WHAT MAKES BIG-O ANALYSIS DIFFICULT: UNDERSTANDING HOW STUDENTS UNDERSTAND RUNTIME ANALYSIS *

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 classes. 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 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

---

* Copyright © 2014 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

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 [5]. We assume that big-O is important to students' ability to write efficient code and habitually consider the efficiency of their 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 at a college level [9]. 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 figuring out why it is difficult [9].

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 paragraph to find the runtime of the algorithm involved and invited students to write code that would run in a certain runtime. Afterwards their answers and actions were qualitatively analyzed in order to gain insight into their understanding of big-O.

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.

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 in 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 characteristic in the algorithm that they had seen before, such as a certain recursive pattern, 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, for some students it is 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 how 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 that students have trouble with.

2. PREVIOUS RESEARCH

Previous studies have focused on teaching big-O analysis, including teaching strategies [1,2,3,7,10,13], homework [8,9], and labs [7]. However, these are heavily focused on the teacher, since the teacher teaches the material and they create, assign, and grade the assignments. This means that this research has heavily emphasized how the teacher understands big-O and how they try to transfer that knowledge to students. We have strategies for teaching big-O, but need to understand why the topic is difficult for the students.

Various research has been done on different learning strategies, with the end goal being to motivate students more [13] or to help students understand the material more [1,2,3,7,8,9,10]. These papers look at different ways to teach and learn about big-O [1,3,13], different ways to analyze big-O [2,8], and different activities to do boost understanding of big-O [7,9]. These approaches can be offered up as solutions for students having trouble with big-O analysis. Some of these techniques demonstrated great effectiveness, but it is not possible to know what aspects of these techniques are necessary or how to adapt these techniques to various contexts. This led us look into where students have trouble with big-O analysis. This might, in turn, help to craft a more effective solution, catered to the students' needs.

Despite the presence of various teaching strategies, it is commonly acknowledged that students struggle with abstract concepts in computer science [3,10,12]. However, these studies do not present hypotheses regarding why students struggle. More often, brief explanations are given saying:

- big-O is "hard to grasp" [10], or
- the "difficulty stems from the complexity of the abstract notions involved and the required mathematical background" [3], or
- the topic is theoretically different than other topics covered in other computer science courses and students failing "to appreciate the relevance of the material due to its theoretical nature." [12].

None of the reasons why big-O might be difficult are justified by an investigation [3,10,12].

This work expands concepts discussed in these works by formalizing where students are having difficulty with big-O analysis. This work can address some areas that these works did not, such as specifically focusing on the difficult aspects of big-O analysis for students and why those are the difficult ones.

3. METHODS

3.1 Data Collection Methods

We interviewed two students who had taken both Introduction to Computer Science and Principles of Computer Sciences[1] and had been introduced to the concept of big-O

[1]One student had taken only one college computer science class before, but it covered the material from both of these classes.

analysis. Both were taught and tested on big-O analysis. These interviews were one-on-one, and lasted approximately an hour.

The interviewer started by asking the students introductory questions, such as what big-O means, to get a sense of their overall understanding of big-O analysis. Then they were asked six big-O questions. They were asked two questions where the participant was asked to analyze a code sample and determine the big-O runtime, two questions that asked the participant to analyze an algorithm described by a scenario and determine the big-O runtime, and two questions that invited the participant to write a function that would run within a certain runtime. If done correctly, the solutions involved two logarithmic runtimes, two polynomial runtimes, one linear runtime, and one constant runtime. The participants were given simple instructions, only asking them what the runtime is or to write a function.

During the interview, if the student came to a certain answer, the interviewer would not tell them if it was right or wrong but instead ask the student to walk through their thought process and try to justify their answer. If a student was stuck, the interviewer tried to ask them about any understanding they might have of the problem, which usually helped. For example, the student would be asked to consider what problems they had seen like this one before. Sometimes the student would either not talk through their thought process or would skip explaining a step, in which case they would be asked to walk the interviewer through that part again.

3.2 Data Analysis Methods

We used a grounded approach to analyze the interviews [4,6], which involved the creation of analytic memos, content logs, and selected transcripts.

3.2.1 Analytic Memos

This analytic process began with creating analytic memos in which the interviewer's reactions and notable patterns from the interview were documented. This type of memo is common in grounded analyses to aid in later data analysis and to document insights and interpretations of the interviewer, which may be relevant to the analysis, but are not easily recreated.

In these analytic memos, the interviewer noted that both students had difficulty with logarithmic runtimes. They both hesitated on the logarithmic problems, and both answered these questions incorrectly. Before additional data analysis, the interviewer developed the hypothesis that students' trouble with big-O may actually be a problem with logarithms.

3.2.2 Content Logs

After each interview, the interviewer watched the video recording of the interview and created content logs [6] documenting significant events within the interview. These content logs helped to determine which portion of the interviews to focus on in later analysis. Rather than just documenting events, these content logs are part of the process of analysis and interpretation.

Logarithms recurred as a theme of interest in creating the content logs due to the depth of answers students provided regarding logarithms. In delineating each question and answer in the interview we observed that students tended to spend more time on the logarithms than on other questions.

3.2.3 Selected Transcripts

The interviewer selected pieces of the interview to focus on, based on their interpretation of the content logs. Only one of the students, Ethan (a pseudonym), was chosen to be analyzed, and only a small portion of his interview when he talked freely about his understanding of big-O analysis and logarithm was analyzed. The monologue was narrowed down further by coming up with main points that the student was making, and choosing portions of the transcript that best exemplified those points.

We chose to analyze this student and this section of his interview because they appeared fruitful in terms of gaining a candid perspective on the how the student reasons about big-O.

4. PROBLEM DESCRIPTION

During the section of interview the analysis focuses on, the student is working on a problem where he is told to write a function that runs in *log*(n) time. This question was written as simply as possible:

Please write an algorithm that runs in O(*log*(n)) time.

This problem has endless solutions. For example, this problem could be solved by writing a binary search function.

5. ANALYSIS

In the analysis we focus on portions of an explanation Ethan 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 it the plug-and-chug or reductive thinking techniques.

Table 1 shows the definitions of terms used in the analysis. The relationship between these terms will be discussed later on.

Table 1. A list of terms with definitions

Term	Definition
Reductive thinking	This is a technique where a student transforms a problem "into a simpler problem (not problems) for which a solution is already known, and constructing, or deducing, the solution to the original problem based on the solutions to the reduced problems" [1].
Plug-and-chug	This is a technique where a student plugs in an appropriate input into an algorithm and computes what the algorithm will return, keeping in mind the number of steps the algorithm takes. Usually this is repeated multiple times, after which the student will formulate a solution for the runtime of the algorithm based on the relationship between the input and how many steps the algorithm ran for.
Solution technique	This is the technique the student used to solve the big-O analysis problem. This contains both the reductive thinking technique and the plug-and-chug technique.
Mathematical function	This is the mathematical function used in the big-O analysis. *log*(n) would be the mathematical function in an algorithm that has a runtime of O(*log*(n)).

5.1 Episode One

5.1.1 Summary

In this episode, Ethan provides evidence of having difficulty with the mathematical function. This is Ethan's first mention of logarithms during this portion of the interview, and he mentions them with a negative connotation. He describes his difficulty as that logs are much harder to grasp conceptually than polynomials.

5.1.2 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.

5.1.3 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.

5.2 Episode Two

5.2.1 Summary

In this episode, Ethan provides evidence for difficulty with the plug-and-chug solution technique. Ethan discusses his technique for solving big-O problems. This technique involves plugging in a "tangible" input to the function and then reasoning the output into "variable form." While using this technique to solve problems earlier in the interview, he occasionally came up with incorrect answers.

5.2.2 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**.

5.2.3 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.

5.3 Episode Three

5.3.1 Summary

In this final episode, Ethan provides evidence for the reductive thinking aspect of difficulty with big-O. Ethan mentions the relation of big-O analyses to certain programming problems, and his current understanding of those connections.

5.3.2 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.

5.3.3 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 connections, and admits as much for logs in line four. We see this as a reference to reductive reasoning, where Ethan is searching for a connection to a previous, perhaps simpler, problem that he has seen before in order to solve the one he is faced with. Since this is the runtime he has specifically mentioned having trouble with, not having that concrete connection could be hurting his understanding. This, like the plug-and-chug technique, also ties back to the mathematical function involved in the analysis. It might be harder for a student to use reductive thinking with certain math functions because the latter may have made the former harder to conceptualize and remember, and thus harder to recall. However, again, this concept is separate from the mathematical function since the student could still have problems with reductive thinking even if they completely understand the mathematical function used in the correct answer. In other words, a student could plausibly understand logarithms in a math context but not relate logarithms to inherently binary structures in a computer science context.

5.4 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.

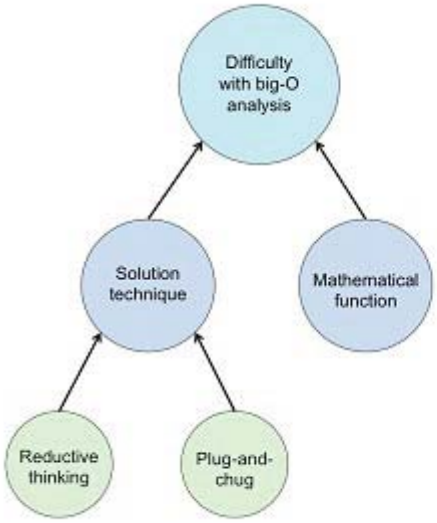

Figure 1. Relationship between different aspects of solving a big-O analysis problem.

6. CONCLUSION

In this work, we try to understand how students understand this topic in order to work towards the goal of improving big-O analysis pedagogy.

This study offers suggestions as to where students struggle with big-O analysis. This is in contrast to previous studies that frequently look more at the solution to students having trouble than understanding the source of the problem. This study begins by being open to the idea that students could have trouble with any aspect of the process, including the mathematical function, the programmed function, the theory behind big-O analysis, etc. Properly pinpointing the problem(s) could help formulate ways to help the students having trouble.

We draw the idea that students can have trouble with big-O analysis for two possible reasons: the mathematical function that is used in the problem or the solution technique the students uses to solve the problem. These are connected since "harder" mathematical functions may be more difficult to find with certain solution techniques. For example, a student may have a more difficult time using their technique when the answer is O(*log*(n)), since logarithms can produce a pattern that some students do not properly recognize.

We qualitatively analyze the results from a segment of interview with Ethan, who describes his trouble with logarithms because they are "less tangible" and "the least friendly things" to him. He also goes on to talk about how plugging in a number makes it more tangible, and thus easier, for him. This is an example of having trouble with the mathematical function, and how that relates to the technique the student uses. Finally, he mentions the idea of correspondence between the analysis and a "programming thing", which we use the term "reductive thinking"[1] and put it in the category of how a student

may go about solving the problem. These analyses of the interview with Ethan led to our current hypothesis that both the mathematical function and the technique used in solving the problem are connected and both could contribute to difficulties students have with big-O analysis.

7. FUTURE WORK

We hope to continue this research by interviewing more undergraduate students to get a more defined sense of students' understanding of big-O analysis. We will qualitatively analyze these interviews, from which we will either strengthen the current argument for the breakdown of big-O understanding, or reformulate it accordingly.

We are also interested in interviewing more experienced computer science students and ask them about their understanding of big-O analysis. If the students remember having trouble and remember what helped them that could be fruitful in better understanding how students' understand big-O analysis.

From the findings from all the interviews, we want to create a plan of action for professors in teaching big-O analysis and for tutors in helping students understand the topic. For the former, this would entail a description of how students become confused while learning big-O, as well as techniques that may better assist the students in learning the material. For the latter, this could include a flowchart detailing where students have difficulty with big-O analysis and steps the tutor could take to remedy that difficulty.

8. REFERENCES

[1] Armoni, M., Gal-Ezer, J. and Hazzan, O., Reductive thinking in undergraduate CS courses, *Proceedings of the 11th annual SIGCSE conference on innovation and technology in computer science education*, (ITICSE '06), 133-137, 2006.

[2] Chaudhuri, R., Teaching bit-level algorithm analysis to the undergraduates in computer science, *SIGCSE Bull*, 36, (2), 62-63, June 2004.

[3] Chesñevar, C.I., González, M.P. and Maguitman, A.G., Didactic strategies for promoting significant learning in formal languages and automata theory, *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, (ITICSE '04), 7-11. 2004.

[4] Corbin, J. M., and Strauss, A. C., *Basics of Qualitative Research*, Thousand Oaks, CA: SAGE Publications, 2008.

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

[6] Engle, R. A., Conant, F. R. and Greeno, J. G., Progressive refinement of hypotheses in video-supported research, R. Goldman, R. Pea, B. J. Barron & S. Derry (Eds.), *Video research in the learning sciences* (pp. 239-254). Mahwah, NJ: Erlbaum, 2007.

[7] Hundhausen, C.D. and Brown, J.L., Personalizing and discussing algorithms within CS1 studio experiences: an observational study*, Proceedings of the first*

international workshop on computing education research (ICER '05), 45-56, 2005.

[8] Kerr, S., A big-O experiment: which function is it?, *Proceedings of the 43rd annual Southeast regional conference - Volume 1* (ACM-SE 43), Vol. 1, 317-318, 2005.

[9] Luxton-Reilly, A., Bertinshaw, D., Denny, P., Plimmer, B. and Sheehan, R., The impact of question generation activities on performance, *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (SIGCSE '12, 391-396, 2012.

[10] Roussos, C., Teaching growth of functions using equivalence classes: an alternative to big O notation, *Proceedings of the 35th SIGCSE technical symposium on Computer science education* (SIGCSE '04), 170-174, 2004.

[11] Schulte, C., and Bennedsen, J., What do teachers teach in introductory programming?, *Proceedings of the second international workshop on Computing education research* (ICER '06), 17-28, 2006.

[12] Sigman, S., Engaging students in formal language theory and theory of computation, *SIGCSE Bull.* 39, (1), 450-453, March 2007.

[13] Standish, T.A. and Jacobson, N., Using O(*n*) ProxmapSort and O(1) ProxmapSearch to motivate CS2 students (Part I), *SIGCSE Bull*. 37, (4), 41-44, December 2005.