CHAPTER

8 AUTOMATING LEARNER FEEDBACK IN AN eTEXTBOOK FOR DATA STRUCTURES AND ALGORITHMS COURSES

E. Fouh, S. Hamouda, M.F. Farghally, C.A. Shaffer
Virginia Tech, Blacksburg, VA, United States

1 INTRODUCTION

Perhaps the greatest contribution of the computer to education is the ability to give immediate feedback to a learner about the state of their knowledge. This feedback often comes in the form of automated assessment for practice problems. Assessment systems with automated feedback provide more practice for students with less grading time for instructors. Self-assessment can increase learners motivation, promote students' ability to guide their own learning, and help them internalize factors used when judging performance (Andrade and Valtcheva, 2009; McMillan and Hearn, 2008). Other forms of feedback include "gamification" elements where students are informed about progress through the material, and they might be given "rewards" for progress in an effort to provide motivation. Instructors interested in experimenting with new pedagogical approaches (blended learning, "flipped" classroom, etc.) might find that the path to implementing these pedagogies is easier due to the affordances provided by new learning technology.

Online educational systems can also provide feedback to system developers and instructors about the performance of their students. The most obvious type of feedback to instructors is level and time of completion for individual students. Another type of feedback is student analytics data in the form of logs to record keystrokes, mouse clicks, and timestamps, as well as higher-order information such as performance on practice problems. Such information is sometimes referred to as learning analytics (Ferguson, 2012; Siemens and Baker, 2012). With such information, developers can hope to first deduce patterns of student behavior, then correlate the different behavior patterns to successful outcomes, and finally improve the system so as to enhance student performance in various ways.

In this chapter, we discuss all of these aspects of feedback given by online courseware to students, instructors, and system developers. We provide a number of case studies. The context for our case studies is the OpenDSA eTextbook system (Fouh et al., 2014b). OpenDSA is an open-source project with international collaboration that seeks to create an infrastructure and a body of materials appropriate for use in a range of computer science courses. The bulk of the content relates to data structures and algorithms (DSA). This particular topic area lends itself to an eTextbook treatment for two reasons.

First, a significant fraction of the content relates to understanding the dynamic process of computer algorithms. This means that visualizations of the behavior of the algorithms (referred to as algorithm visualizations or AVs) can be especially useful to learners. Second, as with many disciplines, students are traditionally unable to get sufficient practice exercises due to limitations on the part of instructors' abilities to give feedback through grading of the students' work. With OpenDSA, we are able to give students a rich collection of practice questions and exercises.

The basic instructional unit for OpenDSA materials is the module, which represents a single topic or part of a typical lecture, such as a single sorting algorithm. Each module is a complete unit of instruction and typically contains AVs, interactive assessment activities with automated feedback, and textbook quality text. Modules can be grouped together into chapters, such as might be found in traditional paper books. OpenDSA content is built using HTML5 and JavaScript, making it device and browser independent. AVs are built using the JavaScript algorithm visualization (JSAV) library (Karavirta and Shaffer, 2013). OpenDSA includes extensive support for collecting learning analytics data.

Many OpenDSA exercises are "algorithm simulations." These require that the student manipulate a data structure to show the changes that an algorithm would make to it, such as clicking to swap elements in an array or clicking on appropriate nodes in a tree or graph. The AVs and algorithm simulation exercises are specifically designed to be manipulated either through mouse and pointer interactions or touch interactions when using touchscreen devices. We generally refer to algorithm simulation exercises, as "proficiency exercises." This type of exercise was inspired by the TRAKLA2 system (Korhonen et al., 2003), and many of OpenDSA's proficiency exercises were created in collaboration with the team that created TRAKLA2. We make use of the Khan Academy framework (http://github.com/Khan/khan-exercises) to provide support for multiple choice, T/F, and custom interactive exercises that we call "mini-proficiency" exercises. We will refer to these collectively as KA exercises. All exercises are automatically graded and provide immediate feedback to the user. Students can repeat exercises as many times as they want until they get credit, or repeat them as a study aid for an exam.

Several experiments have shown the pedagogical effectiveness of TRAKLA2's proficiency exercises. Laakso et al. (2005, 2009) reported on the use of TRAKLA2 exercises in data structures and algorithms courses at two universities in Finland. TRAKLA2 exercises were incorporated as classroom (closed lab) exercises, and supplemented lecture hours and classroom sessions. TRAKLA2 exercises were also incorporated into the final exam (one out of five questions) and midterm (they replaced half of a question with a TRAKLA2 exercise in a total of five questions). A year after the introduction of TRAKLA2 in the curriculum, the activity of students increased in all aspects of the course, not only the part involving TRAKLA2 exercises. The average performance in classroom exercises rose from 54.5% to 60.3% (as percentage of exercises completed). A study (Karavirta et al., 2013) of several TRAKLA proficiency exercises translated to OpenDSA demonstrated that they remain equally effective. Student opinions of TRAKLA2 were collected through an online survey. Eighty percent of the students reported having a good opinion of TRAKLA2. After a year of using TRAKLA2, the opinion of students on its suitability for instruction rose significantly. Ninety-four percent of students agreed that TRAKLA2 exercises helped in their learning process. Their opinion did not change regarding their preference on the medium used to access and perform the exercises: they preferred a mixed (online and traditional classroom) experience.

2 MONITORING THE STUDENT LEARNING PROCESS: WHAT WE LEARN FROM SURVEYS AND LEARNING ANALYTICS

Online courseware systems provide the opportunity to automatically log massive amounts of user interaction data in fine detail, to the level of individual mouse events. This wealth of information might be used in a number of ways to improve the pedagogical value of online materials. Developers can hope to learn from interactive log data simple things like, which exercises are taking an unreasonable amount of time or which ones appear too easy because students never get them wrong. But careful analysis of log data can hope to deduce more complex behavior.

For example, by examining the times of various interactions, we should be able to determine whether students are reading the materials before attempting the associated exercises. We can hope to tell whether students are viewing all of the slides in a visualization, or are skipping through them. We should be able to tell whether students are going back to the materials to use them to study for a test by the fact that they look at them a day or two before the test, even if they had previously received credit for completing them.

Based on this feedback, we can then try to better engineer the courseware, for example, to make sure that students do not solve problems by guessing, or skip over key material. Intelligent tutoring systems (Corbett et al., 1997; Roll et al., 2011; Feng et al., 2014) can be built to react to the behavior of an individual student, attempting to identify and directly address the underlying misconception that leads to providing an incorrect answer to an exercise. Instructors can be given better feedback on student performance. For all of these reasons, learning analytics and techniques for analyzing the data are poised to become a major focus of CS education research (and, more generally, CS contribution to education research in all disciplines).

OpenDSA collects log data for all user interactions occurring on an OpenDSA web page. Since Spring 2013, we have been able to collect detailed interaction data for hundreds of students per year who are using the system, amounting to millions of individual interaction events. Interaction log data plays the central role in all of the feedback mechanisms that we discuss in this paper. The act of collecting the data itself typically does not directly impact students (though more sophisticated intelligent tutoring systems can use this information for better adaptive feedback (Roll et al., 2011)). But it is this log data that allows us to deduce most of the results presented in the following sections about how students are using the system. This allows us to understand the role that feedback to the students is actually having on student learning. This in turn can allow us to actively improve the system in ways that will lead to better learning outcomes.

In this section, we first present several use cases for using interaction log data to understand student behavior with the OpenDSA system. We then present data that we have collected related to student perceptions regarding the value of OpenDSA, and their preferences for how the system should be used in courses.

Throughout this paper, our case studies and our discussions of student surveys and log data will refer to various classes. The vast majority of classes that have used OpenDSA so far can be categorized as one of two types. We use the term "CS2" to refer to a second semester programming course in computer science. This course is typical for most CS departments, and includes some combination of basic software engineering and introductory data structures (linear structures, some simple sorting algorithms, and some introductory material on binary trees). We use the term "CS3" to mean a course that comes after the typical CS2 course, primarily focused on more advanced data structures. This typically includes algorithm analysis, advanced sorting algorithms, advanced tree structures, hashing, and perhaps an introduction to file structures.

2.1 MONITORING LEVELS OF STUDENT ENGAGEMENT

Instructors have always recognized that students have difficulty with certain topics. In at least some instances that we have studied, we find that one source of this difficulty relates to a consistent lack of engagement of students with the learning materials on that particular topic. In other cases, we find that difficulty arises from insufficient time spent by students engaging with the material due to lack of necessary practice activities. We illustrate both of these situations in our case studies. Further details can be found in Fouh et al. (2014a).

2.1.1 Case study: Recursion

Recursion is known to be a hard topic at both the CS2 and CS3 levels. Surveys from 15 instructors for CS2 courses indicate that these instructors believe students should spend an average of 10 h studying and practicing recursion outside of class. Surveys of 54 students in a CS2 course indicate that students were spending an average of only 4 h in out-of-class study and practice of recursion. The instructors unanimously felt that students were not spending enough time, even before having this confirmed by the survey data. Students come out of the typical instructional experience with a below-average feeling of confidence (2.6 out of 5 on a Likert scale for the students that we surveyed).

This information provides guidance for the design of a new tutorial on recursion. We see that goals for an online tutorial should include a significant body of practice exercises, and that reading tutorial content and working through the exercises should be calibrated to encourage students to spend about 10 h on the topic.

We created an online tutorial on recursion, presented as a chapter in OpenDSA. Students using this material during Spring 2015 reported spending an average of 7.3 h out of class, a marked increase over the prior semester's average of 4 h. From analyzing log data, we find that the median time spent on the recursion tutorial exercises is approximately 5 h. This does not include time spent reviewing the tutorial content (reading material and viewing the slideshows). Measuring reading times is much more difficult since our only logged interactions come when the page is opened and then when it is closed or passed on to another page. This means that actual measurements for "reading" material could include time spent away from the computer. However, it appears that our log data confirms the student estimates of 7 h or so. This is a significant finding in itself, since it supports a claim that we can trust data collected from either one of these sources (interaction log data or survey data where students estimate time spent on an activity).

2.1.2 Case study: Recursion exercises

The recursion tutorial discussed earlier includes a substantial number of programming exercises. A few of these exercises are relatively difficult for many of the students. As a result, many students are motivated to seek ways to get around doing the exercises.

The exercises are implemented based on the Khan Academy Exercise Framework. The fundamental approach is that a given programming "exercise" contains several specific programming problems. Of these individual problems, students are supposed to do a random subset, say two or three randomly selected out of five. Unfortunately, students can view the problem instance that comes up, then reload the page without penalty to get another problem at random. As a result many students will do one problem, then keep reloading the page until that problem repeats, then do it again. Through analysis of student responses to the programming exercises, we have found the following:

1. 52% of students who attempt the programming exercises do — on at least one instance — exhibit this behavior of reloading the page to get the same problem that they have just solved correctly, then resolve it to get a second credit instead of solving another one. We will name those students "Proficiency Seekers."
2. 93% of the "Proficiency Seekers" have exhibited this behavior in the exercises the students consider as hard. According to the student surveys, the students consider the exercises that ask them to write a full function or complete a nonsimple recursive function that has more than one recursive call or base case as hard. We have not found that students exhibit this behavior for simple code completion or the tracing exercises.
3. 42% of "Proficiency Seekers" have repeated this behavior for all three recursive function writing exercise sets.

This case study illustrates several points:

1. Log data allows us to deduce this behavior.
2. The details of exercise framework implementation have a huge impact on student behavior.

As a result of this study, we are reengineering first the tutorial and then the framework infrastructure to avoid the negative behavior.

2.1.3 Case study: Algorithm analysis

For a topic such as a particular sorting algorithm, an OpenDSA module (like a typical textbook presentation) contains both material on the dynamic behavior of the algorithm, and analytical material in the form of a runtime analysis (that is, the "algorithm analysis") of that algorithm. While dynamic behavior is well presented in OpenDSA by use of AVs, the analytical material relies mainly on textual discussion supported by static images. In other words, OpenDSA has historically presented algorithm analysis topics in a manner indistinguishable from a standard textbook. As a whole, the AV development community has a lot of experience with creating visualizations for the dynamic behavior of algorithms, but almost none with presenting analytical material (Shaffer et al., 2010). We also know from prior research into AVs in general that the level of student interaction with the material is a crucial factor in learning (Hundhausen et al., 2002; Naps et al., 2002). Unfortunately, algorithm analysis is also one of the most difficult topics in a typical CS2 course (Parker and Lewis, 2014). Thus we have the situation that our least engaging content is on one of our most difficult topics.

In order to see whether the typical "textbook" approach to presenting algorithm analysis content is effective for students, we analyzed student interaction logs from a CS3 course during Fall 2014. In particular, we focused on student engagement with OpenDSA's algorithm analysis material presented in the sorting chapter. This chapter contains more than 40% of all algorithm analysis material in that course. Unfortunately, OpenDSA's data collection tools (Breakiron, 2013) don't provide a direct method that we can use to estimate the time spent by students reading the analysis material in each module, because the main focus of these tools is to collect user interactions with the interactive content. Accordingly, the OpenDSA authoring system was extended to support wrapping arbitrary module sections behind a "show/hide" button as is done with other interactive content. When the student first sees the module that presents a given sorting algorithm, the algorithm analysis material is initially hidden, and a button is shown in its place. When the button is pressed, the analytical content is displayed. The time is recorded, and can be used as an estimate of the time when the student started reading the material. The time for finishing the material can be estimated from the time of the next user interaction, such as loading the next exercise or leaving the module which is recorded by the data collection tools.

We analyzed the interaction logs available for three OpenDSA book instances used at three different universities (Virginia Tech, University of Texas El Paso, and University of Florida). Fig. 8.1 presents the distribution of the estimated reading time for three sorting modules (Insertionsort, Mergesort, and Quicksort) for the Virginia Tech students. Similar results were found for students at the other universities. Results are summarized in Table 8.1. As we see from the results, more than 74% of the students spent <1 min on the analysis material for each of the three modules. Based on this result, we believe that most of the students are not reading the analysis material.

When the students were asked to provide suggestions for improving the presentation of the analysis material in OpenDSA, we found that most of them are expecting a more interactive presentation in the form of visualizations. Representative quotes from the survey include: "Visualizations definitely help," "I think making the click through pictures into actual animations would be nice," "more animation, the visualizations are great!," "more visualizations is always good," "Visualizations always help," "visualizations showing each step of analysis would help," "an animation will make a much bigger difference."

Based on this information, we determined to make new presentations for this content that would result in students spending more time using the material. This could happen in two ways. One is to motivate students to spend more time through the use of a more appealing presentation, perhaps visualization. The other is to force students to spend more time as a byproduct of completing specific tasks that are required to get credit for completing the tutorial. This might be engineered through a combination of requiring students to view slideshows (though these might be quickly skipped through), answering questions, and working exercises.

As an initial step, we have taken the approach of providing a visual presentation of the material. Inspired by the concept of visual proofs (Goodrich and Tamassia, 1998), a set of algorithm analysis visualizations were implemented for OpenDSA sorting modules (Insertionsort, Bubblesort, Selectionsort, Mergesort, Heapsort, Quicksort, Radixsort, and the lower bounds for sorting). Fig. 8.2 shows some examples of the new visualizations.

We have collected preliminary data with two small classes using the sorting analysis visualizations. Summary results for the two modules teaching Insertionsort and Quicksort appear in Table 8.2. We performed Kruskal Wallis tests to see if there is any significant difference between the time spent on text versus visualizations for these two modules. In each case, the difference in the means is significant ($p < 0.00002$), indicating that students spend more time on the material when presented as visualizations.

2.1.4 Case study: Student use of exercises for study

We sought to determine if students would only use OpenDSA when required for homework credit, or if they would use it voluntarily such as to review for exams. Participants included in this study were students enrolled in a CS2 course and a CS3 course at Virginia Tech during Spring 2015. We used

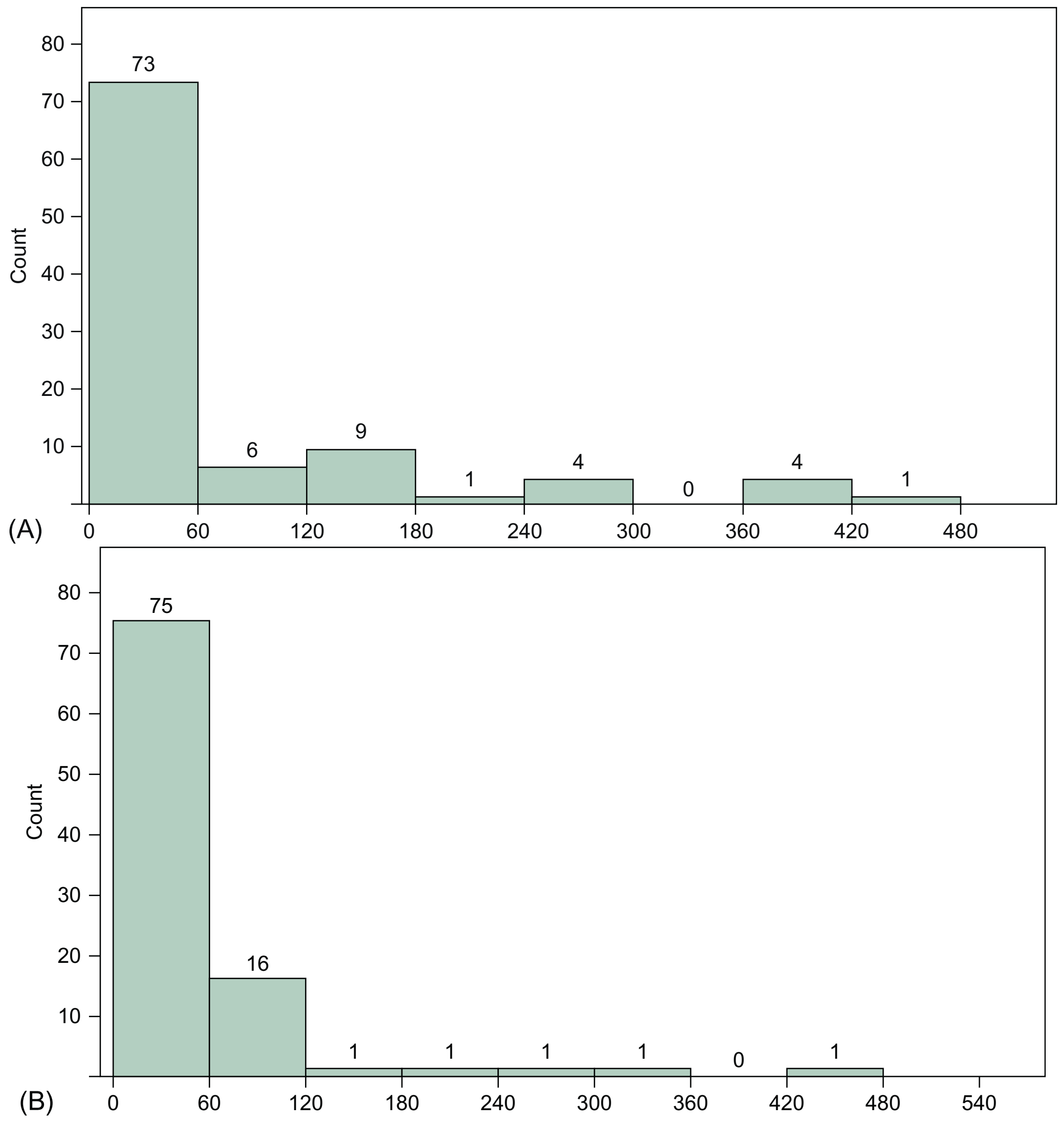


FIGURE 8.1

Estimated time spent by Virginia Tech students reading the analysis material. (A) Insertionsort module, (B) Mergesort module, and

Continued

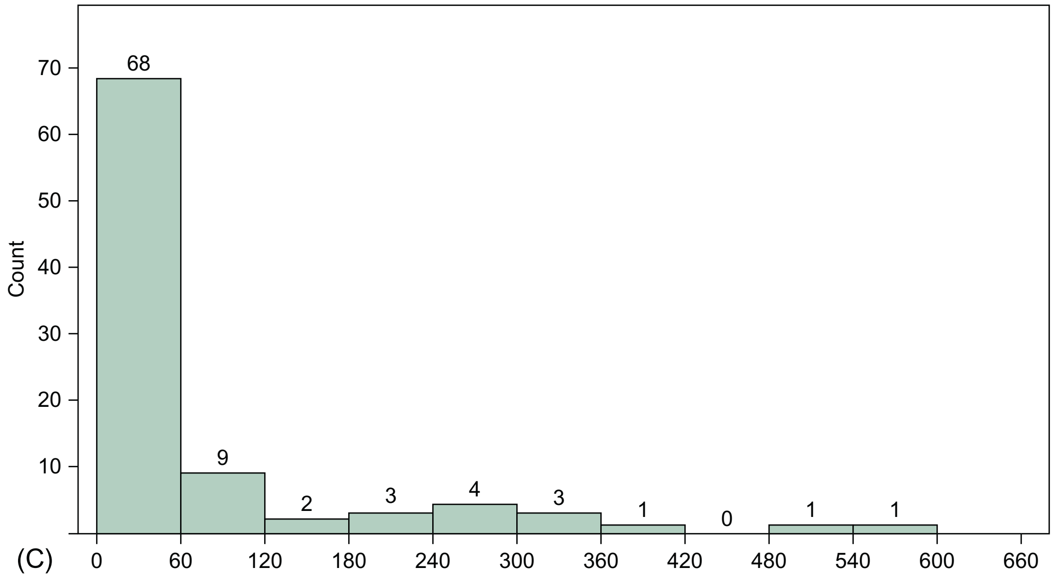
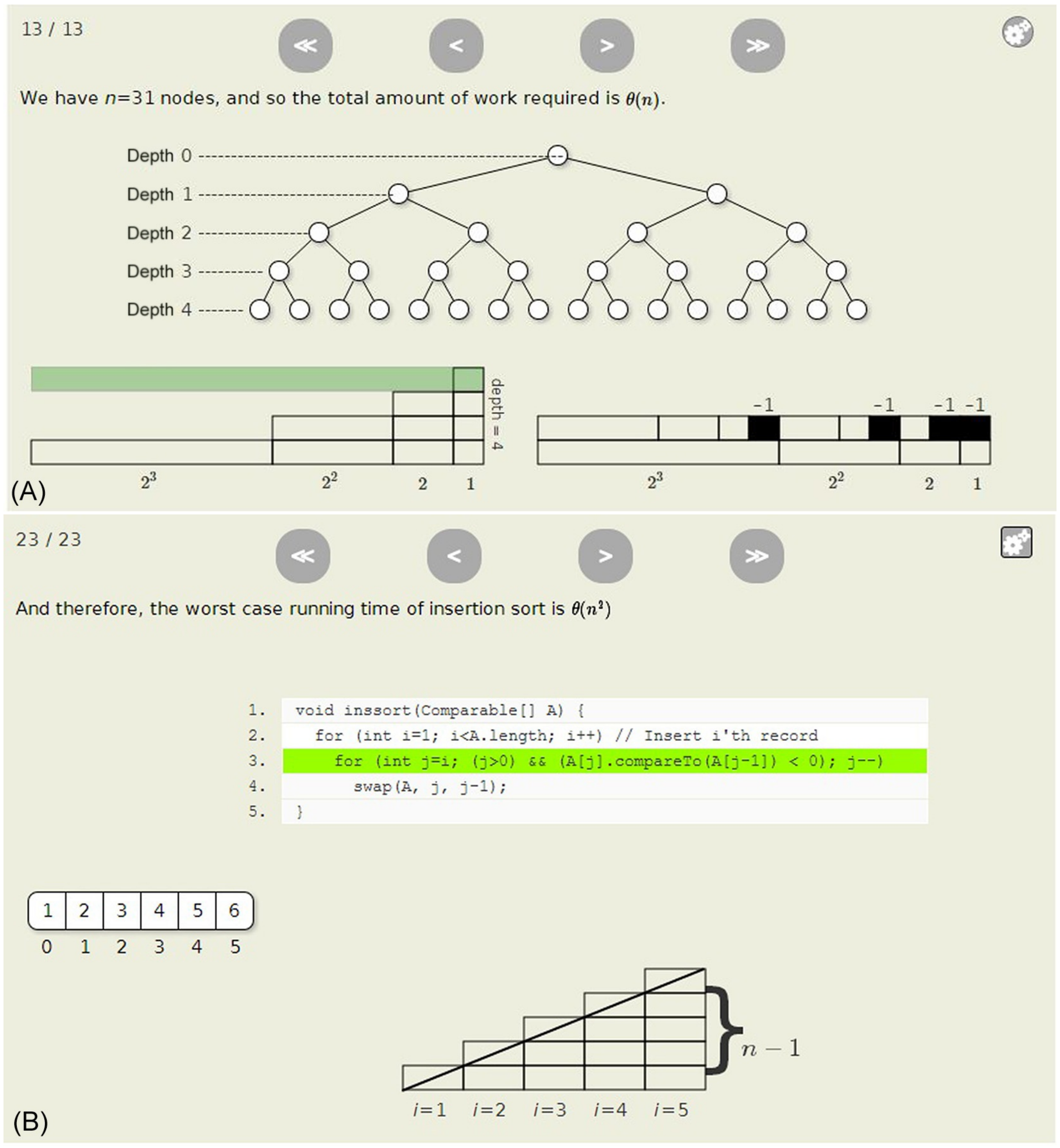


FIGURE 8.1, CONT'D

(C) Quicksort module.

Table 8.1 Results Summary

University	Module	*N*	Mean (S)	% <1 min
Virginia Tech	Insertionsort	98	63.57	74.48
	Mergesort	96	39.79	78.12
	Quicksort	92	64.71	73.91
UTEP	Insertionsort	26	49.84	80.76
	Mergesort	22	41.45	77.27
	Quicksort	16	16.18	93.75
Florida	Insertionsort	53	40.39	84.90
	Mergesort	44	18.63	95.45
	Quicksort	39	26.12	92.30
All	Insertionsort	177	54.6	78.52
	Quicksort	147	49.2	80.94



13 / 13

We have n=31 nodes, and so the total amount of work required is $\theta(n)$.

23 / 23

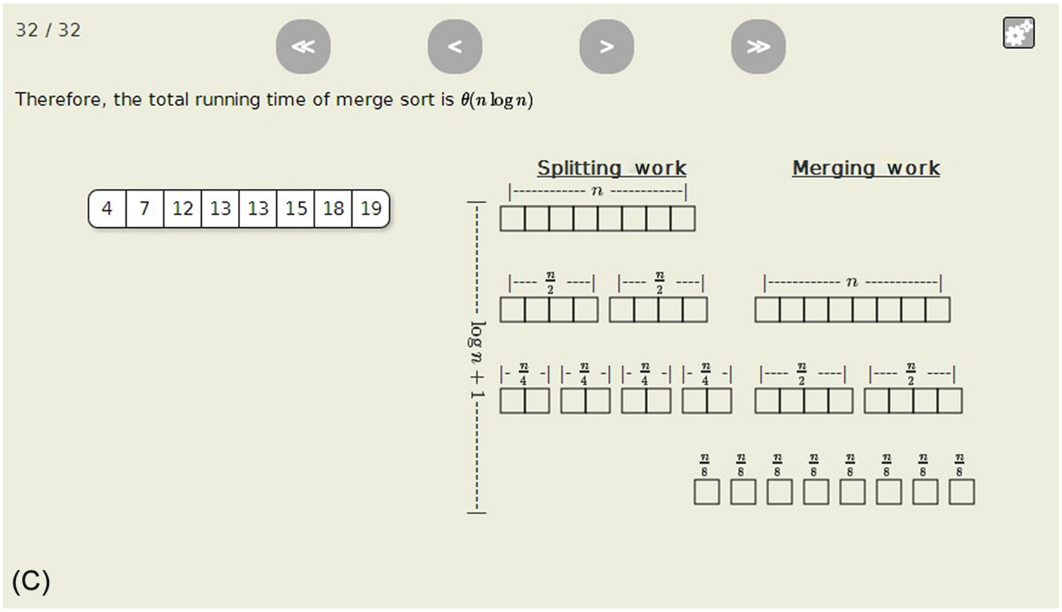
And therefore, the worst case running time of insertion sort is $\theta(n^2)$

```
void inssort(Comparable[] A) {
  for (int i=1; i<A.length; i++) // Insert i'th record
    for (int j=i; (j>0) && (A[j].compareTo(A[j-1]) < 0); j--)
      swap(A, j, j-1);
}
```

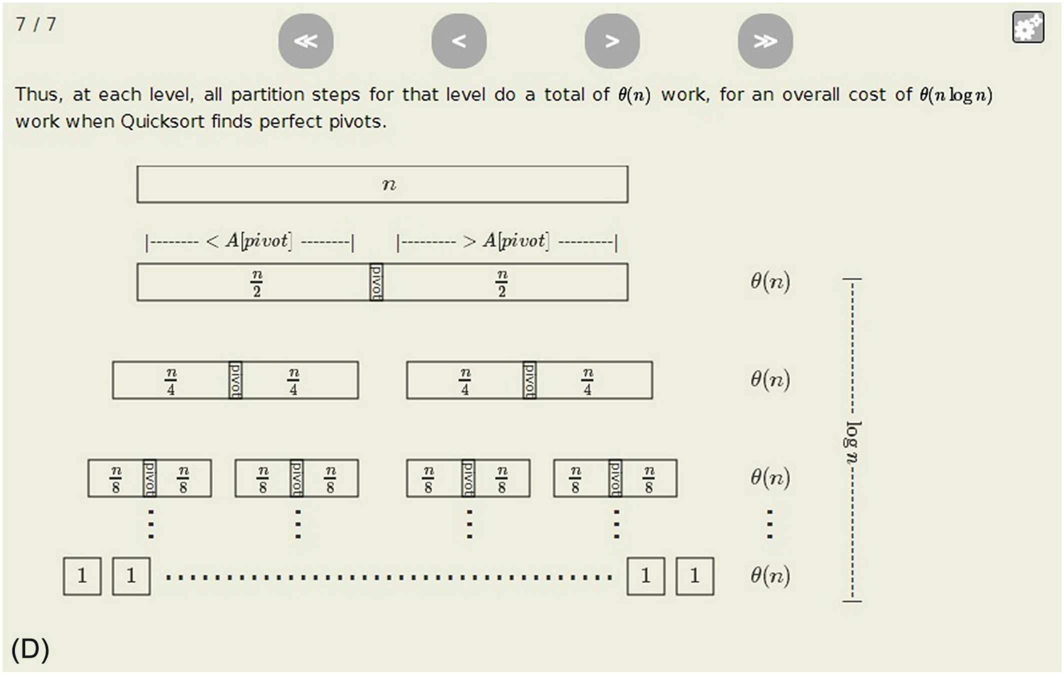
FIGURE 8.2

Visualizations illustrating the running time analysis of some sorting algorithms. (A) Build heap analysis, (B) Insertionsort worst case analysis,

Continued


32 / 32

Therefore, the total running time of merge sort is $\theta(n \log n)$


7 / 7

Thus, at each level, all partition steps for that level do a total of $\theta(n)$ work, for an overall cost of $\theta(n \log n)$ work when Quicksort finds perfect pivots.

FIGURE 8.2, CONT'D

(C) Mergesort analysis, and (D) Quicksort best case analysis.

Table 8.2 Text vs. Visualization Times, Measured in Seconds

	Text			Slideshow		
Module	**Median Time**	**Mean Time**	**Standard Deviation**	**Median Time**	**Mean Time**	**Standard Deviation**
Insertionsort	10	54.6	99.4	46	67.7	69.0
Quicksort	6	49.2	102.7	40.5	70.3	73.9

interaction log data to analyze student use of OpenDSA. We plotted the timelines of interactions within OpenDSA and added class events (homework due date, exams date) to the graph.

Logs analysis showed that students used OpenDSA to review for exams, thus indicating the usefulness of OpenDSA as a study aid. Figs. 8.3–8.5 present timelines showing exercise completions. Clusters of interactions appear at homework due dates as expected. But there are also spikes in the number of interactions immediately prior to the exams.

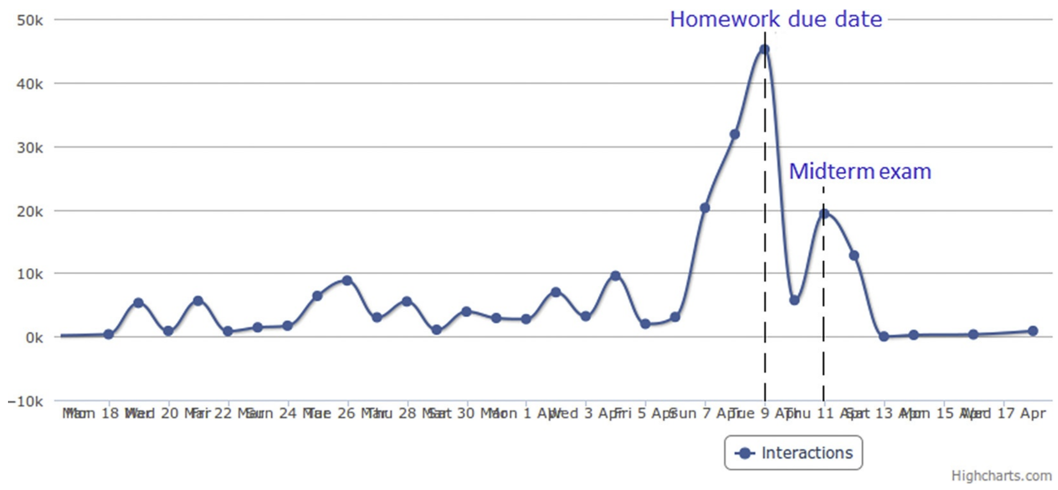


FIGURE 8.3

Timeline of interactions: CS3 at Virginia Tech, Spring 2013 (sorting and hashing chapters).

2.2 STUDENT PERCEPTIONS OF OpenDSA

Some of the case studies from the previous section are rather negative, in that they paint a picture of students avoiding reading content and skipping at least some exercise types whenever possible. However, students routinely offer positive feedback about the system and experience as a whole. Whenever surveyed, students always give a positive value to OpenDSA. In this section, we report on some specific survey feedback from students. We then provide one further case study of student behavior, related to voluntary use of OpenDSA as a study aid and its relationship to grades.

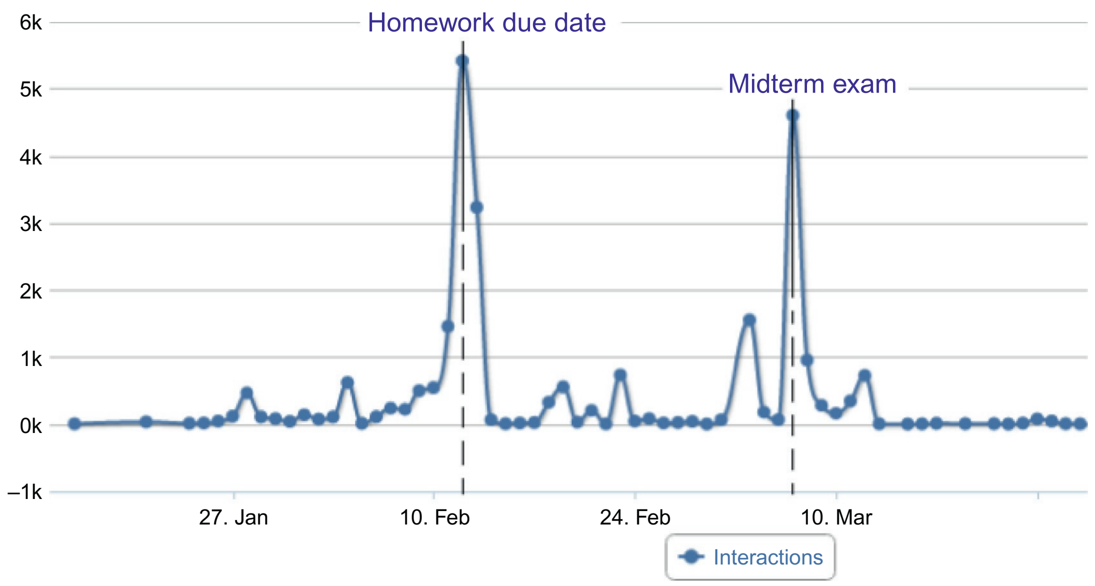


FIGURE 8.4

Timeline of interactions: CS2 at New York University, Spring 2014.

2.2.1 Student ranking of OpenDSA exercises

We investigated how students ranked OpenDSA exercises in terms of perceived learning benefits, specifically as related to other course elements. Participants comprised 54 students enrolled in a CS2 course at Virginia Tech. This is a programming-intensive class with 2 h of programming labs each week. OpenDSA exercises were used as weekly mandatory homework assignments. At the end of the semester, students answered survey questions to report about their experience using OpenDSA.

Students were asked: "This class included these elements: (1) lectures, (2) OpenDSA Exercises, (3) textbook (paper), (4) programming labs, (5) reading quizzes, and (6) interacting with instructor. Please rank these in order from the one that gave you the most learning gains to the one that gave you the least." Thus they ranked class items from 1 — most useful to 6 — least useful in regards to their learning gains in the course.

Students on average ranked OpenDSA exercises second, with an average score of 2.48. Note that OpenDSA materials were used for only part of the course. The main course textbook was a traditional paper textbook. Programming labs were ranked first with an average score of 1.42, the course (paper) textbook was ranked third (average score = 3.77). Lectures, interacting with instructor, and reading quizzes were ranked fourth, fifth, and sixth with average scores 4.13, 4.19, and 4.57, respectively.

A Friedman test was run to identify if the difference between perceived leaning benefits was statistically significant. Because the Friedman test is only applicable to complete block designs, we discarded incomplete blocks (ie, data from students who did not rank all six class elements). Thus 11 incomplete evaluations were discarded resulting in 43 evaluations used in our analysis. The Friedman test revealed a significant difference between the ranking of class items ($X^2(5) = 101.5$, $p < 0.01$). A post hoc Wilcoxon test with Bonferroni correction identified three clusters of items.

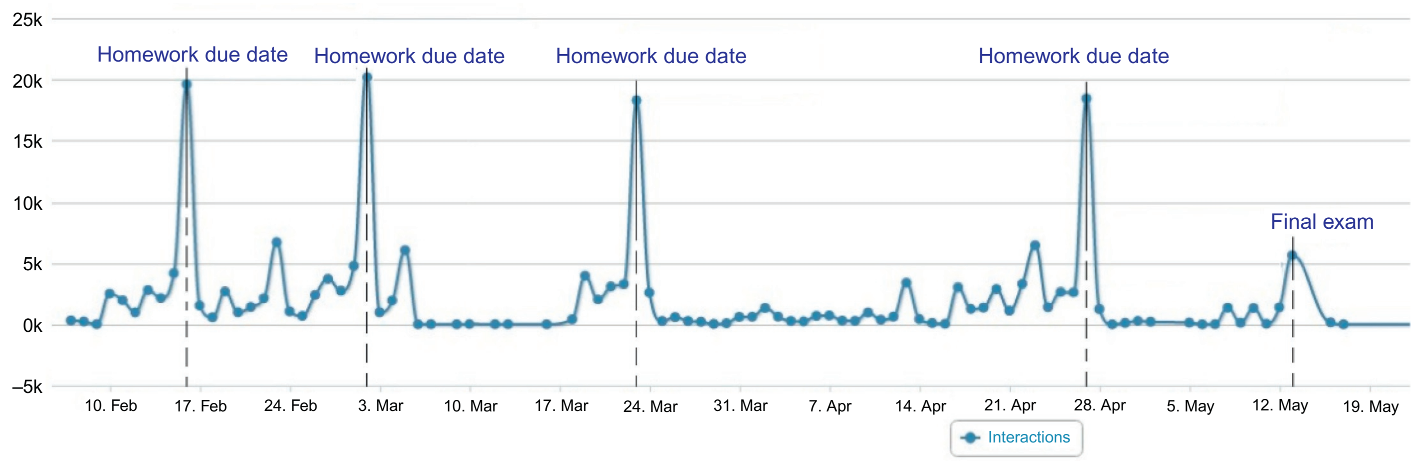


FIGURE 8.5

Timeline of interactions: CS2 at Virginia Tech, Spring 2014.

The highest-ranking cluster consisted of the programming labs. The second highest-ranking cluster consisted of the OpenDSA exercises. The remaining, lowest-ranking cluster included all other items: the paper textbook, lectures, instructor, and reading quizzes. Students attributed most of the learning gains in the course to programming labs first, then to OpenDSA exercises. The other class items were perceived to have a significantly lower impact in learning benefits.

2.2.2 Student self-efficacy regarding learning DSA material

OpenDSA's assessment activities provide the learner with many mastery-based experiences, thus improving students' abilities to learn course material. We asked students to rate their ability to succeed in the course under various scenarios of use for OpenDSA. This information came from a survey carried out on two courses, a CS2 course during Spring 2014 at Virginia Tech, and a CS3 course during Fall 2014 also at Virginia Tech. In both groups, OpenDSA exercises were used as mandatory assignments. OpenDSA was the sole class textbook in the CS3 course.

At the end of the semester, students answered survey questions asking them to rate their ability to learn course material under several (hypothetical) scenarios. We asked students in the CS2 course to use a Likert scale of 1 (low confidence) to 5 (high confidence) to rate their confidence in their ability to learn the course materials under the following conditions:

- Scenario 1: I could succeed in learning the course material if the class was only taught using a standard textbook with Moodle quizzes.
- Scenario 2: I could succeed in learning the course material using OpenDSA on my own if I was required to do the OpenDSA exercises for a grade.
- Scenario 3: I could succeed in learning the course material using OpenDSA on my own with no graded exercises.
- Scenario 4: I could succeed in learning the course material using OpenDSA if the instructor goes over OpenDSA's content during a lecture but I am NOT required to do the exercises for a grade.
- Scenario 5: I could succeed in learning the course material using OpenDSA if the instructor goes over OpenDSA's content during a lecture and I AM required to do the OpenDSA exercises for a grade.

Students enrolled in the CS3 course were asked to answer the same questions, but Scenario 1 was removed since OpenDSA was the textbook of the course and the course did not include Moodle quizzes.

Our analysis of student responses showed that students ranked scenarios involving using OpenDSA during lecture and for graded assignment higher. In CS2, out of the 49 responses that we collected, 38% gave Scenario 5 the highest score of 5, and 69% of the responders ranked their confidence in succeeding in the class under Scenario 5 with a score of 4 or 5. Scenarios involving mandatory OpenDSA exercises (for grading) received higher ability to learn course content scores, thus showing that students have a preference for that type of use of OpenDSA. The above results also corroborate our hypothesis that the mastery-based experience provided by OpenDSA exercises will help student ability to learn course materials. We found similar results from the CS3 responses.

A Kruskal Wallis test revealed for scenarios of use a significant effect on predicted ability to learn course materials for both CS2 ($X^2(4)=51.9, p<0.01$) and CS3 ($X^2(3)=48.5, p<0.01$). A post hoc test using Wilcoxon tests with Bonferroni correction identified two clusters for CS2 data: Scenario 5 versus the other four scenarios. In other words, Scenario 5 was significantly preferred over all other responses, among which the ordering was not significant.

A post hoc test using Wilcoxon tests with Bonferroni correction identified three clusters for CS3 data. Again, Scenario 5 was most preferred, and the difference was significant. Scenario 2 was the clear second choice. Scenarios 3 and 4 were the least preferred, but their relative order was not significant.

To summarize, the results show that students believe that a class format with lecture content on OpenDSA material, combined with mandatory assignments, will be most effective.

2.2.3 Effects of OpenDSA exercises on student performance

Given the holistic nature of many courses, it can be difficult to extract the effects of one aspect of a course on performance outcomes. Students get information about course content from many sources. We have searched for potential relationships between use of OpenDSA and student performance on exams. Some preliminary results do show a relationship between OpenDSA exercises worked and student performance. Unfortunately, so far we only have sufficient data to show correlation, not causation.

We categorized OpenDSA exercises into three types. Multiple choice, T/F, and fill-in-the-blank questions implemented using the Khan Academy framework are identified as "KA." Algorithm proficiency exercises implemented using the Khan Academy framework are generally short and scored as correct or incorrect; these are identified as "KAV." Full algorithm proficiency exercises with multiple steps are identified as "PE." For each exercise type (KA, KAV, PE), we computed the mean number of exercises completed by the subjects before Midterm 1 and before Midterm 2 in a CS3 course at Virginia Tech during Fall 2013.

The book instance used in our analysis had a total of 95 required exercises, of which 36 were KA simple questions sets (each with several multiple choice or T/F questions), 30 were KA "mini-proficiency" exercises (KAV), 26 were full proficiency exercises (PE), and 3 were other interactive activities. Before Midterm 1, students were required to solve a total of 15 KA simple question sets, 12 KAV exercises, and 6 PE exercises. Between Midterm 1 and 2, they had to complete 19 KA simple question sets, 18 KAV exercises, and 20 PE exercises.

We wanted to see if there would be a difference between the students who correctly completed many exercises "postproficiency" (that is, repeated exercises after receiving credit as a study aid) and those who did not. We found a low correlation between the number of correct exercises (regardless of the type) completed by the students and exam scores. A negative correlation observed between the total number of KA simple questions attempted might be an indication that some students were just using a trial-and-error strategy when solving those exercises, resulting in a lot of incorrect answer submissions (and thus driving up the total number of attempts).

We computed the average number of correct exercises performed by students grouped by score quartile as follows: $q0$ represents students who performed below the 25th percentile; $q1$ represents students with a score between the 25th and the 50th percentile; $q2$ represents students with a score between the 50th and the 75th percentile; and $q3$ represents students with a score above the 75th percentile. In order to compare OpenDSA use across quartiles, we computed the mean number of completed exercises for each quartile. Tables 8.3 and 8.4 show the statistics of the different groups. As shown in Fig. 8.6, *students with higher exam scores tend to complete more exercises correctly than those with lower exam scores*. Since nearly all students complete enough exercises correctly as needed to get full credit for the OpenDSA assignments, what this means is that there is a correlation between repeating more exercises (correctly) as a study aid, and getting a better score on the exam.

Table 8.3 Average Correct Exercises Completed, Grouped by Midterm 1 Score

Groups	Correct.KA	Correct.KAV	Correct.PE
$q0$ ($N=35$)	15.14 (SD = 8.04)	9.17 (SD = 4.85)	9.37 (SD = 7.36)
$q1$ ($N=37$)	19.05 (SD = 9.82)	11.43 (SD = 4.11)	11.75 (SD = 8.36)
$q2$ ($N=32$)	19.43 (SD = 5.89)	11.53 (SD = 3.91)	13.93 (SD = 14.43)
$q3$ ($N=31$)	23.64 (SD = 9.15)	12.70 (SD = 5.71)	13.87 (SD = 12.18)

Table 8.4 Average Correct Exercises Completed, Table Grouped by Midterm 2 Score

Groups	Correct.KA	Correct.KAV	Correct.PE
$q0$ ($N=38$)	15.42 (SD = 5.91)	14.55 (SD = 6)	26.89 (SD = 12.76)
$q1$ ($N=31$)	16.83 (SD = 10.11)	14.22 (SD = 6.57)	26.45 (SD = 13.08)
$q2$ ($N=33$)	18.09 (SD = 7.27)	16.33 (SD = 4.55)	31.24 (SD = 14.93)
$q3$ ($N=33$)	24.60 (SD = 12.09)	18.78 (SD = 5.75)	34.03 (SD = 11.85)

The difference in the average number of correct exercises between student groups was significant at the $p < 0.05$ level for KA simple questions and KAV exercises for Midterm 1 and 2. However, the difference was not statistically significant for PE exercises. The difference was still not significant when merging the number of correct KAV exercises with the number of correct PE exercises.

We can say that higher scores on written tests are correlated with a high number of correct exercises (beyond the minimum level required to receive full homework credit). These findings can be used to detect struggling students, since they are less likely to use an exercise for study purposes. In addition, students with the lowest grades have the lowest "number of correct exercises to number of exercises attempted" ratio. In other words, they appear to be using some suboptimal behavior such as guessing or viewing hints in an attempt to build a catalog of correct answers, rather than actively using their knowledge to correctly answer the question.

3 GAMIFICATION: HOW FEEDBACK MOTIVATES STUDENT BEHAVIOR

3.1 INTRODUCTION

There can be no doubt that many people are motivated by indications of progress and closure. For example, witness the feeling of satisfaction that many people get from the simple act of crossing a task off of a "to do" list. We have implemented several types of feedback to users regarding completion of an exercise, and progress toward completion of modules and chapters. This feedback can be considered a form of "gamification." Gamification is the use of game design elements in nongame contexts (Deterding et al., 2011).

Here we are interested in exploring how visual elements (indicators) and other techniques borrowed from game design influence student behavior when using OpenDSA. In this section, we first enumerate the gamification features within OpenDSA, and report on a survey of student opinions on these

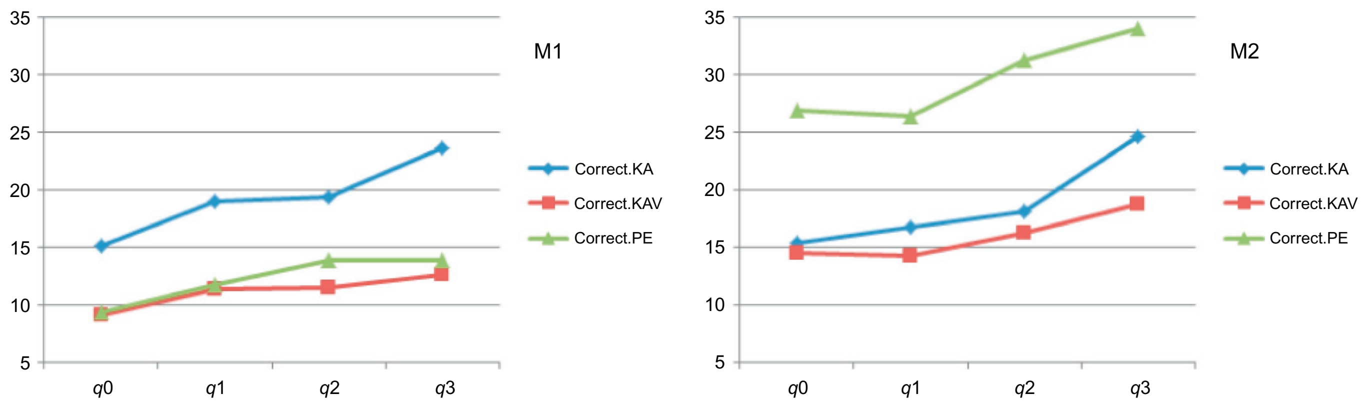


FIGURE 8.6

Midterm 1 and 2 exercises means per quartile.

features. We then report on indications from our log data about how the presence of visual indicators influences student interactions with OpenDSA. Further details on some aspects of this section can be found in Fouh (2015).

3.2 RELATED WORK

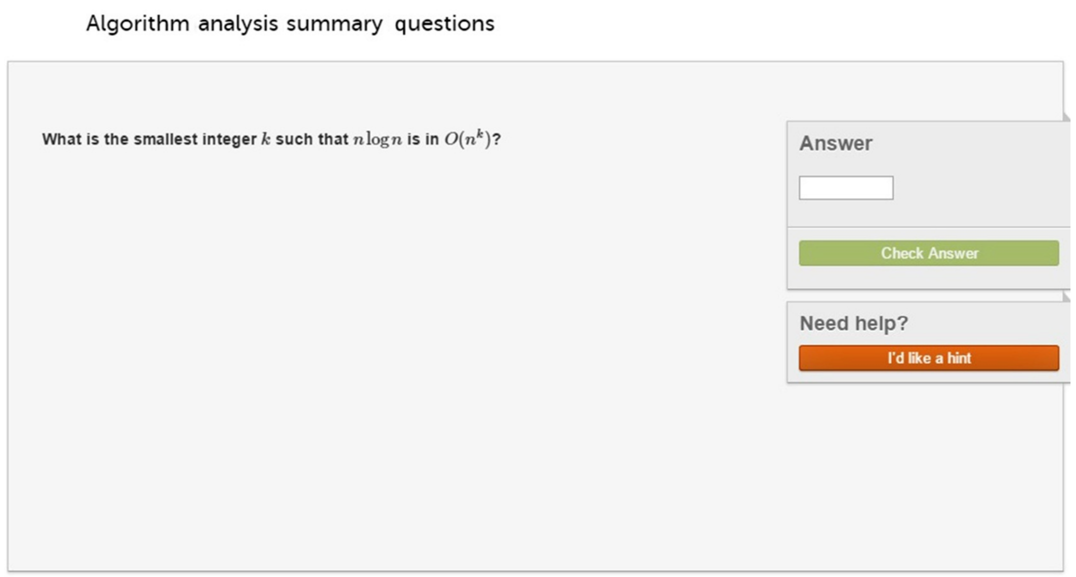
One element of gamification is the use of motivational affordances in order to encourage particular behaviors (Hamari et al., 2014). Zhang (2008) identified five main motivational sources and argued that games and learning system design should be influenced by cognitive (achievements and accomplishments) motivations and needs. Elements present in games that have been mainly used in educational systems design include scores, leaderboards, achievements/badges, and level (Hamari et al., 2014; Deterding et al., 2011). Some elements like scores have always been part of educational systems, and seem to be motivating. On the other hand, there is little evidence in the literature to conclude that there is a positive impact from badges on student engagement or effectiveness (Falkner and Falkner, 2014). Despite this, badges are increasingly being used in computer science education systems. Haaranen et al. (2014) awarded badges to students for completing exercises in a data structures course, where badges were displayed within the learning management system. They reported a mixed impact on student learning behavior. A leaderboard and badges were used in an introductory media computation course, and are credited with contributing to positive experiences for students and high overall engagement (Latulipe et al., 2015).

3.3 OpenDSA GAMIFICATION ELEMENTS

OpenDSA aims to provide students with visual indicators upon successful completion of an interactive activity. Such indicators are used in conjunction with a traditional points system for homework assignments. Points are earned when an exercise has been solved correctly. OpenDSA uses a mastery-based approach, meaning that students can attempt an exercise as many times as desired until completion credit has been given (and the exercise can continue to be used after that, perhaps as a study aid). While some “exercises” are merely static multiple choice or T/F questions, many generate a new problem instance each time, making them repeatable in a meaningful way for additional practice. OpenDSA exercises are fundamentally implemented in two ways. Most exercises use the Khan Academy exercises framework (KA exercises). Such exercises provide students with hints to help them answer a question. Fig. 8.7 shows an example of a Khan Academy exercise in which the student can click on the “I’d like a hint” button to get help. Once a hint is used, that problem instance is not graded.

Other exercises are referred to as “JSAV-Proficiency exercises” (PE exercises). These are always algorithm simulations, where the student will manipulate a data structure such as an array or tree through a series of steps to reproduce the behavior of some algorithm. The initial state of the data structure (such as the specific values in an array for a sorting algorithm, or the order of the inputs for insertion into a tree) specifies a new problem instance. For each such instance, a visualization for a model solution is automatically generated. Fig. 8.8 shows an example of a PE with the model solution visible. Students can chose to see the model answer without (or before) solving the exercise. Once a model answer has been displayed, that problem instance will not be graded. So to get credit for the exercise, the student will need to request a new problem instance to work on.

Now we describe the visual indicators displayed when an interactive activity is completed.

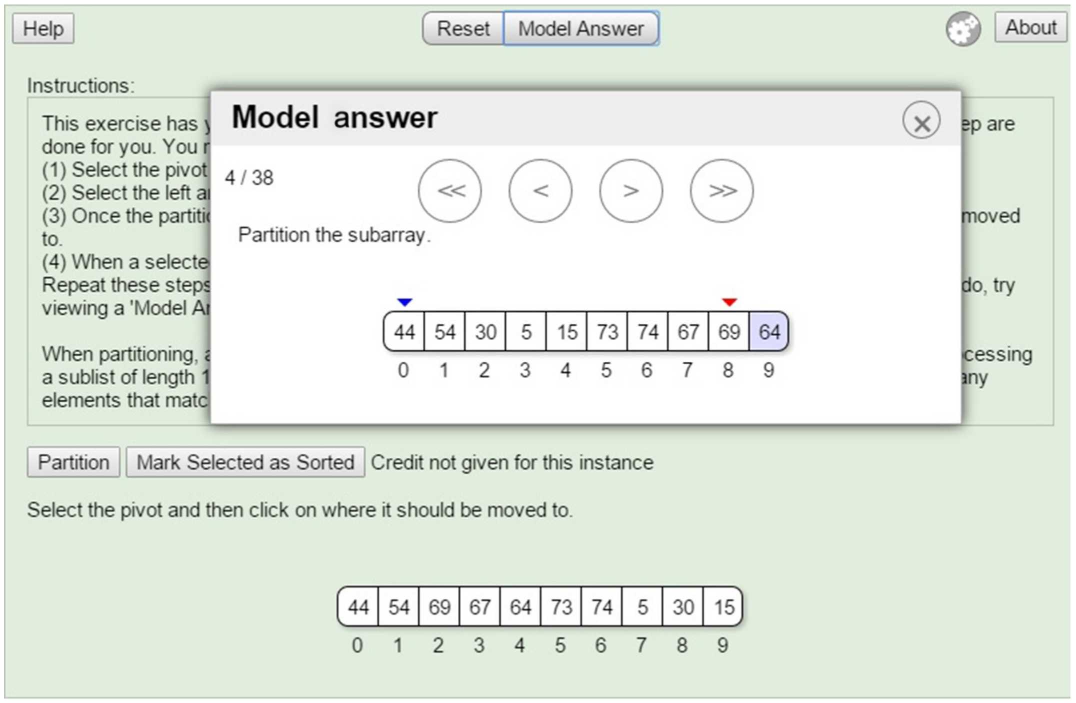


Algorithm analysis summary questions

What is the smallest integer k such that $n \log n$ is in $O(n^k)$?

FIGURE 8.7

Khan Academy exercise.



Instructions:

This exercise has y ... ep are done for you. You r
(1) Select the pivot
(2) Select the left a
(3) Once the partiti ... moved to.
(4) When a selecte
Repeat these steps ... do, try viewing a 'Model A

When partitioning, a ... cessing a sublist of length 1 ... any elements that matc

Model answer

4 / 38

Partition the subarray.

44	54	30	5	15	73	74	67	69	64
0	1	2	3	4	5	6	7	8	9

Credit not given for this instance

Select the pivot and then click on where it should be moved to.

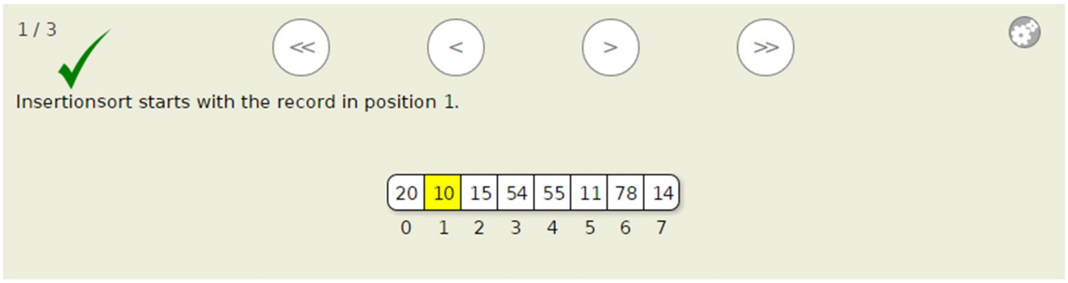
44	54	69	67	64	73	74	5	30	15
0	1	2	3	4	5	6	7	8	9

FIGURE 8.8

Proficiency exercise.

3.3.1 Proficiency for embedded slideshows

OpenDSA visualizations often come in the form of embedded slideshows, which generally are relatively short presentations of a specific example. Users are given an indication when the slideshow has been completed (all slides viewed), shown by the appearance of a green check mark on the left side of the slideshow container (see Fig. 8.9).



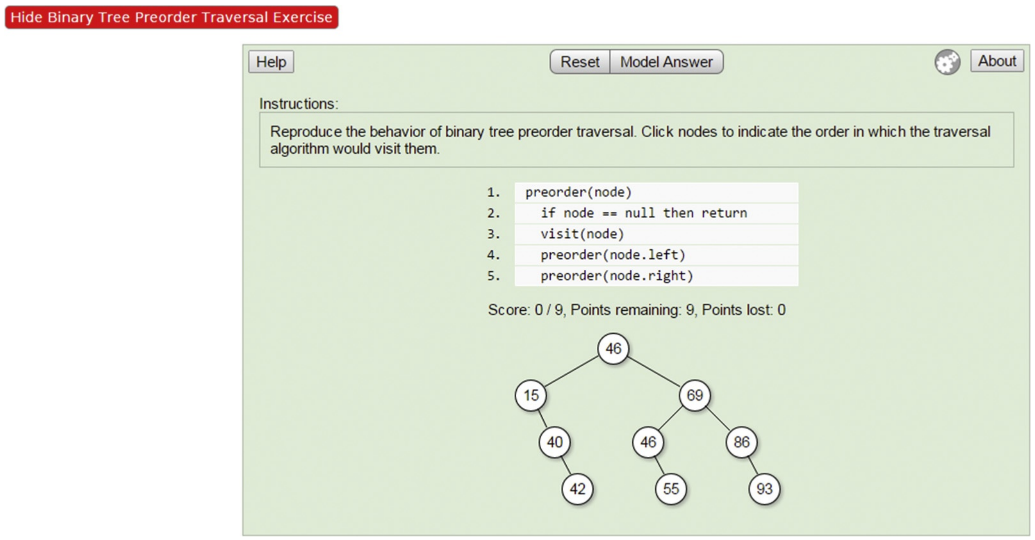
Insertionsort starts with the record in position 1.

FIGURE 8.9

Insertionsort mini-slideshow with *green* (*dark gray* in the print versions) check mark.

3.3.2 Proficiency for embedded exercises

Completion of an exercise is indicated by the color of the button used to control whether the exercise is to be shown or hidden. A red button indicates that the user is not yet given credit for being proficient, as shown in Fig. 8.10. Green indicates that the user has received proficiency credit (see Fig. 8.11).



Instructions:

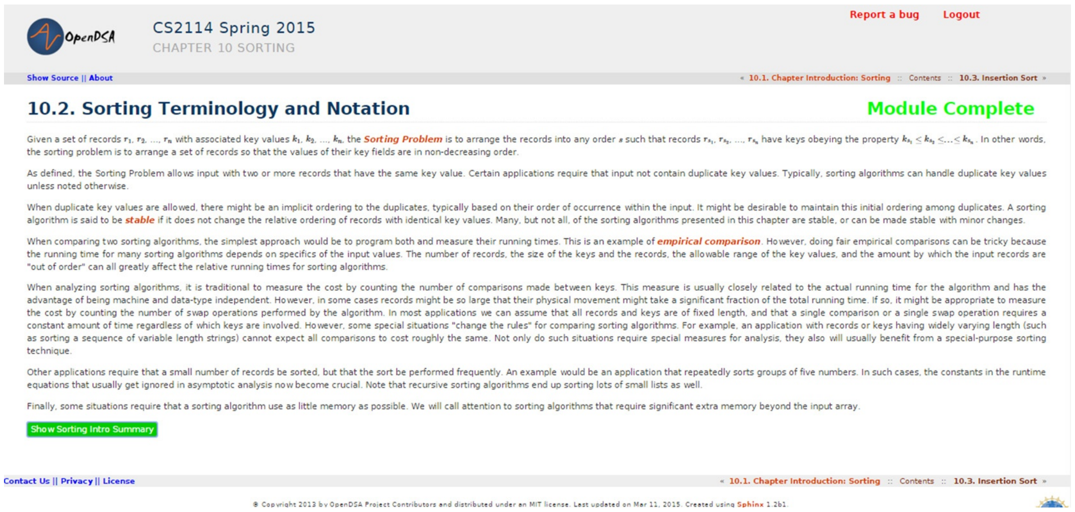
Reproduce the behavior of binary tree preorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

```
preorder(node)
  if node == null then return
  visit(node)
  preorder(node.left)
  preorder(node.right)
```

Score: 0 / 9, Points remaining: 9, Points lost: 0

FIGURE 8.10

Tree preorder traversal exercise with *red* (*dark gray* in the print versions) show/hide button.


OpenDSA
Report a bug
Logout
Show Source || About
« 10.1. Chapter Introduction: Sorting :: Contents :: 10.3. Insertion Sort »
Show Sorting Intro Summary
Contact Us || Privacy || License
« 10.1. Chapter Introduction: Sorting :: Contents :: 10.3. Insertion Sort »

CS2114 Spring 2015
CHAPTER 10 SORTING

10.2. Sorting Terminology and Notation

Module Complete

Given a set of records $r_1, r_2, \ldots, r_n$ with associated key values $k_1, k_2, \ldots, k_n$, the ***Sorting Problem*** is to arrange the records into any order s such that records $r_{s_1}, r_{s_2}, \ldots, r_{s_n}$ have keys obeying the property $k_{s_1} \le k_{s_2} \le \ldots \le k_{s_n}$. In other words, the sorting problem is to arrange a set of records so that the values of their key fields are in non-decreasing order.

As defined, the Sorting Problem allows input with two or more records that have the same key value. Certain applications require that input not contain duplicate key values. Typically, sorting algorithms can handle duplicate key values unless noted otherwise.

When duplicate key values are allowed, there might be an implicit ordering to the duplicates, typically based on their order of occurrence within the input. It might be desirable to maintain this initial ordering among duplicates. A sorting algorithm is said to be ***stable*** if it does not change the relative ordering of records with identical key values. Many, but not all, of the sorting algorithms presented in this chapter are stable, or can be made stable with minor changes.

When comparing two sorting algorithms, the simplest approach would be to program both and measure their running times. This is an example of ***empirical comparison***. However, doing fair empirical comparisons can be tricky because the running time for many sorting algorithms depends on specifics of the input values. The number of records, the size of the keys and the records, the allowable range of the key values, and the amount by which the input records are "out of order" can all greatly affect the relative running times for sorting algorithms.

When analyzing sorting algorithms, it is traditional to measure the cost by counting the number of comparisons made between keys. This measure is usually closely related to the actual running time for the algorithm and has the advantage of being machine and data-type independent. However, in some cases records might be so large that their physical movement might take a significant fraction of the total running time. If so, it might be appropriate to measure the cost by counting the number of swap operations performed by the algorithm. In most applications we can assume that all records and keys are of fixed length, and that a single comparison or a single swap operation requires a constant amount of time regardless of which keys are involved. However, some special situations "change the rules" for comparing sorting algorithms. For example, an application with records or keys having widely varying length (such as sorting a sequence of variable length strings) cannot expect all comparisons to cost roughly the same. Not only do such situations require special measures for analysis, they also will usually benefit from a special-purpose sorting technique.

Other applications require that a small number of records be sorted, but that the sort be performed frequently. An example would be an application that repeatedly sorts groups of five numbers. In such cases, the constants in the runtime equations that usually get ignored in asymptotic analysis now become crucial. Note that recursive sorting algorithms end up sorting lots of small lists as well.

Finally, some situations require that a sorting algorithm use as little memory as possible. We will call attention to sorting algorithms that require significant extra memory beyond the input array.

© Copyright 2013 by OpenDSA Project Contributors and distributed under an MIT license. Last updated on Mar 11, 2015. Created using Sphinx 1.2b1.

FIGURE 8.11

Module page with proficiency indicator for module and exercise.

3.3.3 Proficiency for modules

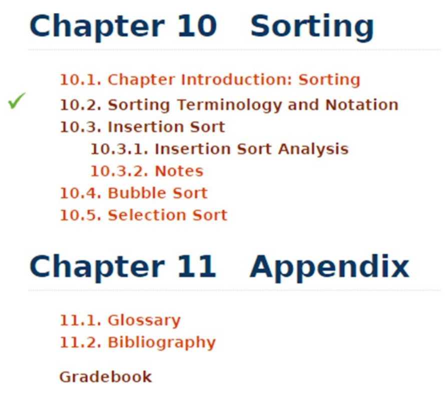
Modules in OpenDSA cover a specific topic or part of a topic. Operationally, a module corresponds to a single HTML page within a book instance. Each module typically includes one or more slideshows and one or more interactive exercises. Module proficiency occurs when a user obtains proficiency for all the required exercises and completes viewing all of the slideshows in a module. Proficiency is signaled to the user by the words "Module Complete" that appear in green at the top of the module page (see Fig. 8.11). A given book instance can optionally be configured to show "Module Complete" automatically when the module is viewed, for modules with no exercises or visualizations. At this point, we have not studied whether it is desirable or not to indicate "Module Complete" signals for such modules. But anecdotal evidence suggests that students prefer this.

3.3.4 Table of contents indicators

A small green check mark appears next to a module title in the Table of Contents when that the module has been completed (see Fig. 8.12).

3.3.5 OpenDSA gradebook

Perhaps the most important notification information that students receive comes through the *gradebook page*. This page shows score status for all exercises and modules. Those for which the user is proficient are highlighted in green. The gradebook is a hierarchical display that allows users to toggle between showing or hiding portions of the gradebook. Fig. 8.13 shows a gradebook page with two modules expanded, one down to the exercise level. Completed exercises within the expanded display for the module are highlighted in green.



Chapter 10 Sorting

10.1. Chapter Introduction: Sorting
✓ 10.2. Sorting Terminology and Notation
10.3. Insertion Sort
10.3.1. Insertion Sort Analysis
10.3.2. Notes
10.4. Bubble Sort
10.5. Selection Sort

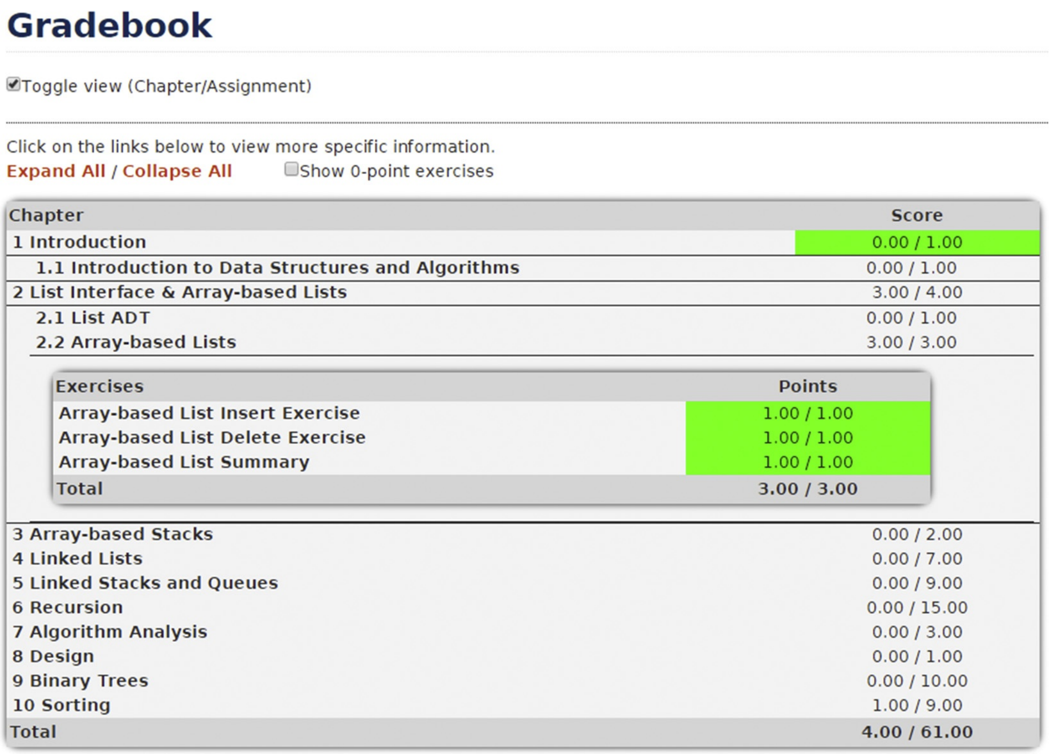
Chapter 11 Appendix

11.1. Glossary
11.2. Bibliography

Gradebook

FIGURE 8.12

Section of table of content with module proficiency indicator.



Gradebook

☑Toggle view (Chapter/Assignment)

Click on the links below to view more specific information.
Expand All / Collapse All ☐Show 0-point exercises

Chapter	Score
1 Introduction	0.00 / 1.00
1.1 Introduction to Data Structures and Algorithms	0.00 / 1.00
2 List Interface & Array-based Lists	3.00 / 4.00
2.1 List ADT	0.00 / 1.00
2.2 Array-based Lists	3.00 / 3.00

Exercises	Points
Array-based List Insert Exercise	1.00 / 1.00
Array-based List Delete Exercise	1.00 / 1.00
Array-based List Summary	1.00 / 1.00
Total	3.00 / 3.00

Chapter	Score
3 Array-based Stacks	0.00 / 2.00
4 Linked Lists	0.00 / 7.00
5 Linked Stacks and Queues	0.00 / 9.00
6 Recursion	0.00 / 15.00
7 Algorithm Analysis	0.00 / 3.00
8 Design	0.00 / 1.00
9 Binary Trees	0.00 / 10.00
10 Sorting	1.00 / 9.00
Total	4.00 / 61.00

FIGURE 8.13

Gradebook page.

3.4 STUDENT PERCEPTIONS OF OpenDSA GAMIFICATION ELEMENTS

We conducted a survey of OpenDSA users in order to understand what they think of the gamification elements. Participants were 143 students enrolled in a CS3 course at Virginia Tech during Fall 2014, from whom we collected 83 survey responses. OpenDSA was used as the main textbook, and students had until the end of the semester to complete all of the OpenDSA exercises. Completion of OpenDSA exercises accounted for 20% of the course final grade.

The survey asked the following questions:

1. OpenDSA includes a number of features to help you track which exercises and modules you have completed. These include green checkmarks by the slideshows, green text saying "Module Complete" at the top of the page, green marks in the table of contents, and green bars in the student gradebook. On a scale of 1–5, with 1 meaning not important and 5 meaning very important, how important are these signals for completing material to you?
2. Does the existence of the green checkmarks and other signals of completion of material affect your behavior in any way? If so, how?

We used a qualitative method of content analysis when examining student responses to Question 2. We used interaction log data to analyze and understand how visual indicators influenced student use of OpenDSA.

3.4.1 Student perceptions on visual feedback indicators

To the question "how important are these (OpenDSA visual) signals for completing material to you," respondents gave an average score of 4.01 on a five-point Likert scale. Fig. 8.14 shows the response distribution. A majority of students (75%) indicated that visual indicators were important when using OpenDSA to study course content.

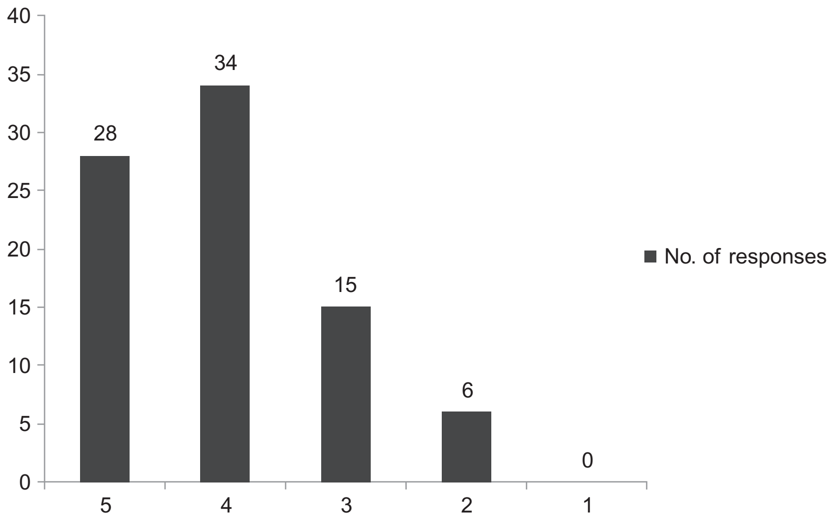


FIGURE 8.14

Responses distribution for each rank.

3.4.2 Effects of visual feedback indicators on student behavior

We used codes to label ideas expressed in each response to the question "Does the existence of the green checkmarks and other signals of completion of material affect your behavior in any way? If so, how?" The codes attempt to capture how the visual indicators affected students' learning behavior. We derived the codes from students' responses, and grouped responses with the same code together. Table 8.5 summarizes the results. In this table, we show the average score from the first question for the students in each code group.

Out of 80 responses collected, 65% reported that visual elements had an effect on their behavior while 35% reported no impact. Thirty-one percent reported using visual indicators to keep track their progress. Similarly, 15% indicated using the visual elements to track work remaining (labeled as "Goal tracking"). Students can easily track their progress (or work that remains) by looking at the greened items in the gradebook or by the presence of green checkmarks in front of sections in the table of contents. Three students acknowledged actively seeking the green checkmarks while working. One student wrote "Yes (completion signals affected my behavior), I try to get checkmarks on everything."

Among the students who reported using visual indicators to keep track of the goal (amount of work left), 10 indicated that the visual indicators *motivated them to do more work*. A student wrote "Yeah it's fun to see it all get completed makes me want to do it more," while another one wrote "yes, made me more willing to complete everything."

Nine students reported that *visual indicators reinforced their learning*. They indicated liking them and being happy to see them. Examples of comments in that category include "Yes, I think it's rewarding and make me feel better," "They make me feel accomplished so I work harder," and "I can be sure I have completed the graded material." Students in this category on average gave the highest importance score to visual indicators (mean = 4.77).

Among the students who reported using visual indicators to track their accomplishments, eight indicated that *they stopped working on items as soon as they are marked as completed*, *until they needed to study for a test*. A student noted that "Usually after it's checked I don't go back unless I have a test or need it for a project." Another wrote "(I) don't continue practicing them (checked items) until test review time." This trend needs to be taken into consideration when deciding a proficiency threshold for an exercise. A low proficiency threshold might not provide the students with enough practice. Some students reported that visual indicators are a better way to track progress than a points system. For example, a student wrote "Kind of gave concrete evidence on completion as opposed to 'What is my grade now'."

Table 8.5 Student Response Summary

Code	Value	Frequency	Percentage	Avg. Score	Agg. Score
1	Accomplishment tracking	25	31	4.12	4.33
2	Goal tracking	12	15	4.08	
3	Accomplishment/goal tracking	2	3	4.5	
4	Verification grade saved on server	2	3	4	
5	Reinforcement	9	11	4.77	
6	Reinforcement/goal tracking	2	3	4.5	
7	No impact	28	35	3.57	3.52
	Total	80	100		

A Wilcoxon rank sum test revealed they ranked visual indicators as more important when they reported that these indicators affected their behavior ($W(79) = 995$, $p < 0.01$). This is to be expected, but it also rules out a situation where people feel compelled to get all of the marks since they are there, but would have preferred that they not be visible.

3.5 IMPACT OF PROFICIENCY INDICATORS ON STUDENT TIME SPENT ON SLIDESHOWS

Visual proficiency indicators appear upon correct completion of an interactive element and remain visible thereafter. Therefore, it is unlikely that visual indicators will motivate students to complete the same exercise or slideshow after they gain proficiency. Students receive points only for exercises. However, the “Module Complete” message and the green checkmarks in the table of contents only appear if they have gone through all the visualizations (on top of the exercises).

Previous studies (Breakiron, 2013; Fouh et al., 2014a) have shown that many students go straight to the exercises upon loading a module and they rush through slideshows (during their first viewing). We investigated the time spent by students on slideshows before proficiency (first viewing) and post proficiency (second viewing) to see if students behavior changed once they got the green checkmark for completing a slideshow.

The majority of students viewed most slideshows at least twice. Ninety-eight percent of the students completed more than 50% of the slideshows at least twice, with 20 (out of 135) students completing all the slideshows at least two times. Fig. 8.15 shows the distribution of students’ subsequent slideshow completion (reattempt ratio). The mean fraction of slideshows that were viewed multiple times was 80%.

Student’s behavior during their second viewing should not be influenced by the check mark (since the visual status cannot change at this point). In Fig. 8.16, for each slideshow we have plotted the mean time

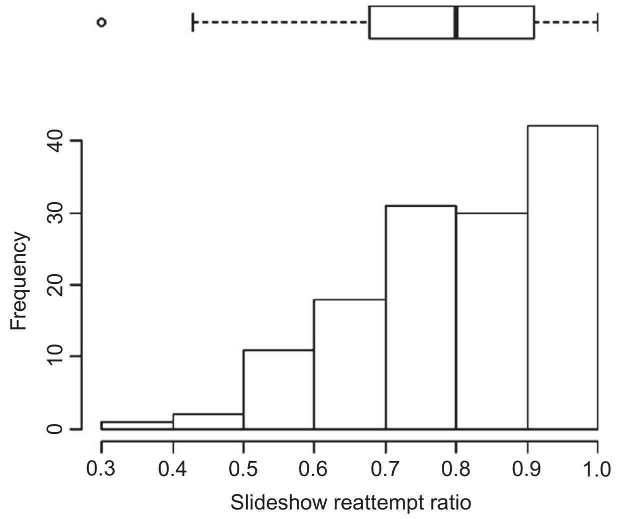


FIGURE 8.15

Multiple viewings of slideshows. Each bar in the histogram shows the fraction of students who revisited that fraction of slideshows.

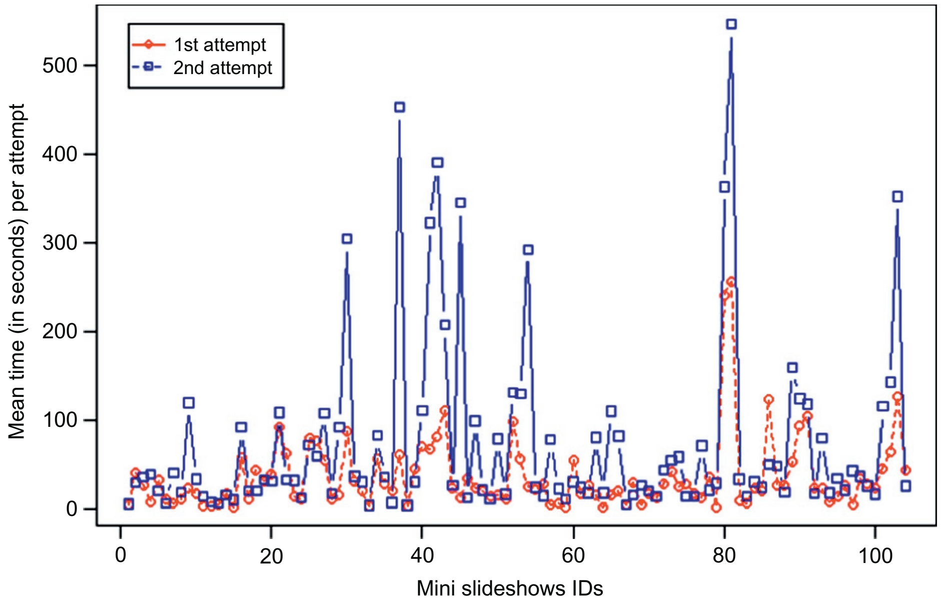


FIGURE 8.16

Mean time per slideshow per viewing.

spent by students during the first and second viewings. (There were not enough third viewings to analyze.) Students spent a mean of 36 s and a median of 24 s on each slideshow during their first viewing. They spent a mean of 73 s and a median of 31 s on each slideshow on their second viewing. A Wilcoxon rank sum test showed that the increase in time spent on each slideshow between the first and the second viewings was statistically significant ($W(1125)=995$, $p<0.01$). Since rushing through slideshows is more pronounced on the first viewing, it seems that preproficiency work is influenced by the green checkmark, and postproficiency work is more influenced by learning, especially to review for exams.

3.6 STUDENTS USE OF THE GRADEBOOK

There were 171 HTML pages in the book instance used by students enrolled in the CS3 course offering at Virginia Tech in Fall 2014. We examined which pages were most visited. The top four pages, in order, were as follows:

1. The book's Table of Contents page was the most visited (with 31,496 hits).
2. The second most visited page presented examples of hashing functions (with 12,838 hits).
3. The gradebook page was the third most visited page (with 9905 hits).
4. The page covering Quicksort was the fourth most visited page (with 6155 hits).

As shown in Fig. 8.18, students enrolled in the course visited the gradebook page on average 61 times during the course of the semester (117 days). The median number of gradebook visits was 25. Eight percent (11 out of 135) of the students enrolled in the course never looked at the gradebook. The average OpenDSA exercise completion rate for student who never visited the gradebook was 91% while the average completion rate for the rest of the class was 92%.

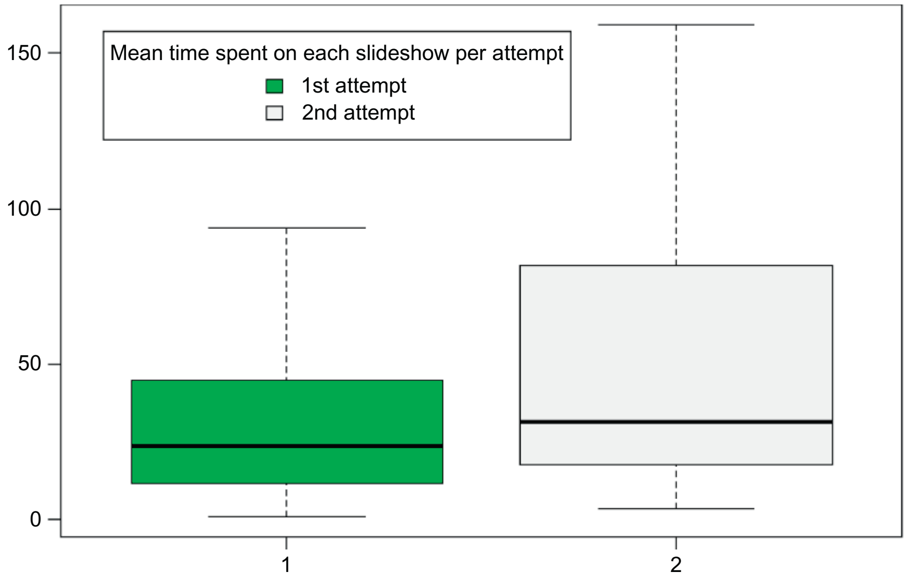


FIGURE 8.17

Distribution of time per slideshow, by viewing. In each case, we see a boxplot with median line.

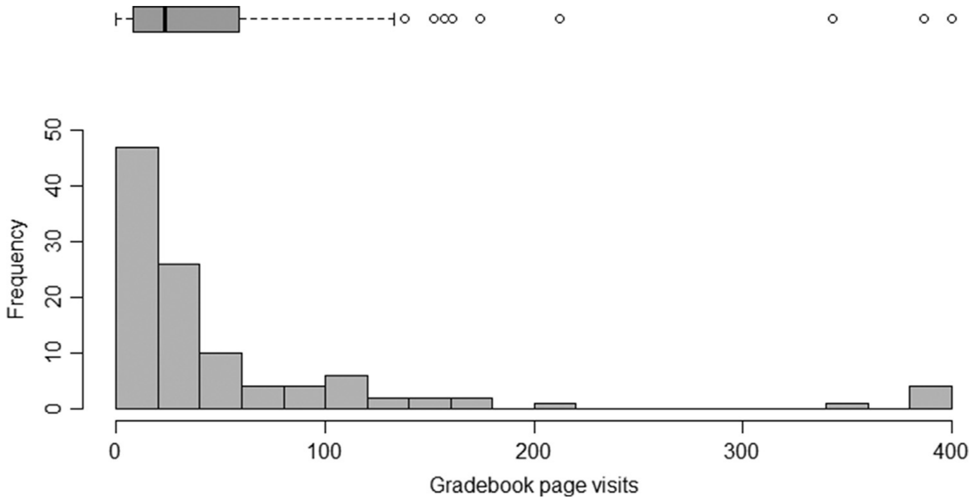


FIGURE 8.18

Distribution for number of times a given student visited the gradebook during the semester. The right-most bin includes all students who viewed the gradebook 400 times or more.

There was a strong correlation ($r=0.9$) between the overall daily total number of exercises attempted and daily total gradebook page hits as shown in Fig. 8.19. This indicates that students used the gradebook to check their progress while working on exercises. On any given day, only a mean of 40% (SD = 23%, median = 35%) of the students who attempted an exercise viewed the gradebook that same day.

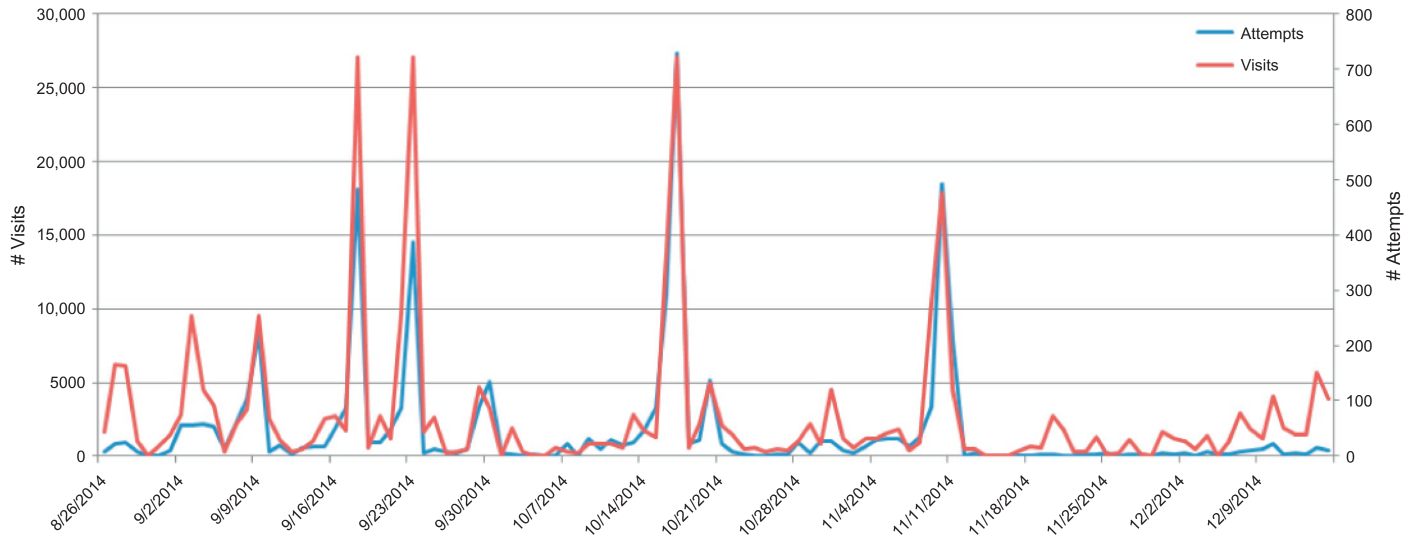


FIGURE 8.19

Gradebook page visits and exercises attempts time series plot.

We sought to investigate if there was a difference in the number of exercises attempted between students who visited the gradebook a lot and students who did not. We ranked students by the number of gradebook visits and grouped them into quartiles. There was no statistically significant difference in the mean number of exercises attempted among quartiles for gradebook use. We concluded that the presence of the gradebook has no impact on the amount of student interaction with OpenDSA exercises.

3.7 DISCUSSION

Our investigation of the impact of proficiency visual indicators revealed the following:

- Student responses indicated that the display of the green checkmarks (and other visual elements) is a motivating factor. Visual indicators are used in this case for reinforcement. Students like them and many students try to get closure from completion everywhere possible.
- Despite the fact that students like the gamification elements, we have so far found no evidence of positive impact on students' learning behavior due to them. Analysis of log data revealed that students often do not engage in a meaningful way with slideshows on their first attempt. They rush through slideshows, we assume to get the green checkmarks. However, they do spend more time on the slideshows during subsequent attempts. OpenDSA slideshows aim to present course material in a way that will encourage students to engage more deeply with the content. The fact that students do not spend enough time on slideshows despite the presence of reinforcements indicates that we should include other mechanisms that will better encourage students to engage with them.
- Students regularly refer to the gradebook to check their progress (though there is no evidence that this affects or is even correlated to their performance). Further study using data from courses in which students are assigned frequent assignments needs to be done in order to have a complete picture of student use of the gradebook. We found no relationship between gradebook page visits and the amount of exercises attempted by students.

Given these findings, the next step is to use this information in order to improve student outcomes. This can most likely be accomplished by modifications to the way that materials are presented. Such modifications might include further use of gamification elements. In order for students to learn the material better, it seems that the key goal needs to be to get them to engage more with the types of content that they tend to avoid now. Here are a number of ideas that might be promising avenues for future study:

- Increase the use of visualizations (slideshows) and interactive elements as much as possible, as this is a proven technique to increase student attention.
- Create "progression" mechanisms that lock access to certain aspects of the content until minimum requirements have been met. For examples, access to an exercise might be blocked until students have viewed the corresponding visualization. Access to proficiency exercises might require completion of simpler screening questions.
- Visualizations might themselves incorporate simple questions that must be passed in order to continue. This might discourage skipping through the slideshows for completion checkmarks.
- Add (optional) challenges for students who want to do more work.

- Add public "leaderboards" (perhaps using aliases) to let students see their performance in comparison to others in the class.

These ideas need careful consideration and experimentation before deployment. Some of them carry the risk of failing in various ways, in that they lead to unintended behavior. Or they might simply lead to increased frustration on the part of the users.

4 CONCLUSION AND FUTURE WORK

Visual interactive content, rich interactive exercises, automated assessment, detailed student analytics, and adaptive tutoring techniques can be combined in many ways to give students rich courseware that goes well beyond the capabilities previously available. However, we find that with any new courseware techniques, it is important to carefully monitor how it affects student learning. Often the results are not entirely as expected. Prior research, such as the rich body of work around cognitive theory of multimedia presentations (Mayer, 2008), provides us with a practical basis for developing multimedia content. However, the research community has only scratched the surface with respect to understanding the effects of student analytics, gamification, and interactive exercises with automated assessment. We are hopeful that the education research community can over time develop best practice principles to guide future course content and online educational systems.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Numbers DUE-1139861, IIS-1258571, and DUE-1432008.

REFERENCES

Andrade, H., Valtcheva, A., 2009. Promoting learning and achievement through self-assessment. Theory Pract. 48 (1), 12–19. http://www.tandfonline.com/doi/abs/10.1080/00405840802577544.

Breakiron, D.A., 2013. Evaluating the integration of online, interactive tutorials into a data structures and algorithms course. Master's thesis, Virginia Tech.

Corbett, A.T., Koedinger, K.R., Anderson, J.R., 1997. Intelligent tutoring systems. In: Helander, M.G., Landauer, T.K., Prabhu, P.V. (Eds.), Handbook of Human-Computer Interaction. Elsevier, pp. 849–874.

Deterding, S., Dixon, D., Khaled, R., Nacke, L., 2011. From game design elements to gamefulness: defining "gamification", In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek 2011). ACM, New York, NY, USA, pp. 9–15.

Falkner, N.J.G., Falkner, K.E., 2014. "Whither, badges?" or "Wither, badges!": a metastudy of badges in computer science education to clarify effects, significance and influence. In: Proceedings of the 14th Koli Calling International Conference on Computing Education Research. ACM, New York, NY, USA, pp. 127–135.

Feng, M., Roschelle, J., Heffernan, N., Fairman, J., Murphy, R., 2014. Implementation of an intelligent tutoring system for online homework support in an efficacy trial. In: Intelligent Tutoring Systems. Springer, pp. 561–566.

Ferguson, R., 2012. Learning analytics: drivers, developments and challenges. Int. J. Technol. Enhanc. Learn. 4 (5–6), 304–317.

Fouh, E., 2015. Building and evaluating a learning environment for data structures and algorithms courses. Ph.D. thesis, Virginia Tech, May.

Fouh, E., Breakiron, D.A., Hamouda, S., Farghally, M.F., Shaffer, C.A., 2014a. Exploring students learning behavior with an interactive etextbook in computer science courses. Comput. Hum. Behav. 41, 478–485.

Fouh, E., Karavirta, V., Breakiron, D.A., Hamouda, S., Hall, S., Naps, T.L., Shaffer, C.A., 2014b. Design and architecture of an interactive etextbook — the OpenDSA system. Sci. Comput. Program. 88, 22–40.

Goodrich, M.T., Tamassia, R., 1998. Teaching the analysis of algorithms with visual proofs. ACM SIGCSE Bull. 30, 207–211.

Haaranen, L., Ihantola, P., Hakulinen, L., Korhonen, A., 2014. How (not) to introduce badges to online exercises. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE 2014). ACM, New York, NY, USA, pp. 33–38.

Hamari, J., Koivisto, J., Sarsa, H., 2014. Does gamification work? A literature review of empirical studies on gamification. In: 47th Hawaii International Conference on System Sciences (HICSS). IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 3025–3034.

Hundhausen, C.D., Douglas, S.A., Stasko, J.T., 2002. A meta-study of algorithm visualization effectiveness. J. Vis. Lang. Comput. 13 (3), 259–290.

Karavirta, V., Shaffer, C.A., 2013. JSAV: the JavaScript algorithm visualization library. In: Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2013). ACM, New York, NY, USA, pp. 159–164.

Karavirta, V., Korhonen, A., Seppälä, O., 2013. Misconceptions in visual algorithm simulation revisited: on UI's effect on student performance, attitudes, and misconceptions. In: Proceedings of Learning and Teaching in Computing and Engineering, Macau.

Korhonen, A., Malmi, L., Silvasti, P., Nikander, J., Tenhunen, P., Mård, P., Salonen, H., Karavirta, V., 2003. TRAKLA2 website. http://www.cse.hut.fi/en/research/SVG/TRAKLA2/.

Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A., Malmi, L., 2005. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. Inform. Educ. 4 (1), 49–68.

Laakso, M.-J., Myller, N., Korhonen, A., 2009. Comparing learning performance of students using algorithm visualizations collaboratively on different engagement levels. Educ. Technol. Soc. 12 (2), 267–282. URL http://www.ifets.info/abstract.php?art_id=945.

Latulipe, C., Long, N.B., Seminario, C.E., 2015. Structuring flipped classes with lightweight teams and gamification. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education. ACM, New York, NY, USA, pp. 392–397.

Mayer, R., 2008. Applying the science of learning: evidence-based principles for the design of multimedia instruction. Am. Psychol. 63 (8), 760–769.

McMillan, J.H., Hearn, J., 2008. Student self-assessment: the key to stronger student motivation and higher achievement. Educ. Horiz. 87 (1), 40–49. URL http://www.eric.ed.gov/ERICDocs/data/ericdocs2sql/content_storage_01/0000019b/80/41/9e/80.pdf.

Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., et al., 2002. Exploring the role of visualization and engagement in computer science education. ACM SIGCSE Bull. 35, 131–152.

Parker, M., Lewis, C., 2014. What makes big-O analysis difficult: understanding how students understand runtime analysis. J. Comput. Sci. Coll. 29 (4), 164–174.

Roll, I., Aleven, V., McLaren, B.M., Koedinger, K.R., 2011. Improving students help-seeking skills using metacognitive feedback in an intelligent tutoring system. Learn. Instr. 21 (2), 267–280.

Shaffer, C.A., Cooper, M.L., Alon, A.J.D., Akbar, M., Stewart, M., Ponce, S., Edwards, S.H., 2010. Algorithm visualization: the state of the field. ACM Trans. Comput. Educ. 10 (3), 1–22.

Siemens, G., Baker, R.S.d., 2012. Learning analytics and educational data mining: towards communication and collaboration. In: Proceedings of the 2nd International Conference on Learning Analytics and Knowledge. ACM, New York, NY, USA, pp. 252–254.

Zhang, P., 2008. Technical opinion: motivational affordances: reasons for ICT design and use. Commun. ACM, 51 (11), 145–147.