Why is Big-O Analysis Hard?

Miranda Parker, Colleen Lewis
Harvey Mudd College
301 Platt Blvd
Claremont, CA, USA 91711
mparker, lewis@cs.hmc.edu

ABSTRACT

We are interested in increasing comprehension of how students understand big-O analysis. We conducted a qualitative analysis of interviews with two undergraduate students to identify sources of difficulty within the topic of big-O. This demonstrates the existence of various difficulties, which contribute to the sparse research on students' understanding of pedagogy. The students involved in the study have only minimal experience with big-O analysis, discussed within the first two introductory computer science courses. During these hour-long interviews, the students were asked to analyze code or a paragraph to find the runtime of the algorithm involved and invited students to write code that would in run a certain runtime. From these interactions, we conclude that students that have difficulties with big-O could be having trouble with the mathematical function used in the analysis and/or the techniques they used to solve the problem.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms

Human Factors

Keywords

Big-O, runtime analysis, algorithmic complexity

1. INTRODUCTION

Big-O is used in computer science to estimate the upper-bound of an algorithm's runtime [2]. We assume that big-O is important to students' ability to write efficient code. However, from our experience, students appear to have a fair share of difficulty with this subject. Big-O has been shown to be the most difficult topic for college students at the introductory programming level [3]. However, it's also seen as the least relevant topic at this level, and thus does not get all of the attention it deserves in terms of understanding why it is difficult [3].

We created an interview protocol designed to investigate students' understanding of big-O analysis. During the interview the students were asked to analyze code or a textual description to find the runtime of the algorithm involved. We also invited students to write code that would operate in a certain runtime. Afterwards their answers and actions were qualitatively analyzed in order to gain insight into their understanding of big-O.

Koli Calling'13, November 14-17, 2013, Koli, Finland.
http://dx.doi.org/10.1145/2526968.2526996
ACM 978-1-4503-2482-3/13/11.

2. ANALYSIS

In the analysis we focus on portions of an explanation Ethan (a pseudonym) gave for why he feels he does not understand big-O analysis. This was during the second attempt of a problem that asked him to write a function that runs in O(*log*(n)) time. He initially passed on the problem, but since we had extra time during the interview he chose to reconsider it. He worked towards a solution to the problem while he was mentioning what parts of big-O were hard for him.

From this discussion, we develop the idea that difficulty with big-O derives from two sections of understanding: the mathematical function in the analysis and the technique used to solve the analysis, be in plug-and-chug or reductive thinking [1].

2.1 Episode One

2.1.1 Data

01 Logs, they're always involving logs.
02 **Logs** are like the **least friendly thing**.
03 Like, with you know **n squared I could easily point to life** and
04 say that's an example of something being squared, but
05 a **log** is really **less tangible**, you know?
06 like, and even like trig functions, like sine, cosine, tangent, you can say
07 'Oh, triangles.'
08 Like uh I don't know log and uh
09 also I think part of it is just me.

2.1.2 Analysis

From the transcript presented above, it can be concluded that Ethan can have different levels of difficulty with different mathematical functions. This can be deduced from his differentiation in difficulty between squares and logs. He sees logs as "the least friendly thing" and "less tangible." This helps to provide context for why Ethan initially did not answer the logarithmic runtime problem, since he could tell from the problem statement what mathematical function it involved and he knew that he did not completely understand that function.

2.2 Episode Two

2.2.1 Data

01 Um at least **it helps me**
02 when I put the number six in there and
03 see, actually sort of count it and
04 reason it in my head with a **tangible** number and
05 then put it in **variable form**.

2.2.2 Analysis

Ethan's plug-and-chug technique for solving the problem may lead to difficulties in understanding the problem. He feels a sense of comfort, expressed in line one of this episode, with using this technique. He likes it because it uses "a tangible number." This is connected to the idea of having difficulties with mathematical functions. This is because plugging a number into a coded function can be more or less helpful in the big-O analysis depending on the mathematical function. For example, it may be easier to notice a number being squared than a number that has the log taken of it. However, just because a student understands a mathematical function does not imply that the student also has a valid and dependable technique for solving for the runtime. In other words, even if Ethan understood logarithms, his chosen technique for solving a problem might still give him difficulties with the runtime analysis.

2.3 Episode Three

2.3.1 Data

01 I know loops and recursion and stuff has n attached to them, but
02 I don't know how to mix and match them, and
03 I don't know what **corresponds** with what and
04 what logs **correspond** to.
05 I know there's some type of **correspondence** between **a type of programming thing** and **logs**, or whatever.

2.3.2 Analysis

Ethan desires a connection between the abstract (big-O analysis) and the concrete (algorithms, structures, etc.). He recognizes that certain programming structures or algorithms have certain runtimes, expressed in line one of this episode. However, he does not know all of these correspondences, and admits as much for logs in line four. A student could plausibly understand logarithms in a math context but not relate logarithms to inherently binary structures in a computer science context.

2.4 Analysis Summary

Ethan's interview led us to hypothesize about two possible areas that students could have difficulty in when learning big-O analysis. We are led to this conclusion through Ethan's discussions of tangibleness (leading to the plug-and-chug technique) and correspondence (the reductive thinking technique), which point to key parts of big-O analysis that, if misunderstood, could increase the difficulty of runtime analysis from the student's perspective. Ethan's dislike of logarithms carried through all of these areas, but that does not imply that the mathematical function and the solution technique are one and the same in terms of difficulty. The mathematical function interacts with the solution technique, including plug-and-chug and reductive thinking techniques, to create difficulty with big-O analysis, as seen in Figure 1.

3. CONCLUSION

From the analysis, we conjecture that two things work together to affect a student's understanding of big-O analysis: the mathematical function used in the big-O analysis and the technique the student uses to find the solution to the problem, be it plug-and-chug or reductive thinking [1].

In terms of the mathematical function, students seemed to have different experiences in solving a big-O analysis problem depending on what mathematical function was involved, such as $log(n)$ or n^2. This was most evident when the student was asked to write a function that ran with a certain big-O runtime.

Additionally, there were various techniques the student used to find a big-O runtime, some of which produced more correct answers for a student than others. In some cases, students would plug values into the algorithm and then try to extrapolate the runtime from the number of steps the algorithm took to produce the return value. In other cases, students had an easier time with a problem when they could determine a pattern in the algorithm that they seen before, such as a certain set of recursive calls, and associate it with a certain runtime.

The data suggests that the mathematical function and the technique used in solving the problem are connected, since the technique that a student uses may produce a wrong answer depending on the mathematical function that is involved. For example, some students found it much easier to detect a polynomial pattern than a logarithmic pattern.

This study takes the first step towards understanding how students reason about big-O. Although only a few examples are provided, these examples of why big-O is difficult can still make a difference in the pedagogy of this topic. Furthermore, this research can easily be expanded to explore more areas of big-O with which students struggle.

4. REFERENCES

[1] Armoni, M., Gal-Ezer, J. and Hazzan, O. 2006. Reductive Thinking in Undergraduate CS Courses. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (ITICSE '06). ACM, New York, NY, USA, 133-137. DOI=10.1145/1140124.1140161

[2] Dasgupta, S., Papadimitriou, C.H. and Vazirani, U.V. 2006. *Algorithms*. McGraw-Hill.

[3] Schulte, C., and Bennedsen, J. What Do Teachers Teach in Introductory Programming? 2006. In *Proceedings of the second international workshop on Computing education research* (ICER '06). ACM, New York, NY, USA, 17-28. DOI=10.1145/1151588.1151593