficiency exercises, which require the student to simulate a single operation from an algorithm, or summary exercises, which ask multiple choice questions about the concepts discussed in the modules.

As seen in Figure 6.6, both CS3114 classes performed similarly on the sorting mini-slideshows. More variation was observed when considering the hashing mini-slideshows. CS3114B students tended to take longer than CS3114A students on the hashing slideshows, though not by a substantial amount. The greatest difference was only 24.5 seconds. The CS223 students, however, took longer than both CS3114 classes on all but one mini-slideshow.

In regards to the AVs, CS3114A students tended to take longer than CS3114B students,

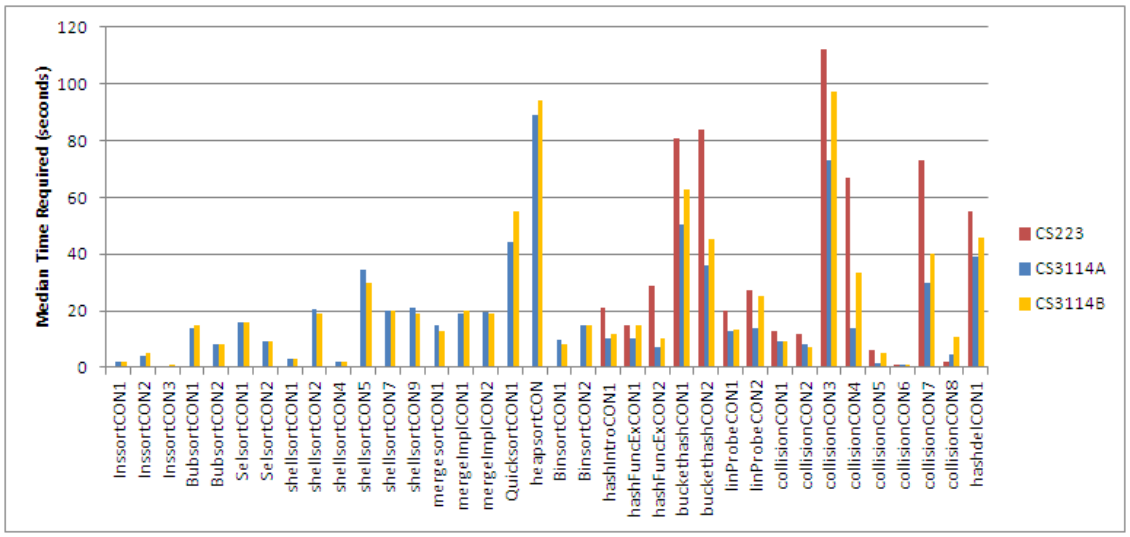


Figure 6.6: Composite Time Required for Mini-Slideshows: Median time required for each class to obtain proficiency with mini-slideshows

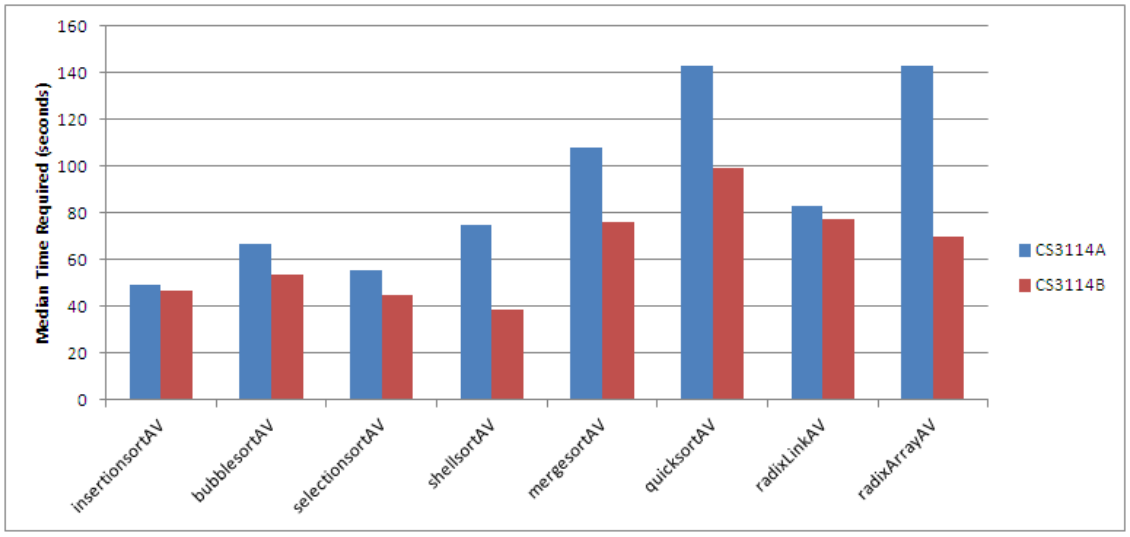


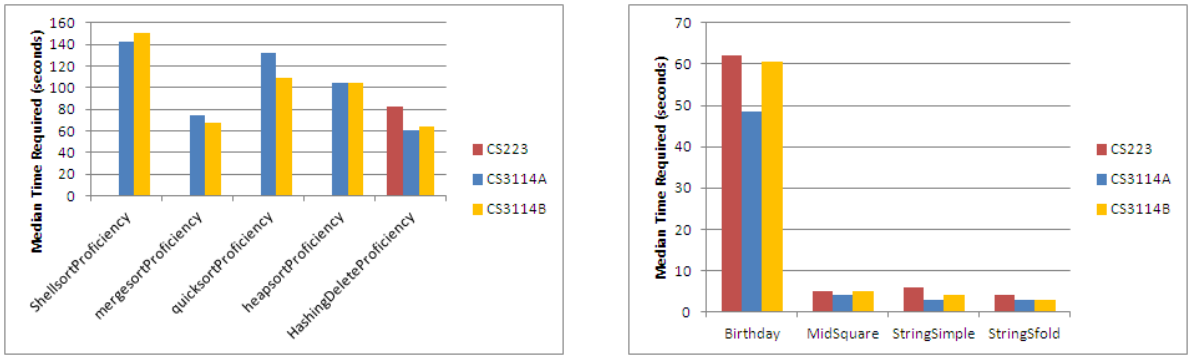
Figure 6.7: Composite Time Required for AVs: Median time required for each class to obtain proficiency with AVs

especially on the Shellsort, Mergesort, Quicksort and array-based Radix sort AVs. Since AVs were not required for CS3114A, this suggests that the students who used them most likely used them as a learning resource rather than simply flipping through them. The results can be seen in Figure 6.7. CS223 does not appear in the figure because they only covered hashing, whereas all the AVs appeared in the sorting chapter.

Both types of proficiency exercises showed very little variation across all three classes, as seen in Figure 6.8. The greatest variation observed on the algorithm simulations was 21 seconds on `HashingDeleteProficiency`, while the largest variation on the calculator exercises was 13.5 seconds on `Birthday`. The substantial difference in time between the `Birthday` exercise and the other calculators is that the `Birthday` exercise requires students to find two values that satisfy different conditions, while the other three simply require students to run them with any input.

With regards to the mini-proficiency exercises, little variation is seen between the CS3114 classes except on `QuicksortPartitPRO` and `HeapsortPRO` where CS3114B students took substantially longer. This suggests the topics covered by those exercises may not have been covered as thoroughly in CS3114B as they were in CS3114A, causing students to take longer. Once again, CS223 took longer than either CS3114 class on all but one exercise, as seen in Figure 6.9. Much like on the mini-proficiency exercises, performance on the Khan Academy-style summary exercises was consistent between both CS3114 classes, while CS223 took substantially longer.

Overall there was little variation between the CS3114 sections, suggesting students are likely to perform equally well in both classes. Additionally, the lack of variation means we can easily compute reliable estimates for the amount of time required for CS3114 students to



(a) Algorithm Simulation Exercises (b) Calculator Exercises

Figure 6.8: Composite Time Required for Proficiency Exercises: Median time required for each class to obtain proficiency with proficiency exercises

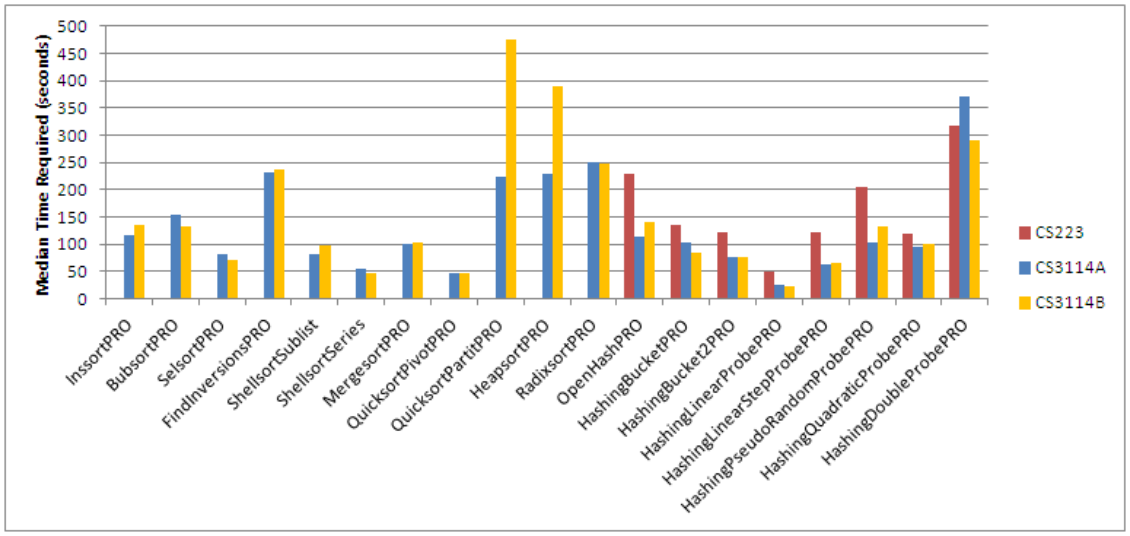


Figure 6.9: Composite Time Required for Mini-Proficiency Exercises: Displays median time required for each class to obtain proficiency with Khan Academy-style mini-proficiency exercises

complete exercises. The CS223 class took consistently longer than the CS3114 classes on nearly all the material. This difference is attributed to the material being presented in English which is not the students' native language.

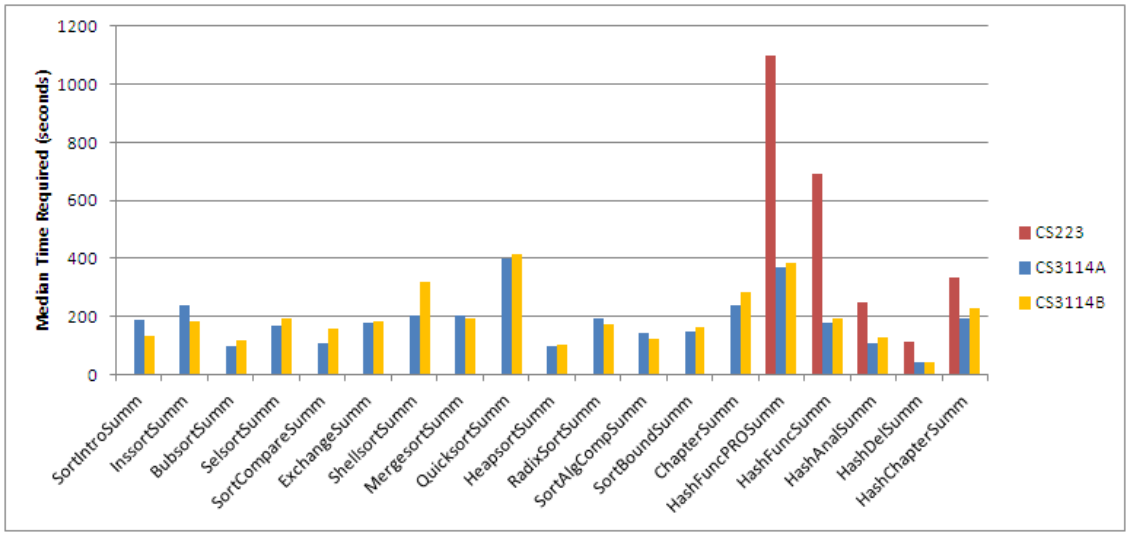


Figure 6.10: Composite Time Required for Summary Exercises: Displays median time required for each class to obtain proficiency with Khan Academy-style summary exercises

6.3 Credit Seeking vs Learning Behavior

6.3.1 Not Reading

As discussed in Section 5.3.1, we expected a histogram of the time between module load and the first exercise attempt to form two clusters. However, contrary to our expectations, the data in all three classes showed a peak at the very beginning of the histogram indicating a large number of students opened modules and immediately started exercises. A sample of each class's respective histogram data, covering a range of five minutes, can be seen in Figures 6.11, 6.12, and 6.13. These figures use a bin size of five seconds in order to produce a high resolution view of the data, allowing any clusters to become apparent. Larger bin sizes were experimented with, but ultimately they proved too coarse to provide meaningful results.

We originally expected a small group of students would not read and therefore be easily identifiable. Since this proved to not be the case, we decided to change the focus of our analysis. Instead of trying to isolate a specific group of students who did not read, we chose to quantify how much the class as a whole read by identifying the quartiles of each class's distribution. The results can be seen in Table 6.1.

As seen in Table 6.1, the first quartile ranges from 10 to 20 seconds and the second quartile ranges from 70 to 120 seconds. This means that students in all three classes begin exercises on 25% of modules within 20 seconds of loading the page and on 50% of modules within two minutes. The third quartile is slightly more reasonable at roughly five to seven minutes. It is important to note, however, that the upper 25% is likely to include modules that were

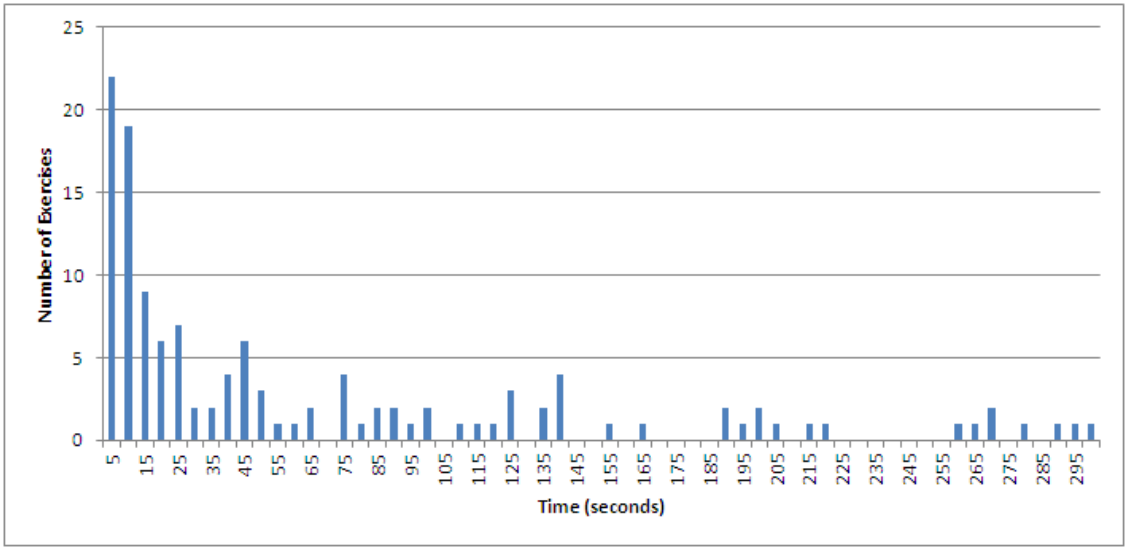


Figure 6.11: Histogram of Load-to-Exercise Time, CS223: Time between module load and the first exercise event, distributed into 5-second bins across a 5-minute sample

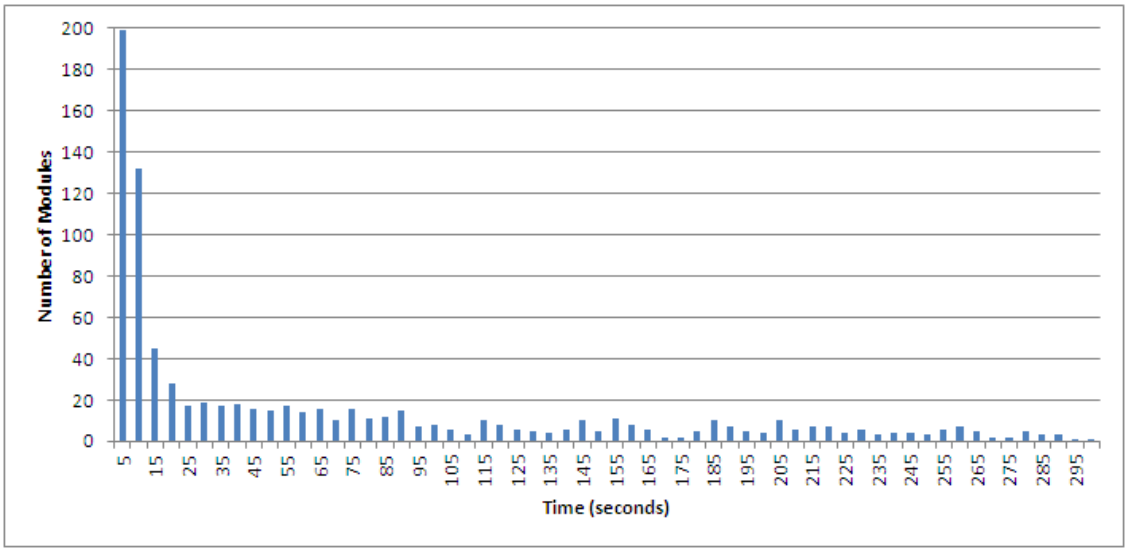


Figure 6.12: Histogram of Load-to-Exercise Time, CS3114A: Time between module load and the first exercise event, distributed into 5-second bins across a 5-minute sample

Class	Quartile 1	Quartile 2	Quartile 3
CS223	15 sec	90 sec	440 sec
CS3114A	10 sec	70 sec	290 sec
CS3114B	20 sec	120 sec	420 sec

Table 6.1: Quartiles Indicating Time Spent Reading: Respectively, each quartile indicates the time when students have started exercises on 25%, 50%, or 75% of modules

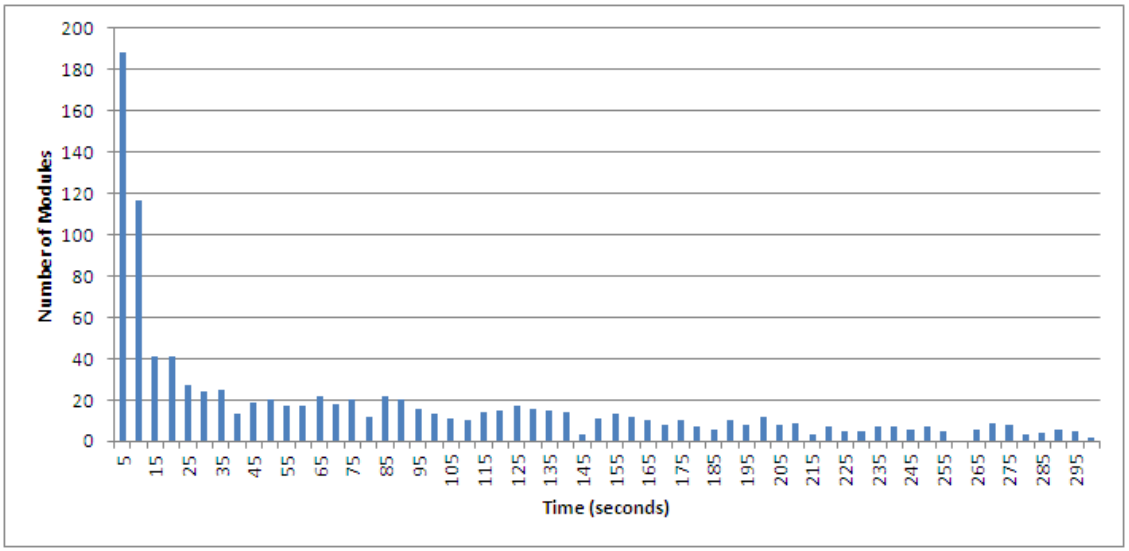


Figure 6.13: Histogram of Load-to-Exercise Time, CS3114B: Time between module load and the first exercise event, distributed into 5-second bins across a 5-minute sample

left open without a user being active, which increases the likelihood of active users' modules being within the lower 75%. While there is naturally a high variability between the speeds at which different people read, and we lack the ability to measure passive interaction, as explained later in Section 7.1.2, we can say with reasonable confidence that students read only about half of the modules, at least on their first view.

Unfortunately, missing event data had an indeterminate effect on these results. Missing exercise data could result in lower measurements, missing module loading data could increase them, and missing module closing data could either increase or decrease the values. While we believe the effect to be negligible since our results are based on median times, future research could use focus and blur events to produce more accurate estimates of how long each module was actually viewed rather than how long it was open.

6.3.2 Clicking Through Slideshows

In order to determine whether students rushed through material, we examined the distribution of mean-time-per-slide for all completed slideshows. As in Section 6.3.1, we expected two clusters which we could use to determine a threshold for identifying rushing behavior. Also similar to Section 6.3.1, our data revealed a peak at the beginning of the distribution (using a bin size of one second), as seen in Figures 6.14, 6.15, and 6.16. We therefore decided to examine the class behavior as a whole through the use of quartiles, the results of which can be seen in Table 6.2.

The first quartile for all three classes was one second, indicating 25% of slideshows were

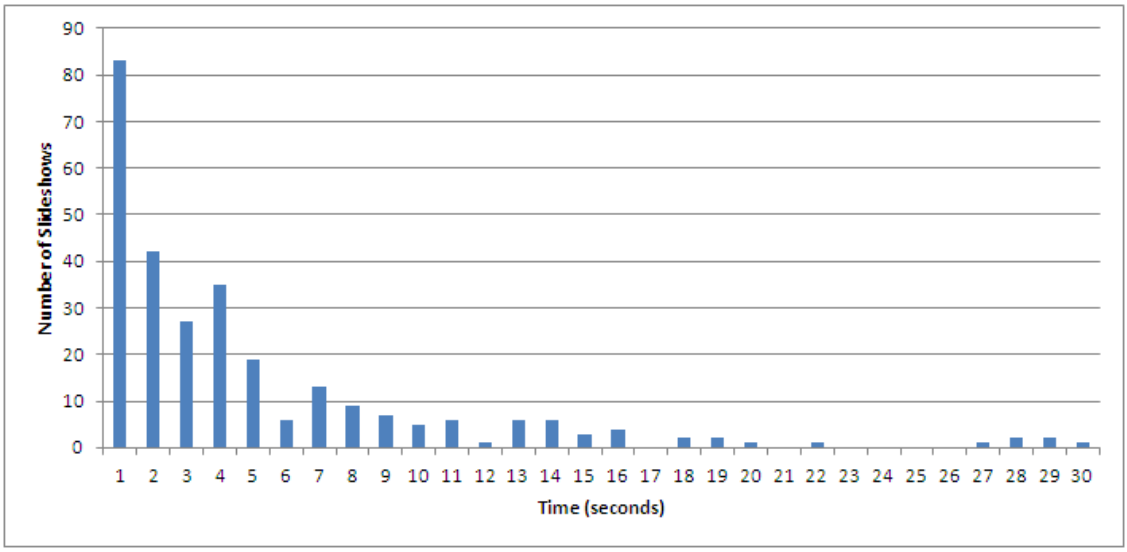


Figure 6.14: Mean Time Per Slide, CS223: A histogram showing the number of slideshows completed with a mean-time-per-slide in each time range (1 second resolution)

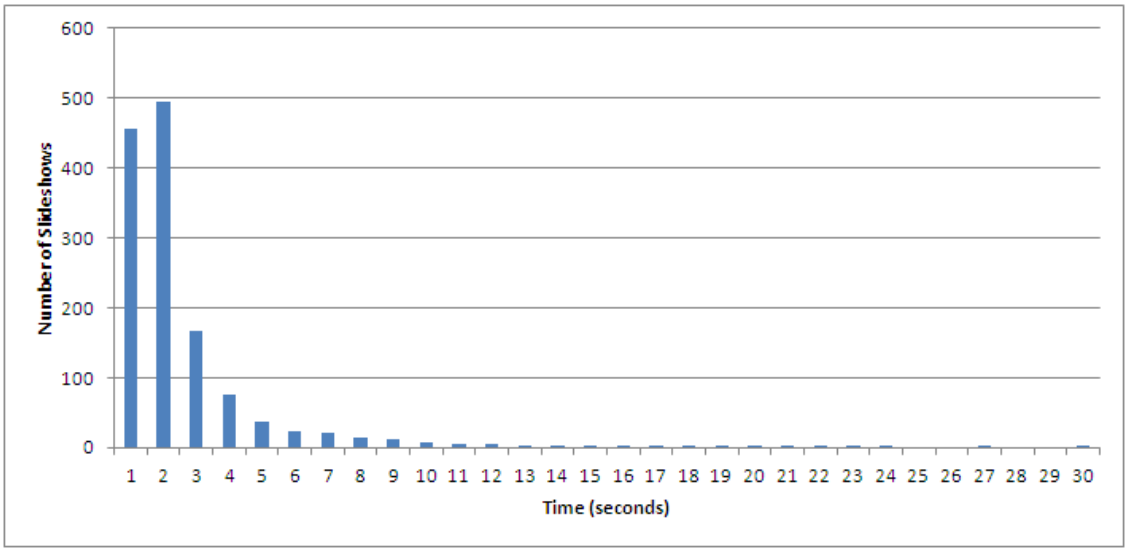


Figure 6.15: Mean Time Per Slide, CS3114A: A histogram showing the number of slideshows completed with a mean-time-per-slide in each time range (1 second resolution)

Class	**Quartile 1**	**Quartile 2**	**Quartile 3**
CS223	1 sec	4 sec	8 sec
CS3114A	1 sec	2 sec	3 sec
CS3114B	1 sec	2 sec	3 sec

Table 6.2: Quartiles Indicating Mean Time Per Slide: Respectively, each quartile indicates the mean time per slide on 25%, 50%, or 75% of slideshows

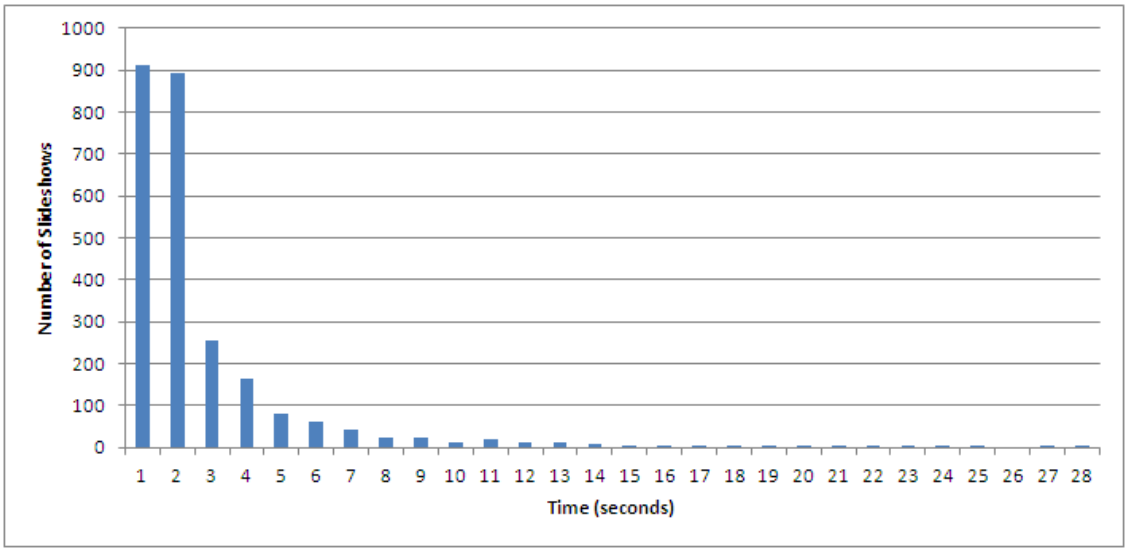


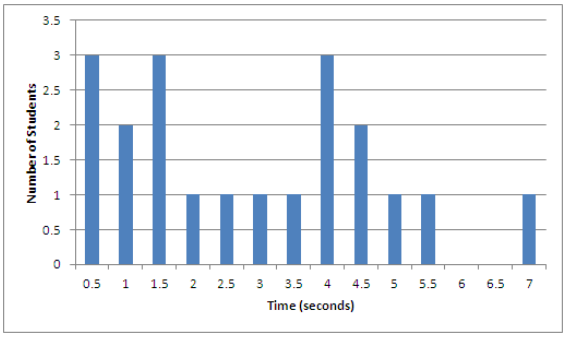
Figure 6.16: Mean Time Per Slide, CS3114B: A histogram showing the number of slideshows completed with a mean-time-per-slide in each time range (1 second resolution)

completed with a mean-time-per-slide of one second or less. The second and third quartiles were the same for both CS3114 classes, indicating 50% were completed with a mean less than two seconds and 75% with a mean less than three seconds. Both the second and third quartiles for CS223 were larger than the third quartile of the CS3114 classes, indicating students in this class tended to spend longer on slideshows. Overall, it appears a sizeable number of students in all classes click through the slideshows very quickly.

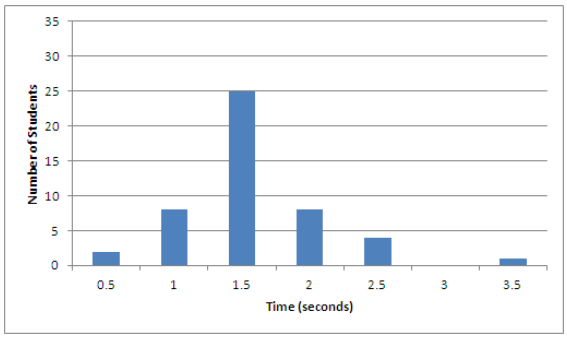
We also examined the distribution of student's median mean-time-per-slide, as seen in Figures 6.17. The distributions for both CS3114 classes were similar, consisting of a normal distribution centered around 1.5 seconds with a slight positive skew. This further supports the claim that students in general do not spend long on each slide. The results for CS223 were nearly evenly distributed with slight peaks at 0.5, 1.5, and 4 seconds. This illustrates a higher range of variability than is seen in the CS3114 classes.

Event data was used to determine the length of slideshows, while the total time was recorded by score data. Missing event data is thought to have little effect on the median, except in cases where a significant amount of event data is missing. For this reason accounts with known invalid event data were eliminated from our calculations as explained above.

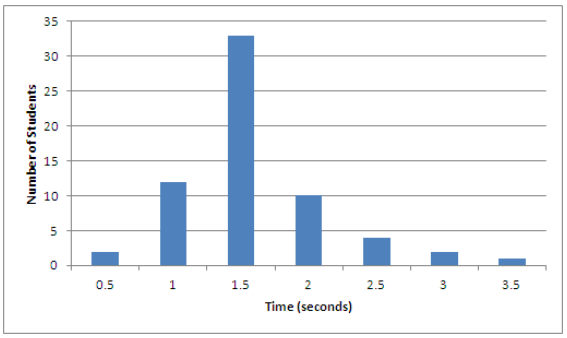
Unfortunately, using the mean-time-per-slide assumes a constant time is spent on each slide, which is unlikely to occur. In particular, given the small mean change, this technique is unable to differentiate between students who click through all the slides quickly and those who initially exhibit learning behavior and then transition to rushing behavior once they grasp the necessary concepts. Additionally, distractions could cause a single slide time to be significantly longer than the others, distorting the overall mean time. Future work utilizing



(a) CS223



(b) CS3114A



(c) CS3114B

Figure 6.17: Mean Time Per Slide Distribution By Student: A histogram showing the distribution of student's median mean-time-per-slide (0.5 second resolution)

more reliable event data should examine individual slide times to account for these factors.

6.3.3 Skipping to the End of Slideshows

As discussed in Section 5.3.3, it was possible for students to skip to the end of slideshows and receive credit without viewing all the slides. A small amount of this behavior was seen in CS223 and CS3114A, while a moderate amount was seen in CS3114B. Since all three classes had a median of zero slideshows skipped, we focused on the small group of students in each class who did skip.

The figures in this section use error bars to indicate the amount of known missing data and its possible effect. The positive error bars indicate the number of slideshow instances for which score data was recorded, but no event data existed. It is possible that some or all of these attempts could have resulted in skipping. The negative error bars indicate slideshows for which only some of the event data was recorded. Our analysis technique marked these as skipping because we cannot confirm every slide was viewed, however it is possible some of these were legitimate. Missing score data could further increase the potential amount of skipping observed, though we believe this data to be accurate as problems with score data would almost certainly elicit complaints from students.

As seen in Figure 6.18, five students in CS223 (23.8% of the class) skipped. While one student skipped on 8 out of the 16 slideshows, three out of the five only skipped a single slideshow, giving the group a median of one slideshow skipped.

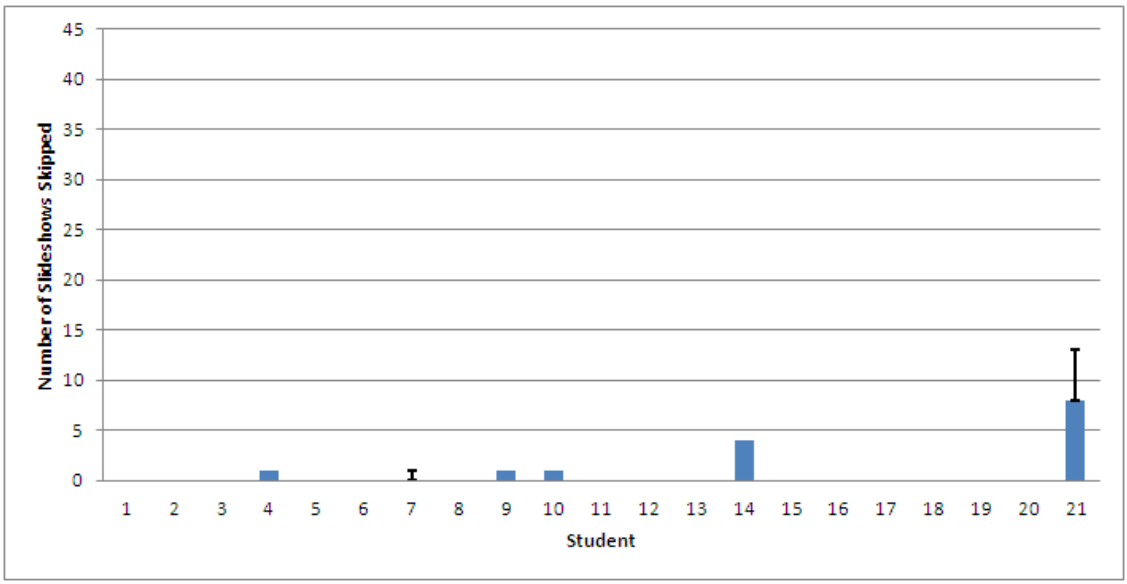


Figure 6.18: Slideshow Skipping, CS223: The number of slideshow instances skipped by each student. Positive error bars indicate the number of slideshow instances for which score data was recorded, but no event data existed. Negative error bars indicate slideshows for which only some of the event data was recorded

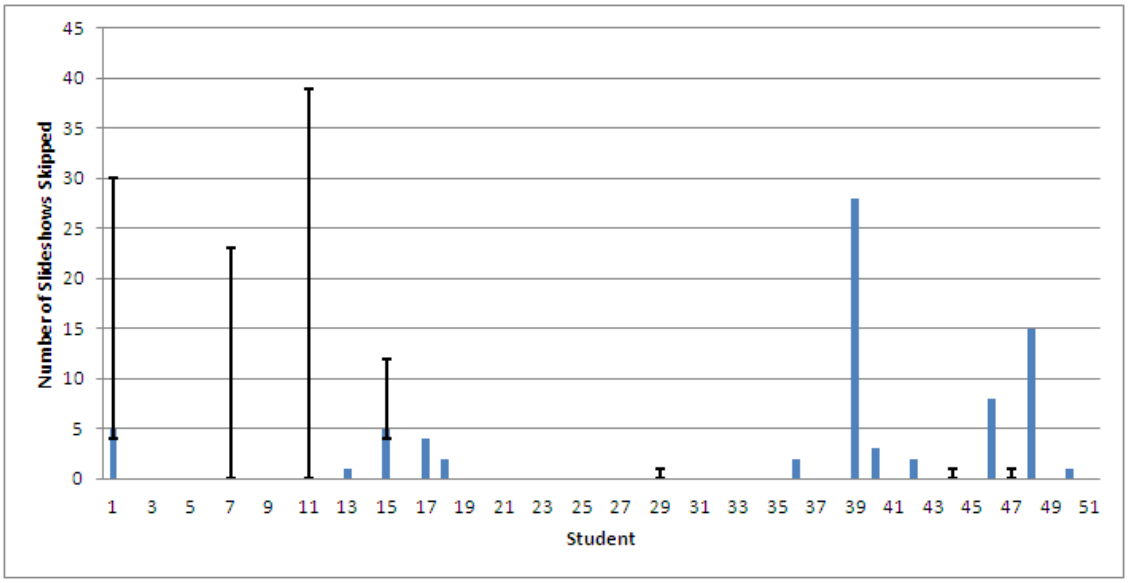


Figure 6.19: Slideshow Skipping, CS3114A: The number of slideshow instances skipped by each student

At 23.5%, CS3114A experienced a nearly identical percentage with 12 out its 51 students skipping, as seen in Figure 6.19. While 75% of the group skipped 5 or fewer slideshows, the remaining three students skipped 8, 15 and 28 of the 44 total. Overall the CS3114A group had a median of 3.5 slideshows skipped. Both the class percentage and median increased when considering CS3114B, which saw 23 of its 65 students, or 35.4% of its class, skipping. Only 30.4% of the CS3114B group skipped 5 or fewer slideshows, giving the group a median of 8 slideshows skipped. The results from CS3114B can be seen in Figure 6.20.

Inspection of the event data related to a sample of the skipped instances revealed two types of behavior. In the first, students viewed a portion of the slideshow, possibly moving backward and forward again to examine a specific operation, then skipped to the end. In the second, students skipped straight to the end without viewing any slides. While we still consider it skipping for the purposes of our analysis, we believe the first behavior is not necessarily harmful as the student initially displays "learning" behavior and appears to skip to the end only once they feel they understand the material. Unfortunately, we were unable to identify the motivation for the second behavior, though we believe it to be caused by either credit-seeking behavior or students skipping material with which they are already familiar (possibly material presented earlier in the module or in lecture). Future research should focus on separating these two behaviors for independent analysis.

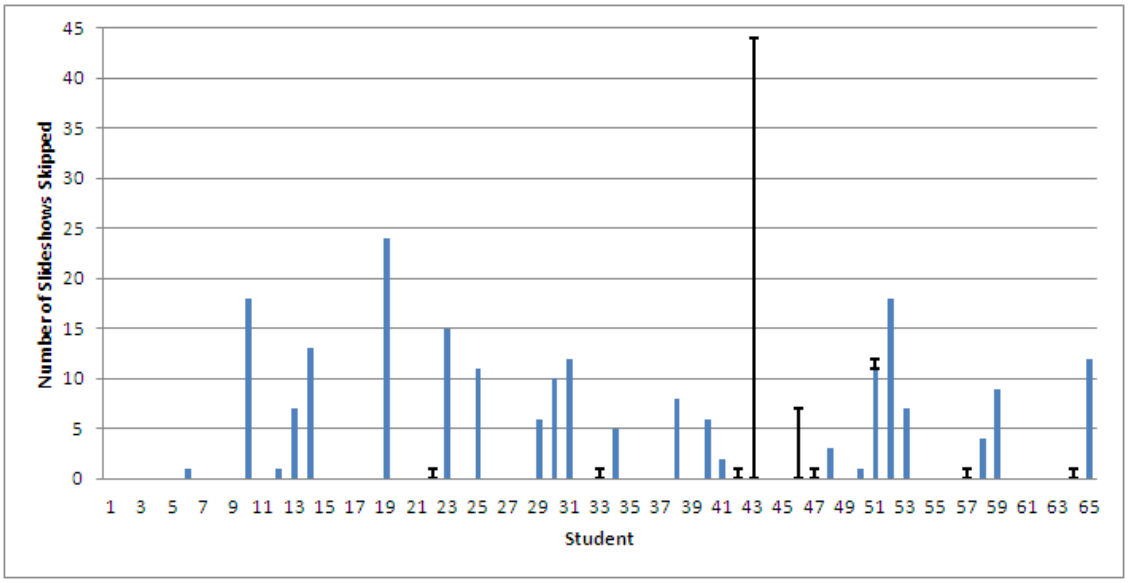


Figure 6.20: Slideshow Skipping, CS3114B: The number of slideshow instances skipped by each student

6.3.4 Using AVs to Complete Exercises

Results By Exercise

In this section we use four measures to analyze AV assistance with regard to specific exercises. These measures include the number of exercises attempted, completed, proficient, and assisted. Attempted exercises are unique exercise instances where a student opened or reset an exercise. Completed exercises are a subset of attempted exercises where the student completed every step, though not necessarily correctly. Proficient exercises are a subset of completed exercises where the student completed the exercise with a score above the exercise's proficiency threshold. Assisted exercises are a subset of attempted exercises where the student ran the proficiency exercise's associated AV with input identical to that of the exercise, before the exercise had been completed. Note that we were interested in general AV assistance, and an assisted instance does not necessarily imply a completed or proficient instance.

As seen in Table 6.3, the results from CS3114A and CS3114B were similar across all four measures. The Quicksort proficiency exercise was attempted the most by both classes and had the highest attempt-to-completion ratio in both classes at 15.56:1 for CS3114A and 14.01:1 for CS3114B. It also experienced the highest rate of assistance in both classes which suggests students experienced the most difficulty with this exercise. While it is possible the Quicksort algorithm may simply be more confusing to students, we suspect students may have difficulty determining what specific actions this exercise requires them to perform. This

	Exercise	Attempted	Completed	Proficient	Assisted
CS3114A	ShellsortProficiency	671	67	57	1
	mergesortProficiency	263	181	181	0
	quicksortProficiency	1105	71	52	14
CS3114B	ShellsortProficiency	561	90	68	2
	mergesortProficiency	127	68	67	0
	quicksortProficiency	1065	76	60	19

Table 6.3: Comparison of Proficiency Exercise Measurements: The number of exercises attempted, completed, completed with a score above the proficiency threshold, and the number of exercises where students used an AV for assistance

speculation comes from the interview after the Fall 2012 experiment which originally inspired this research topic. In the interview the student indicated he did not understand what he was supposed to do in the exercise and resorted to using the AV to help him complete the exercise. It is, therefore, unsurprising that this exercise amassed the largest number of assisted attempts.

The Shellsort exercise received roughly half as many attempts as Quicksort in each class, and AV assistance was only used once in CS3114A and twice in CS3114B. `mergesortProficiency` received the fewest number of total attempts in both classes, and neither class used AV assistance. It also had the best attempt-to-completion ratio out of any proficiency exercise with an average of 1.45:1 in CS3114A and 1.87:1 in CS3114B. These results indicate this exercise was most likely easier for students, thereby causing fewer resets or abandoned attempts.

With the exception of `ShellsortProficiency` in CS3114B, each exercise has a difference of less than 20 between the number of completed exercises and the number of proficient exercises. This tells us that most completions result in proficiency. Given the much larger number of attempts for each exercise we can surmise that students restart an exercise rather than completing it once they have lost too many points to obtain proficiency. Future research could verify this conjecture by examining the number of points lost before each exercise reset.

Furthermore, based on the number of proficient attempts and the number of students in each class, we know that very few exercises were re-attempted after students had obtained proficiency. The obvious exception to this observation is `mergesortProficiency` in CS3114A which had well over three times as many proficient completions as students in the class. We can therefore say with certainty that at least a subset of the CS3114A students completed this exercise more than once.

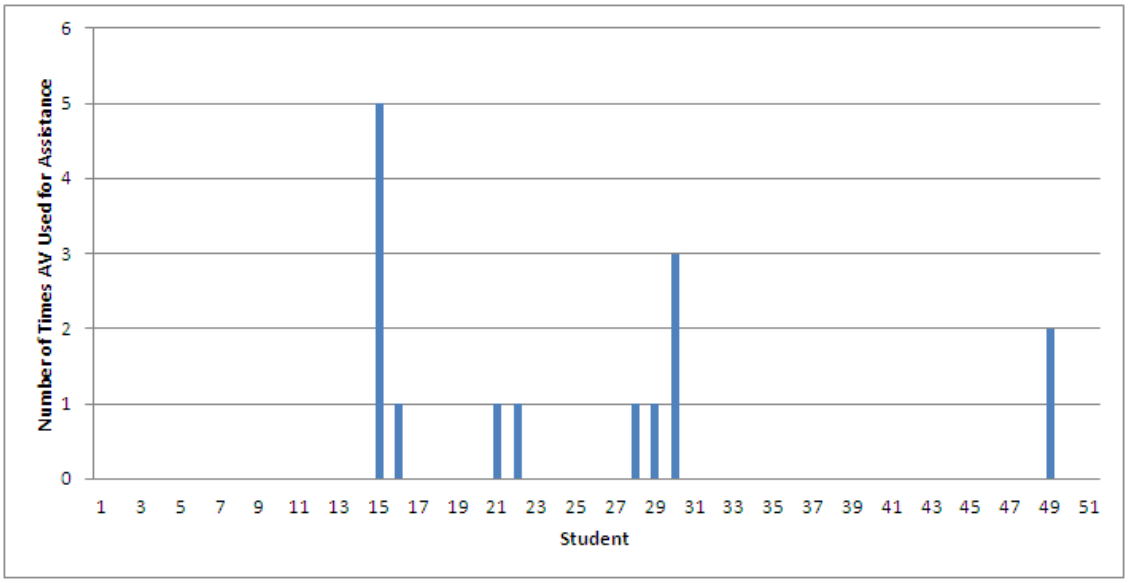


Figure 6.21: Assisted Exercises By Student, CS3114A: The number of exercises on which each CS3114A student used AV assistance

Results By Student

A total of eight CS3114A students (15.69% of the class) used AVs to assist in exercise completion compared with seven students in CS3114B (10.77% of the class). AV assistance was more evenly distributed in CS3114B with four students using AVs on four instances apiece, two using AVs on two instances, and one student using an AV only once. In CS3114A, one student used AV assistance five times, the maximum of any single student, while five students only used an AV on a single exercise instance. As seen in Figures 6.21 and 6.22, no trend is apparent within either class or between classes.

We decided to take this analysis a step further by considering how the group of students who used AV assistance compared with their non-assisted classmates in terms of total time required for proficiency. Using each student's median time for proficiency exercises, we found the assisted group in CS3114A had a median of 127.5 seconds, and the non-assisted group had a median of 84 seconds. The assisted group in CS3114B had a median of 119 seconds, and the non-assisted group had a median of 77.5 seconds. These results indicate students are not receiving any time-saving benefit from using AV assistance. No relationship was observed between the number of exercises on which AV assistance was used and median time required.

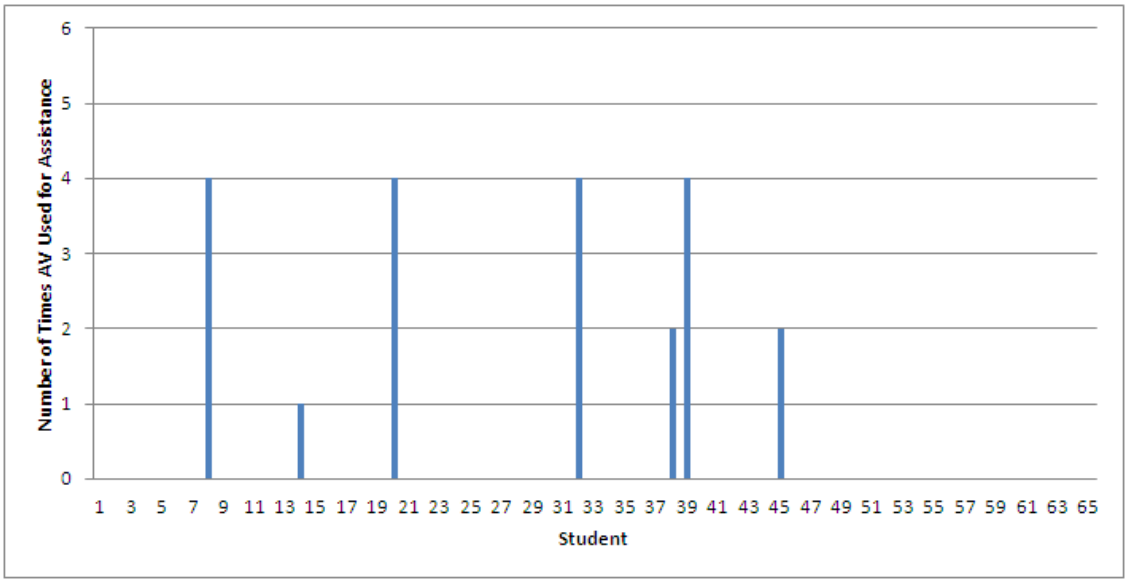


Figure 6.22: Assisted Exercises By Student, CS3114B: The number of exercises on which each CS3114B student used AV assistance

Discussion

Fortunately, overall levels of AV assistance on exercises appears to be quite low, especially when considering the number of attempts and completions. In the future we hope to decrease these levels further by making the Quicksort exercise more intuitive.

Regrettably, the effect of missing event data on these results was indeterminate as it depends on the specific events which were missing. If AV initialization events were missing, the number of assisted instances could increase, whereas if an "end" event such as an exercise reset or a page refresh were missing, it is possible the counts could decrease. Given the unlikely event that a student would run an AV with exercise data after closing the exercise, we are reasonably confident that our numbers form a lower bound for possible exercise assistance. However, it is important to note that our analysis only identified instances of proficiency exercises where students ran the associated AV with the exercise data and does not account for the number of AV steps viewed or whether the student obtained proficiency with the exercise instance.

6.4 Completion of Non-Required Exercises

The following graphs contrast the number of unique slideshows completed by both CS3114 classes. Positive error bars indicate the number of unique slideshows started but not completed, while negative error bars indicate the number of slideshow completions that were

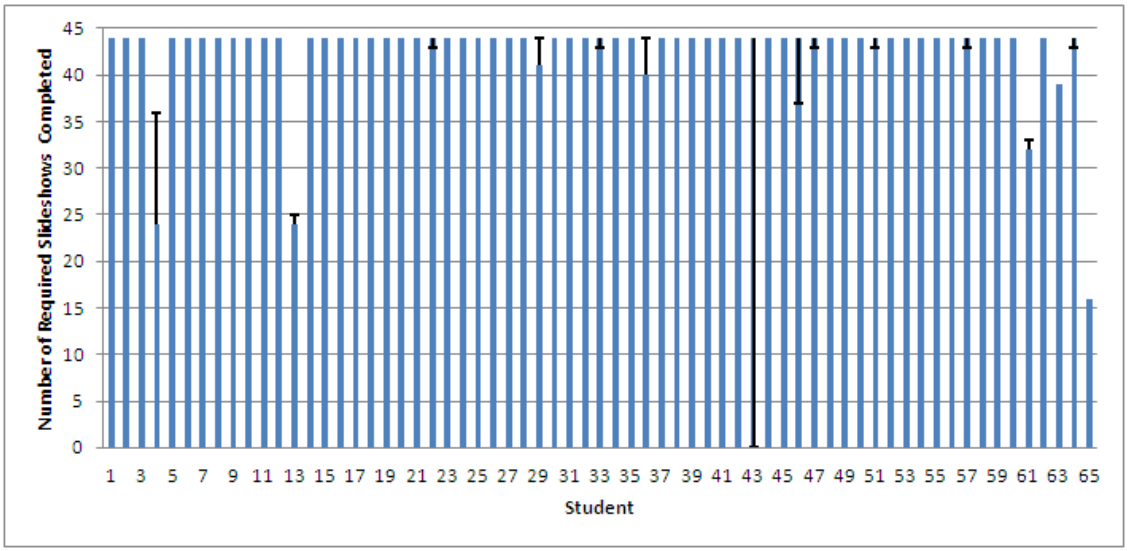


Figure 6.23: Required Slideshow Usage: Number of unique slideshows completed by CS3114B students. Positive error bars indicate the number of unique slideshows started but not completed, while negative error bars indicate the number of slideshow completions missing all associated event data

missing all associated event data. As seen in Figure 6.23, 100% of CS3114B used the slideshows and 89.23% completed all 44, giving the class a median of 44 slideshows, both started and completed. While only 3 students in CS3114A, or 5.88% of the class, completed all 44 slideshows, 100% used the slideshows to some extent, giving the class a median of 36 slideshows started and 32 slideshows completed.

When considering the credit-seeking behavior discussed in Section 6.3, we found low levels of skipping in CS3114A and moderate levels in CS3114B. While the overall amount of skipping was 12% lower in CS3114A than in CS3114B, this could simply be due to the first class completing fewer slideshows. Both classes showed equivalent amounts of rushing behavior. However, since CS3114A students had no external incentive to rush, this indicates that our analysis of rushing behavior may not be detailed enough to fully comprehend the behavior. While observed skipping behavior was low and rushing behavior was moderate to high, both were higher than expected. We remain unsure what motivated students to skip to the end or rush through slideshows where they received no credit.

We were encouraged by the moderate number of students who used slideshows of their own volition. Given the low levels of skipping and comparable levels of rushing, we believe the majority of CS3114A students used the slideshows as a learning resource. Since the number of completions was derived from score data, we are confident of its accuracy. The number of completions forms a definite lower bound for the number of slideshows started which was determined based on event data. Obtaining any missing event data could only increase the number of started slideshows.

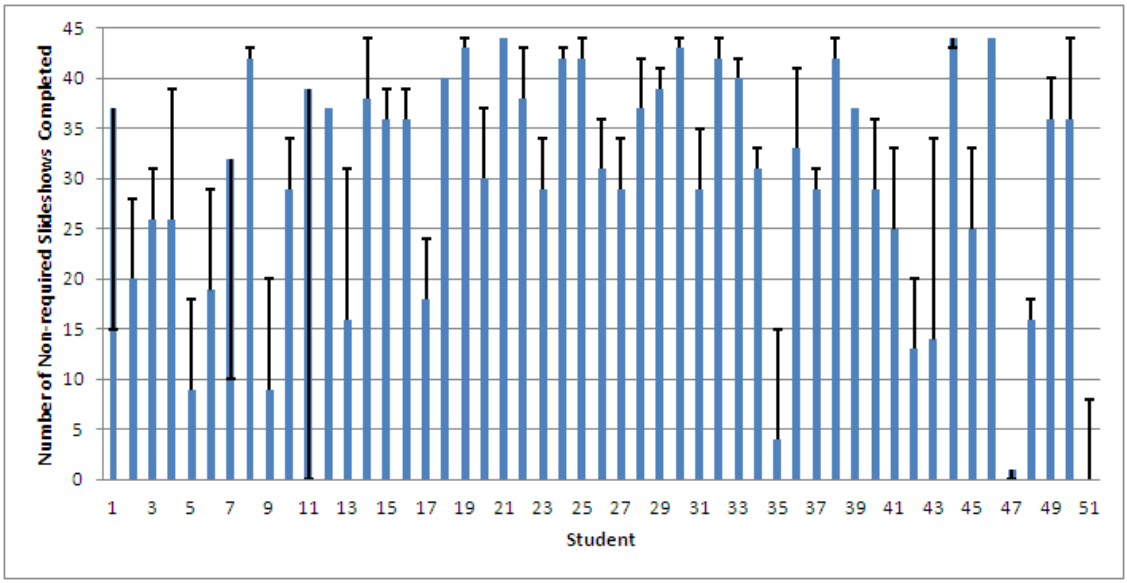


Figure 6.24: Non-Required Slideshow Usage: Number of unique slideshows completed by CS3114A students. Positive error bars indicate the number of unique slideshows started but not completed, while negative error bars indicate the number of slideshow completions missing all associated event data

Chapter 7

Conclusion

In this paper we described the architecture for OpenDSA, a collection of online tutorials designed to teach data structures and algorithms. We presented an analysis of log data generated from using OpenDSA in 3 classes, in order to characterize and quantify student interaction behaviors. The following list presents a summary of our findings.

- We were able to identify clusters of students based on when they completed exercises, which could be used to identify procrastinators and encourage them to start future assignments earlier.
- Little variation was observed between the two CS3114 sections regarding time required for proficiency on the exercises, which should allow us to reliably predict how long future students will take.
- We determined that a majority of students skip directly to the exercises without reading the text, which indicates that the existing text is not engaging enough or is too overwhelming for students.
- Higher rates of AV assistance were observed on proficiency exercises that were known to be confusing, indicating that this measure could be useful for identifying exercises that require additional development.
- Over three times as many proficient completions of one particular exercise were recorded than students in the class, confirming that students at least occasionally complete exercises after obtaining proficiency.
- Students who were not required to complete slideshows did not complete as many as the students who were, however, 100% of the students used slideshows to some extent and overall usage was fairly high.

- Skipping slideshows for credit occurred more often in CS3114B, where slideshows were required, than in CS3114A where they were not required. However, the behavior was still observed in CS3114A even though students did not receive credit.

7.1 Limitations

While care was taken to control as many factors as possible, there were a number of factors beyond our control which may have affected the results of our study. The biggest factors relate to missing or incorrect data and the inability to measure passive activities.

7.1.1 Missing and Incorrect Data

OpenDSA's data collection system experienced significant problems during the Fall 2012 experiment and, consequently, much work was put into increasing the reliability and robustness of the system prior to this study. Unfortunately, these efforts were not entirely sufficient as several discrepancies were noticed during data analysis. One such discrepancy indicates the lack of some event data, specifically more unique Unique Instance Identifiers (UIIDs) appeared in the score data than in the log data, whereas those in the score data should match or be a subset of those appearing in the event data. This problem is likely to have been caused by the way event data is written to and removed from local storage, as discussed in Section 3.2.6. Additionally, when event data is removed from local storage, it is held in memory during transmission to the DCS and would be lost if the student closed the window before the data was successfully transmitted. Also, since both event and score data are buffered in local storage, if students clear their local storage, then any data waiting to be transmitted will be lost. While the current method helps solve the previous problem of duplicate event data being stored in the database, it requires additional improvement.

A second instance of missing data appeared to affect only a handful students. The score data appeared to be recorded normally, but nearly no event data existed for these students. So far the cause has remained undiscovered, as these students bear no distinction from their classmates and no obvious commonalities between them. We contacted the affected students, but none had responded at the time this thesis was written. The effect on the data is believed to be minimal as this problem affected only a few students, and the analytics which rely solely on event data simply count those students as not completing the material.

The final issue involves score data recording an incorrect amount of time spent. This problem was thought to have been caused by a variable in the AV portion of the client-side framework being initialized to 0 rather than the current time. Since the framework calculates the time taken by subtracting the start time from the end time, when the start time was 0, the time taken is equivalent to the end timestamp or roughly 43 years. Fortunately, the bug only occurred under a particular set of circumstances and only affected 36 out of 81290 records.

Affected records were easily identified and additional processing was performed during analysis to reconstruct an approximate time taken based on other recorded information. In order to preserve its integrity, the original data was not altered.

7.1.2 Measurement of Passive Activities

The second major limitation of this study is the inability to measure passive activities. An attempt was made (described in Section 6.3.1) to determine whether students read the text or skipped ahead to the exercises. Unfortunately, all current data collection techniques rely on user interaction, making them especially poor at measuring passive activities such as reading which do not generate any interactions that can be logged. Another shortcoming is the inability to differentiate between taking a long time in order to learn a concept better and a student leaving the browser in the middle of an example or exercise. One possible way to address this would be to implement a timeout feature where OpenDSA goes inactive when it does not sense any user input for some amount of time. If this route is chosen, care must be taken to select an appropriate threshold and design an interface that will not negatively impact a student's user experience.

7.1.3 Minor Limitations

While there were relatively few reports, some students did encounter bugs with specific exercises or features. Unfortunately, there is not sufficient data to determine what, if any, affect this may have had on student behavior. Even if these students' original intentions were known, it is not known whether other students experienced bugs without reporting them. For the purposes of this study, the affect of bugs is discounted in particular because most trends are identified based on averaged measurements and bugs are not believe to have sufficiently altered average behavior.

Another limitation is the assumption (made at several points during the analysis) that since OpenDSA is mastery-based, students would complete all or nearly all of the material. In practice this assumption proved acceptable, with students completing a majority, though not all, of the assigned material. Since the assumption held, the effect is believed to be minimal. Additionally, students who fail to complete an exercise or module are simply not included when determining the related statistics, unless the lack of completion is central to the topic in question, minimizing the effect on the results.

7.1.4 Abandoned Accounts

Finally, while it is not believed to have negatively affected our results, the data related to several user accounts was disregarded during analysis. All of these accounts appeared to

have been created and abandoned in favor of new accounts. The majority of these accounts had not completed a single exercise, and either the username or the email address matched or closely resembled that of a more recent account which saw much higher levels of activity. A particularly interesting phenomenon occurred in CS223 where a group of five accounts created within the same time range were all abandoned. Given that all of the replacement usernames matched the student's email address and all other accounts created after that range bear the same pattern, we believe an account name convention was established within the class after these students had created accounts.

7.2 Future Work

The OpenDSA project is young, but growing quickly, and as such presents many opportunities for future study. Below we propose several improvements to OpenDSA based on our experiences during this study which we believe will simplify and increase the accuracy of future research. We also suggest additional research questions which we were regrettably unable to address within the scope of this paper.

7.2.1 Improvements to OpenDSA

One of the greatest challenges encountered while processing the data for analysis was attempting to calculate the total time that a user spent looking at a module. The existing mechanism relies on computing the difference between timestamps on module load and unload events, taking into consideration blur and focus events which, respectively, indicate when a student has stopped looking at and returned to a module page. Unfortunately, this technique is fairly fragile and especially vulnerable to missing data. We recommend this technique be replaced with a much simpler and more robust approach where the client-side framework increments a counter while the page has focus and appends a "focus time" field to every event sent to the server. Using this technique would improve several analysis activities such as calculating the total time a student views a module page or the time a student views a module page before starting an exercise.

7.2.2 Additional Research Questions

There are several research question relating to behavior characterization that we were unable to address due to time constraints or lack of log data. These involve using OpenDSA for studying, reloading pages to skip exercises, usage of specific features such as the model answer in proficiency exercises, the gradebook and the search functionality, the effect of incremental due dates, and identification of confusing concepts.

Studying

The first recommended future research topic focuses on students' use of OpenDSA to review material. The proposed study would investigate students' usage of modules and exercises with which they are already proficient. Specific questions include:

- Do students review modules and exercises after they have obtained proficiency?
- If students review exercises, do they surpass the proficiency threshold on subsequent attempts?
  - Does the length of time before the re-attempt affect the success? If a student re-attempts immediately are they more or less likely to be successful than if they re-attempt it after a longer period of time?

Reloading to Skip Exercises

The Khan Academy framework randomly selects questions from a pool each time the user loads an exercise or continues from a completed question to a new one. In order to achieve proficiency, students must obtain a certain number of points, where correct answers gain them 1 point and incorrect answers penalize them by 1 point. Students are not given an obvious means of skipping a question, but the framework maintains their score through page refreshes, meaning that a student must simply refresh the page in order to receive a new question. Future research could seek to detect and quantify this behavior.

Model Answer Usage

Most of the JSAV-based proficiency exercises offer students the option to view the solution to the current problem instance at the cost of no longer being able to receive credit for the current instance. We believe it would be interesting to investigate how and how much students use this feature. Specific questions include:

- What percentage of students used the model answers?
  - For each student who used the model answers, on what percentage of the exercises did they use the model answer?
- What exercises did students use the model answer on and how many times was it used on each?
  - Answering this could help to identify which exercise(s) might be most confusing to students

- What is the average number of model answer steps a student views? Students may open the model answer but not look at it

Gradebook Usage

One feature that was greatly improved since the Fall 2012 experiment is the gradebook. We want to know how important this feature is to users in order to determine how much support it should receive in the future. In addition to usage amount, an interesting research question would be whether students exhibit "gradebook-driven completion" which would be characterized by continually checking the gradebook to determine what they must complete next.

Search Usage

One of the benefits of many e-texts, including OpenDSA, is that they provide search functionality. Like the gradebook feature described above, we are interested to know how and how much students make use of this capability. By recording and analyzing the search terms, it may be possible to determine which terms students find most important yet difficult enough to find that they resort to searching the book.

Verification

In addition to tackling the additional behavior characterization questions discussed above, a useful contribution would be to further improve OpenDSA and the analytics used in this study and verify the results of this report through additional user studies. While not as glamorous as original research, independent verification is a vital step in the research process.

Effect of Incremental Due Dates

All OpenDSA studies to date involved a similar setup where students were given a period of time to complete the assigned work and a single date at the end when all of it was due. As evidenced by this study, that format allows students to procrastinate and does not enforce any user preparation prior to class. We suggest a future study investigate the effect on student behavior when a portion of the assigned work is due either before or after each class period rather than all at once.

Identification of Confusing Concepts

Analysis of interaction data may reveal specific slideshow or exercise steps where individual students or the majority of a class are struggling. We believe these steps can be isolated by considering the number of times slideshow steps are viewed, number of times an exercise step is performed incorrectly and the amount of time spent on each slideshow or exercise step. For instance, if a student is confused by a specific concept in a slideshow, he or she is likely to back up and view the related steps multiple times. This will result in the steps related to the topic of confusion having a higher number of views and likely a longer total view time than the surrounding material. If a sufficient number of students are confused by a specific concept, we expect the class average will exhibit a similarly identifiable behavior. If research shows that these specific steps can be isolated, instructors will be able to provide personalized assistance to individual students, address insufficiently explained material to the entire class during lecture, or improve the slideshow or exercise to reduce confusion.

7.3 Concluding Remarks

OpenDSA provides a much-needed solution to many problems that exist with traditional courses and textbooks such as passive learning, lack of practice material combined with helpful feedback, difficulty explaining abstract processes, cost, and scalability. While still in its infant phase, OpenDSA has great potential to enhance students' knowledge and understanding of DSA materials. The results of this research will allow us to improve OpenDSA, making it better meet the needs of students. As OpenDSA matures and adoption increases, it may even become an archetype for a new form of course material presentation spanning many disciplines.

Bibliography

[1] Baraniuk, R. How open is open education? *Domus* (March 2009), 1.

[2] Brandl, G. Sphinx python documentation generator. http://sphinx-doc.org/index.html, 2011.

[3] Hall, S., Fouh, E., Breakiron, D., Elshehaly, M., and Shaffer, C. Education innovation for data structures and algorithms courses. In *Proceedings of ASEE Annual Conference* (Atlanta GA, June 2013).

[4] Karavirta, V., and Shaffer, C. JSAV: The javascript algorithm visualization library. In *Proceedings of the 18th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2013)* (Canterbury, UK, July 2013).

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

[6] The MIT license. http://opensource.org/licenses/MIT, 2013.

[7] Naps, T., Rössling, G., and nine more authors. Exploring the role of visualization and engagement in computer science education. In *ITiCSE-WGR '02: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (2002), pp. 131–152.

[8] OpenDSA website: Open source interactive data structures and algorithms. http://algoviz.org/OpenDSA/, 2013.

[9] restructuredtext: Markup syntax and parser component of docutils. http://docutils.sourceforge.net/rst.html, 2010.

[10] Shaffer, C., Akbar, M., Alon, A., Stewart, M., and Edwards, S. Getting algorithm visualizations into the classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)* (2011), pp. 129–134.

[11] Shaffer, C., Cooper, M., Alon, A., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education 10* (August 2010), 1–22.

[12] Shaffer, C., Karavirta, V., Korhonen, A., and Naps, T. OpenDSA: Beginning a community hypertextbook project. In *Proceedings of the Eleventh Koli Calling International Conference on Computing Education Research* (Koli National Park, Finland, November 2011), pp. 112–117.

[13] Shaffer, C., Naps, T., and Fouh, E. Truly interactive textbooks for computer science education. In *Proceedings of the Sixth Program Visualization Workshop* (Darmstadt, Germany, June 2011), pp. 97–103.

[14] Silvasti, P., Malmi, L., and Torvinen, P. Collecting statistical data of the usage of a web-based educational software. In *Proceedings of the IASTED International Conference on Web-Based Education* (2004), pp. 107–110.