INVESTIGATING DIFFICULT TOPICS IN A DATA STRUCTURES COURSE USING ITEM RESPONSE THEORY AND LOGGED DATA ANALYSIS

Eric Fouh
Lehigh University

Mohammed F. Farghally
Virginia Tech

Sally Hamouda
Virginia Tech

Kyuhan Koh
Virginia Tech

Clifford A. Shaffer
Virginia Tech
Agenda

- The OpenDSA System
- Identifying Difficult Topics
  - Through logged Data Analysis
  - Through Item Response Theory
  - Through Instructor Survey
- Why Algorithm Analysis is Hard?
- Future Work
The OpenDSA System

- Interactive eTextbook infrastructure and a body of content.
- Automatically graded interactive exercises and visualizations.
Identifying Difficult Topics

- To develop appropriate interventions and better allocate course resources.
- Analyzing exercise answers:
  - OpenDSA log data
    - Exercise correct ratio
    - Using hints and guessing
  - Item Response Theory
- 143 students enrolled in a CS3 course in Virginia Tech during Fall 2014.
- OpenDSA exercises accounted for 20% of total grade.
OpenDSA adopts a mastery level approach.

A Student’s performance on an exercise is assessed by:

\[ r_i = \frac{\text{# of correct attempts}}{\text{# of total attempts}} \]

Each exercise’s difficulty level is assessed by:

\[ dl = 1 - \frac{\sum_{i=1}^{n} r_i}{n} \]
Min: 0, Max: 0.72

<table>
<thead>
<tr>
<th>Quartile</th>
<th>Difficulty Level Range</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fourth</td>
<td>( dl &gt; 0.25 )</td>
<td>22 / 26 Algorithm Analysis</td>
</tr>
<tr>
<td>Third</td>
<td>( 0.13 \leq dl \leq 0.25 )</td>
<td>14 /25 Algorithm Mechanics, 10 /25 General course concepts</td>
</tr>
<tr>
<td>Second</td>
<td>( 0.05 \leq dl &lt; 0.13 )</td>
<td>23 / 25 Algorithm Mechanics</td>
</tr>
<tr>
<td>First</td>
<td>( dl &lt; 0.05 )</td>
<td>Algorithm Mechanics</td>
</tr>
</tbody>
</table>
Harder exercises are expected to show a higher rate of hint use and trial and error.

Hint ratio is defined as:

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

Incorrect ratio is defined as:

\[
ir = \frac{\text{# of incorrect answers}}{\text{# of total attempts}}
\]
Fourth Quartile:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Hint Ratio</th>
<th>Incorrect Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Analysis</td>
<td>0.24</td>
<td>0.77</td>
</tr>
<tr>
<td>Tree Overhead Analysis</td>
<td>0.78</td>
<td>0.73</td>
</tr>
<tr>
<td>Quicksort Analysis</td>
<td>0.32</td>
<td>0.67</td>
</tr>
<tr>
<td>Mathematical Background</td>
<td>0.25</td>
<td>0.63</td>
</tr>
<tr>
<td>Shellsort</td>
<td>0.16</td>
<td>0.61</td>
</tr>
<tr>
<td>List Overhead Analysis</td>
<td>0.93</td>
<td>0.60</td>
</tr>
<tr>
<td>Quicksort partition</td>
<td>0.27</td>
<td>0.58</td>
</tr>
</tbody>
</table>

First Quartile exercises are related to linear structures.
Item Response Theory Analysis

- A theory of testing that relates student performance on an individual item to his/her overall ability.

- IRT curves:
  - Item Characteristics Curve (ICC).
  - Item Information Curve (IIC).
  - Test Information Function (TIF).
Each chapter was treated as a test with exercises as items.

Exercise difficulty ratios were dichotomized.

- $r \geq 0.75$ -------- 1
- $r \leq 0.75$ -------- 0

$1PL$ model was adopted.

$$p_i(\theta) = \frac{e^{(\theta - b_i)}}{1 + e^{(\theta - b_i)}}$$
Overall 21 / 100 exercises discriminates between above average students.

19 of these 21 are related to Algorithm Analysis.
A survey was sent to CS3 instructors through SIGCSE mailing list.

23 instructor responses were analyzed.

<table>
<thead>
<tr>
<th>Topic</th>
<th># of instructors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Programming</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>Algorithm Analysis</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>OOP and Design</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>Recursion</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Trees and Heaps</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Proofs</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Why Algorithm Analysis is Hard?

OpenDSA logs were analyzed for 3 universities to see the time spent in Algorithm analysis content for the sorting chapter.

<table>
<thead>
<tr>
<th>University</th>
<th>Module</th>
<th>N</th>
<th>Mean (Sec)</th>
<th>% &lt; 1 min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia Tech</td>
<td>Insertionsort</td>
<td>98</td>
<td>63.57</td>
<td>74.48</td>
</tr>
<tr>
<td></td>
<td>Mergesort</td>
<td>96</td>
<td>39.79</td>
<td>78.12</td>
</tr>
<tr>
<td></td>
<td>Quicksort</td>
<td>92</td>
<td>64.71</td>
<td>73.91</td>
</tr>
<tr>
<td>Texas El Paso</td>
<td>Insertionsort</td>
<td>26</td>
<td>49.84</td>
<td>80.76</td>
</tr>
<tr>
<td></td>
<td>Mergesort</td>
<td>22</td>
<td>41.45</td>
<td>77.27</td>
</tr>
<tr>
<td></td>
<td>Quicksort</td>
<td>16</td>
<td>16.18</td>
<td>93.75</td>
</tr>
<tr>
<td>Florida</td>
<td>Insertionsort</td>
<td>53</td>
<td>40.39</td>
<td>84.90</td>
</tr>
<tr>
<td></td>
<td>Mergesort</td>
<td>44</td>
<td>18.63</td>
<td>95.45</td>
</tr>
<tr>
<td></td>
<td>Quicksort</td>
<td>39</td>
<td>26.12</td>
<td>92.30</td>
</tr>
</tbody>
</table>
84 students were surveyed at the end of Fall 2014 in Virginia Tech:

**Which is easier?**
- Algorithm Mechanics Easier: 86%
- Algorithm Analysis Easier: 14%

**Reason**
- Material: 78%
- Presentation Style: 22%

Why Algorithm Analysis is Hard? (Cont.)
Why Algorithm Analysis is Hard? (Cont.)

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

\[
\sum_{i=1}^{n} = n(n-1) \frac{n+1}{2} = \Theta(n^2).
\]

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

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

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

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

- Towards Visualizing Algorithm Analysis content in OpenDSA.
- 28 visualization were developed for the Algorithm Analysis Introduction and Sorting chapters.
- Preliminary results indicate higher student engagement and better student performance.
Worst case
Insertionsort

Mergesort

Worst case
Insertionsort
Mergesort
Selectionsort

Best case
Quicksort
Questions