University of Nebraska at Omaha
DigitalCommons@UNO

Theses/Capstones/Creative Projects

University Honors Program

12-2022

Obstacles in Learning Algorithm Run-time Complexity Analysis

Bailey Licht
baileylicht@unomaha.edu

Follow this and additional works at: https://digitalcommons.unomaha.edu/university_honors_program

Part of the Educational Methods Commons, Science and Mathematics Education Commons, and the Theory and Algorithms Commons

Recommended Citation
Licht, Bailey, "Obstacles in Learning Algorithm Run-time Complexity Analysis" (2022). *Theses/Capstones/Creative Projects*. 193.
https://digitalcommons.unomaha.edu/university_honors_program/193

This Dissertation/Thesis is brought to you for free and open access by the University Honors Program at DigitalCommons@UNO. It has been accepted for inclusion in Theses/Capstones/Creative Projects by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

University of Nebraska at Omaha
College of Information Science & Technology
Department of Computer Science
Supervisor: Dr. Harvey Siy

Honors Capstone Report

in partial fulfillment for the degree
Bachelor of Science in Computer Science (Honors Distinction)
in Fall 2022

Obstacles in Learning Algorithm Run-time Complexity Analysis

—

Addressing the Challenges Faced by Students

Submitted by:

Bailey Licht
E-Mail: baileylicht@unomaha.edu
B.S. Computer Science

Submission date: December 2022

Abstract

Algorithm run-time complexity analysis is an important topic in data structures and algorithms courses, but it is also a topic that many students struggle with. Commonly cited difficulties include the necessary mathematical background knowledge, the abstract nature of the topic, and the presentation style of the material. Analyzing the subject of algorithm analysis using multiple learning theories shows that course materials often leave out key steps in the learning process and neglect certain learning styles. Students can be more successful at learning algorithm run-time complexity analysis if these missing stages and learning styles are addressed.

Contents

List of Figures iii

1 Introduction 1

2 Common Challenges of Learning Algorithm Run-time Complexity Analysis 2

2.1 Lack of Necessary Mathematical Background Knowledge 2

2.2 Abstract Concept 3

2.3 Lack of Engagement With the Material 4

3 Explaining the Difficulties of Learning Algorithm Run-time Complexity Analysis Using Learning Theories 5

3.1 Threshold Concepts 5

3.2 APOS Theory 6

3.3 Kolb's Theory of Experiential Learning 7

3.4 Felder's Learning Styles 8

4 Suggestions for Teaching Algorithm Run-time Complexity Analysis 11

5 Conclusion 13

References 14

List of Figures

Figure 1: Big-O Complexity Graph 1
Figure 2: Kolb's Theory of Experiential Learning 8
Figure 3: Felder's Learning Styles 9
Figure 4: Application for Teaching Algorithm Run-time Complexity Analysis 12

1 Introduction

Algorithm run-time complexity analysis is a common topic in introductory data structures and algorithms courses. It is typically taught using Big-O notation. Big-O notation is used to describe the rate of growth of the run time of an algorithm with regard to the size of the input. Coefficients and non-dominant terms are dropped, so an algorithm with a rate of growth of $6n^2 + 5n + 2$ would have a Big-O complexity of $O(n^2)$. Big-O notation is usually used to describe the upper bound of an algorithm's complexity, while Big-Omega and Big-Theta are used to describe the lower bound and exact bound, respectively (Huang, 2020). A graph showing the number of operations performed by an algorithm with different complexities can be seen in Figure 1.

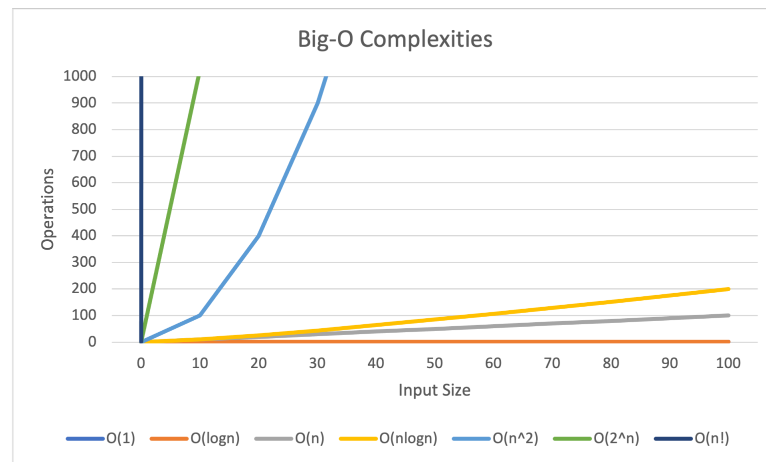

Figure 1: Big-O Complexity Graph

Source: Own Illustration

Learning algorithm run-time complexity analysis can help students improve their ability to write fast and efficient code. It is a valuable tool for comparing multiple approaches to solving the same problem. Unfortunately, it is commonly cited as one of the most difficult topics in data structures and algorithms courses (Fouh et al., 2016). Students and teachers point to varying reasons related to the topic itself and the method of information delivery that make algorithm run-time complexity analysis difficult to learn. This paper will examine some of the most commonly cited learning obstacles and how they can be addressed. It will also present multiple learning theories that can be used to examine these difficulties. Students can be more successful at learning algorithm run-time complexity analysis if the teaching material accommodates varying learning styles and addresses all stages of learning.

2 Common Challenges of Learning Algorithm Run-time Complexity Analysis

The reason that algorithm run-time complexity analysis is difficult to learn can vary between individual students. However, there are hurdles that are frequently pointed to by instructors and students alike as sticking points. Understanding these hurdles can equip instructors to address them and students to overcome them.

2.1 Lack of Necessary Mathematical Background Knowledge

When asked in a survey about the reasons that students struggle with algorithm run-time complexity analysis, instructors frequently cited a lack of mathematical proficiency as a major hurdle. Some students echoed a similar sentiment, stating that they were intimidated by the mathematical notation (Fouh et al., 2016). Logarithms are an example of a concept that students are often unfamiliar with, making it difficult for them to recognize when an algorithm has logarithmic complexity (Parker and Lewis, 2014). There are many mathematical concepts that must be understood before learning algorithm run-time complexity analysis, and it appears that students have not yet gained the necessary familiarity with these concepts before starting the data structures and algorithms course.

The obvious solution to this issue is to ensure that these concepts are covered in prerequisite courses to data structures and algorithms. However, this is already the case in many computer science programs, yet students are still uncomfortable with the material. A review of the important mathematical concepts could benefit students. Harris recommends using mathematical software such as Maple to help students learn algorithm analysis. The software can be used to generate a plot that will give students a visualization of the growth in run time of an algorithm. It can also be used to automate parts of the analysis of more complex algorithms that may require a higher level of mathematical knowledge (2006). These solutions can all benefit students. However, some instructors argue that the issue is emblematic of a larger problem in computer science.

"Too many students enter the field of computer science with high aspirations but poor math skills. These students often do not realize the significance of mathematics in computer science" (Beaubouef, 2002). Beaubouef has found that many students are not aware of the relevance and importance of mathematics to most key topics in computer science, causing students to avoid focusing on important mathematical concepts. This may not cause issues in introductory programming courses, but it can create hurdles for students later in their education and career (2002). A potential solution to the lack of mathematical background

knowledge among data structures and algorithms students is to better emphasize the importance of math in early computer science courses. If students understand the importance of math, they will put more effort into understanding the key concepts rather than viewing it as a general education subject that is not directly relevant to their area of study.

2.2 Abstract Concept

A common complaint from students learning algorithm analysis is that the material is purely theoretical, and they do not see the practical application. Many students are more comfortable with algorithm dynamics than algorithm analysis because they find the latter to be too abstract (Fouh et al., 2016). Big-O notation seems unnatural and unintuitive to some students. Kay and Wong found that when asked to analyze an algorithm in coursework, students would measure the run times of the algorithm, but they required an external source to determine the Big-O complexity (2018). Similarly, Parker and Lewis found that a student memorized the run-time complexity of specific algorithms. However, they struggled to analyze the run time of an algorithm without comparing it to another that they already knew (2014).

It is understandable that timing the actual run time of an algorithm is the first solution that comes to mind for students when trying to determine the speed and complexity of an algorithm. However, it appears that students are failing to see the benefits of Big-O as a form of run-time complexity analysis that has a broader application. The Big-O complexity of an algorithm is independent of the hardware or programming language that is used to implement the algorithm. It determines how run time changes with the size of the input. While timing an algorithm may be useful for small-scale projects, it is impractical when an algorithm will be used on varying platforms and with variably sized inputs. Data structures and algorithms course materials should emphasize the real-world benefits of algorithm run-time complexity analysis. Concrete examples of Big-O complexity's impact on run time could help students see these benefits. Shi and Shi suggest showing the formal derivation of algorithms in computer science courses. They argue that if students understand the ideas behind an algorithm, it will help them improve their algorithm design and analysis abilities (2009). However, this could cause even more confusion for students who are already struggling with the mathematical ideas in a data structures and algorithms course. Course materials should also emphasize the process of analyzing algorithm complexity. Knowing the complexity of existing algorithms is useful, but it does not help when trying to program an efficient solution to a novel problem.

2.3 Lack of Engagement With the Material

Another common complaint from students is that the learning material for algorithm run-time complexity analysis is not engaging. In their analysis of student activity in a digital data structures and algorithms textbook, Fouh et al. found that students were spending less than a minute on the algorithm analysis sections. Some students chalked this up to the presentation of the material. Other sections in the textbook made use of interactive visualizations, and students tended to spend more time on these sections and report that they were more beneficial (2016).

Farghally et al. set out to create visualizations that would help students understand algorithm analysis. These visualizations featured an input array and a graph corresponding to the number of operations performed. They found these visualizations to be effective at improving student engagement and performance (2017). It appears that the presentation style of algorithm run-time complexity analysis is a significant hurdle. While it may be that students are simply uninterested in the typical presentation style, it could also be that the material does not cater to certain learning styles. This idea will be explored further in sections 3.3 and 3.4.

3 Explaining the Difficulties of Learning Algorithm Run-time Complexity Analysis Using Learning Theories

Addressing the most common complaints could benefit students, but it does not explain what makes algorithm run-time complexity analysis fundamentally difficult to learn. This section will examine the challenges of learning algorithm analysis using four different theories. These theories point to a more overarching idea of what makes the topic difficult and lead to some ideas for the general design of algorithm run-time complexity analysis learning materials.

3.1 Threshold Concepts

The idea of threshold concepts was introduced by Eric Meyer and Ray Land. They are concepts that need to be understood to gain mastery of a subject. Threshold concepts tend to cause a slowdown in students' progress. However, once they are understood, they cause a change in a student's perspective (Kallia and Sentance, 2017). There are seven key features that characterize a threshold concept. The first is that threshold concepts are transformative. They fundamentally change the way that a student understands the subject. The second key feature is that threshold concepts are integrative. They integrate with existing knowledge and change how students view ideas that they already understand. Threshold concepts are also bounded. "...this feature indicates that threshold concepts have borders that, when traversed, can lead to other conceptual developments" (2017). They can be used to differentiate between areas of study. Threshold concepts are troublesome. They are difficult to understand, and they tend to be obstacles for students that are learning a subject. Threshold concepts are irreversible. Because of the way that they shift a student's understanding, they are unlikely to be forgotten. Threshold concepts are discursive. They will lead students to start using the language and jargon of the field that they are studying. Finally, threshold concepts are reconstitutive. They will cause students to reconfigure the schema by which they understand the subject (2017).

Kallia and Sentance found that instructors commonly cite parameter passing, variable scope, control flow, and recursion as threshold concepts within computer science (2017). Perhaps algorithm run-time complexity analysis could be viewed as a threshold concept in data structures and algorithms. It is clearly troublesome, as it is considered one of the most difficult topics within a data structures and algorithms course by students and instructors alike. It also has the potential to fundamentally change a student's understanding of algorithms. Kay and Wong noted that students who did not understand algorithm run-time complexity

analysis would perform analysis by running an algorithm and measuring the actual running time (2018). Learning run-time analysis could change how students approach the problem of determining an algorithm's complexity, as well as their approach to many programming problems. If algorithm run-time complexity analysis is a threshold concept, that helps to explain why it is difficult and frustrating for students to learn. It may be that it needs to be focused on as a topic that is central to one's understanding of data structures and algorithms. Instructors and students alike should be prepared to spend a significant amount of time on the topic.

3.2 APOS Theory

APOS Theory is a constructivist learning theory that was developed for mathematics education. It breaks learning up into three stages. The first is the action stage. When a learner is at this stage, they can perform a task by following step-by step instructions. The next stage of learning is the process stage. At this stage, the student is able to perform the task without any external help. They can skip and reorder steps, and even imagine the result of the task without carrying it out. The final stage is the object stage. At this stage, the student can understand the task as an object and visualize the whole process. As students learn, they develop a schema, which connects related actions, processes, and objects (Kay and Wong, 2018). "APOS Theory is a tool that can be used objectively to explain student difficulties with a broad range of mathematical concepts and to suggest ways that students can learn these concepts" (Dubinsky and McDonald, 2001). While it was originally developed for mathematics education, APOS Theory could help highlight the points that are causing difficulties for students that are learning algorithm run-time complexity analysis.

Kay and Wong attempted to do just that in their 2018 study. They noted that students were relying on textual cues to determine the complexity of an algorithm, such as assuming that code containing a for loop had a complexity of $O(n)$. This showed that they had only reached the action stage. They also noted that the coursework for the data structures and algorithms course that they were analyzing skipped over the action stage. It featured discussion of algorithm analysis without showing students how to perform it on a concrete algorithm (2018). Their general recommendation for teaching algorithm analysis was to support each stage of learning. The action stage could be supported using concrete worked examples. The process stage could be supported by having students derive multiple growth functions for the same algorithm. Stack traces or call trees could help students understand the process of algorithm analysis as an object (2018). Students need to learn to walk before they can run. They will not be able to apply the concepts of algorithm run-time

complexity analysis in any meaningful way if they are not taught the steps needed to perform the analysis in the first place.

One criticism of APOS Theory is that it does not take individual students' thought processes into account (Kay and Wong, 2018). Every student does not learn in the same way, but APOS theory takes a generalized approach. Sections 3.3 and 3.4 will discuss theories that address the learning styles of individual students.

3.3 Kolb's Theory of Experiential Learning

There are two key elements to Kolb's Learning Theory. The first is the experiential learning cycle. The experiential learning cycle is made up of four stages. "The process of going through the cycle results in the formation of increasingly complex and abstract 'mental models' of whatever the learner is learning about" (Mcleod, 2017). The cycle can be started from any stage, but the learner must go through all four stages to learn effectively. The first stage is concrete experience, where the learner undergoes a new experience. The second stage is reflective observation, where the learner reflects on the experience and compares it to existing knowledge. The third stage is abstract conceptualization, where the learner creates new ideas or modifies existing ideas based on their experience. The fourth stage is active experimentation, where the learner tries applying their newly formed ideas in the real world (2017). The stages of Kolb's Experiential Learning Cycle are similar to those in APOS Theory, but Kolb's Theory argues that learners can begin learning at different stages. Regardless, it is important that coursework supports students through all of the different stages. Both theories seem to imply that concrete worked examples are a good place to start. They can support the action stage of APOS Theory and the concrete experience stage of Kolb's Experiential Learning Cycle. Coursework should also lead students to reflect on the topic in relation to other topics they have learned and give them opportunities to apply their newly learned ideas.

The learning styles are the other element of Kolb's Theory. They categorize people into four quadrants based on the style of learning they prefer. People within the diverging quadrant are good at viewing information from multiple perspectives and like to work in groups. They prefer to watch, gather information, and generate ideas rather than do (2017). This quadrant can also be referred as the "why" quadrant, as students in this quadrant want to know why the subject is important (Howard et al., 1996). Students in the assimilating quadrant are interested in abstract ideas and concepts and require a good explanation of the topic more than a concrete experience (Mcleod, 2017). This quadrant is also known as the "what" quadrant, as individuals within this quadrant like knowing the facts of the subject (Howard et al., 1996). Students in the converging quadrant like technical tasks. They want

to find practical solutions to problems (Mcleod, 2017). This is also known as the "how" quadrant because students in this quadrant want to apply the information to the real world (Howard et al., 1996). The fourth and final quadrant is the accommodating quadrant, which is the most common learning style. Students in this quadrant enjoy new experiences and primarily rely on intuition (Mcleod, 2017). It is also referred to as the "what if" quadrant, as learners in this quadrant like to experiment (Howard et al., 1996).

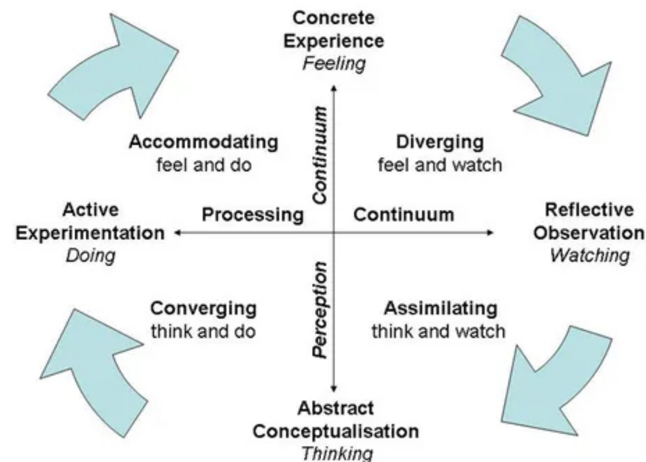

Figure 2: Kolb's Theory of Experiential Learning
Source: Mcleod (2017)

Instructors should attempt to design a course that reaches students in all four quadrants. Howard et al. suggest starting with the "why" quadrant so students understand the motivation for learning the material. The "what" quadrant can be addressed by explaining terminology. Students in the "how" quadrant can be supported if they are given the opportunity to apply their knowledge to real problems. Finally, the "what if" quadrant can be addressed by examining how the problem could be changed and using this to transition to new subjects (1996). Failing to address all of the learning styles could be what causes students to find algorithm run-time complexity analysis material unengaging. Therefore, addressing every quadrant of Kolb's learning styles could help increase the success rate of students.

3.4 Felder's Learning Styles

Felder's Learning Styles are another way to categorize the different ways that students prefer to receive information. Learners are categorized along four axes. The first axis is active

vs. reflective learners. Active learners learn best when they can do something hands-on, and they typically like to work in groups. Reflective leaners prefer thinking and reflecting on new information on their own. The next axis is sensing vs. intuitive learners. Sensing learners like to learn facts and use existing methods to solve problems. Intuitive learners like discovery and innovation, and dislike repetition. The third axis is visual vs. verbal learners. Visual leaners learn best by seeing, while verbal learners learn better by hearing or reading information. Felder and Soloman argue that everyone learns better when information is presented both ways. The final axis is sequential vs. global learners. Sequential learners acquire information in a linear, step-by-step manner. Global learners gain knowledge in large chunks and need to see the bigger picture to understand information (Felder and Soloman, 2000).

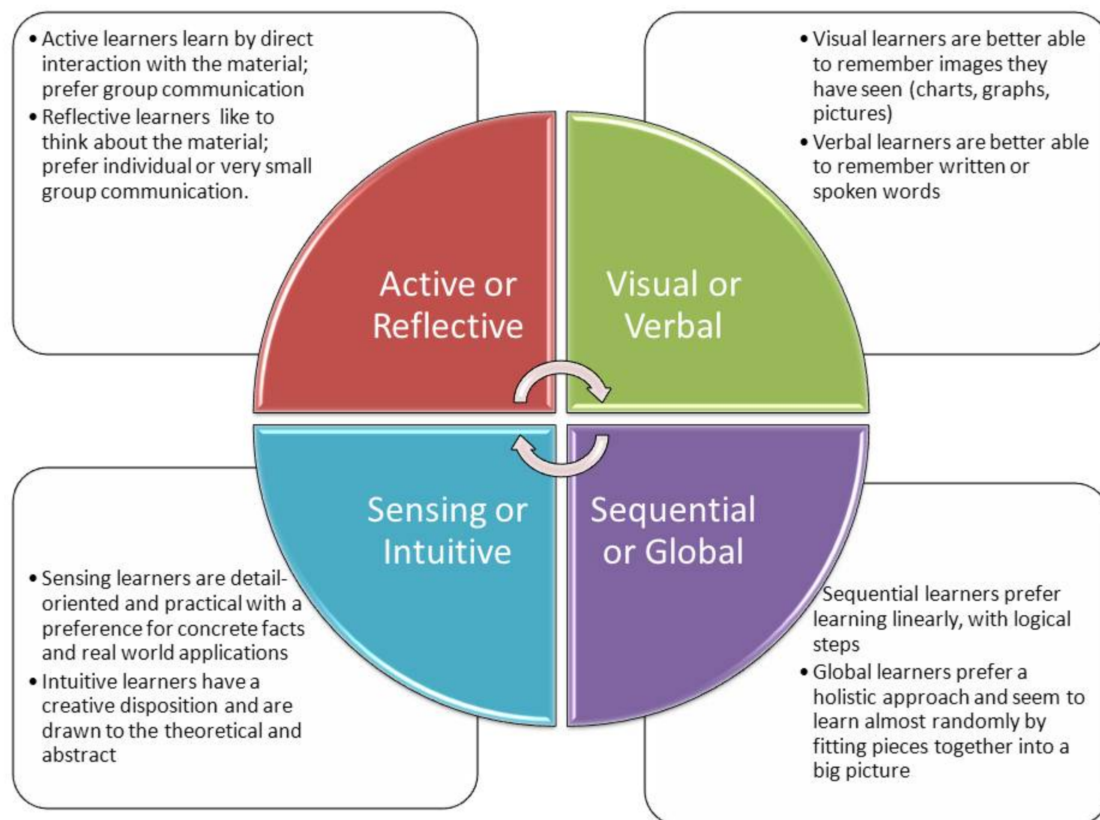

Figure 3: Felder's Learning Styles

Source: Cater (2011)

Teachers should make an effort to accommodate these learning styles. Howard et al. suggest accommodating active learners through group work and programming problems (1996). Verbal learners are easy to accommodate in a traditional lecture setting. Visual learners can be supported through graphs comparing algorithm run-time complexities or

algorithm visualizations. If a course fails to address certain learning styles, Howard et al. suggest study methods for students to supplement the course materials. Active learners can organize group study sessions where they take turns explaining topics to one another. Reflective learners can pause occasionally to review new information. Sensing learners can try to find out how the ideas they have learned apply in practice. Intuitive learners can look for theories that link the facts. Visual learners should seek out diagrams and schematics, while verbal learners can write summaries of the information they have learned. Sequential learners should look for sources that fill in any steps that were skipped. Finally, global learners should skim through the entirety of a book chapter or lecture slideshow ahead of time so they can get an overview of the information (1996).

4 Suggestions for Teaching Algorithm Run-time Complexity Analysis

Reviewing complaints from students and instructors and analyzing the topic using multiple learning theories leads to a general idea of the format by which algorithm run-time complexity analysis should be taught. Giving the motivation for learning the topic is a good place to start. Students need to know why they are learning about algorithm run-time complexity analysis in the first place. Discuss how algorithm analysis can be used to compare multiple approaches to a problem and choose the solution that is most scalable and efficient. Explaining the motivation will appeal to learners from the diverging quadrant in Kolb's learning styles.

The next step could be to explain the key facts and terminology related to the subject. This could include mathematical topics, such as logarithms and the notation used. This will benefit assimilating learners from Kolb's styles and sensing learners from Felder's styles. Then, begin teaching the steps to follow when computing an algorithm's complexity. Make use of worked examples. This will ensure that students reach an action stage of learning as outlined in APOS theory. It is also an example of concrete experience from Kolb's experiential learning cycle, and it will appeal to active and sensing learners from Felder's styles.

Once students have a solid understanding of the process of analyzing the run-time complexity of an algorithm, begin to show them how the concept is applied in the real world. One way to do this would be to have students analyze multiple algorithms that solve the same problem and see how they compare. This reaches the active experimentation stage in Kolb's cycle. It appeals to converging learners from Kolb's styles and intuitive learners from Felder's styles.

The final step would be to relate the topic of algorithm run-time complexity analysis to other topics. This is an opportunity to transition to the next topic. If the class covers data structures, now would be a good time to discuss how certain structures are optimized to improve the run-time complexity for specific operations. This relates to the reflective observation stage in Kolb's experiential learning cycle. It also benefits accommodating learners from Kolb's learning styles

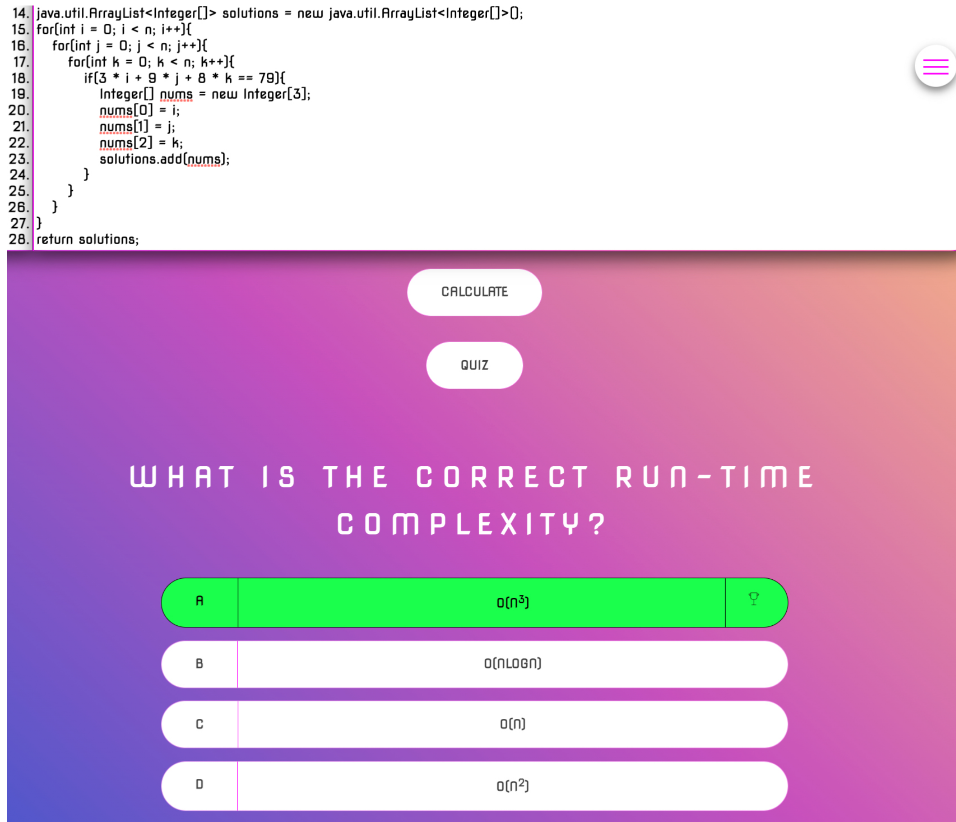

Figure 4: Application for Teaching Algorithm Run-time Complexity Analysis

Source: Own Image

It is important to vary the presentation style of the algorithm run-time complexity analysis material to appeal to the varying learning styles that students might have. Verbal learners can be supported through the standard lecture style. Graphs such as the one shown in Figure 1 and visualizations would benefit visual learners. The application developed in conjunction with this paper, shown in Figure 4, is an example of a way to support active learners. It also allows students to practice worked examples to get a better understanding of the process of performing algorithm run-time complexity analysis. Occasionally taking the time to review what has been covered so far would be helpful for reflective learners. Teaching the material in a logical order helps sequential learners, while giving an outline of the material before beginning benefits global learners.

5 Conclusion

Algorithm run-time complexity analysis is widely cited as one of the most difficult topics in a typical data structures and algorithms course (Fouh et al., 2016). Instructors give varying reasons for this, but some of the most common are that students lack the needed proficiency with key mathematical concepts, students find the material to be too abstract, and that the style of presentation of the material is unengaging. Examining the topic using multiple learning theories reveals additional flaws in the way that the material is taught. Some courses do not spend enough time explaining the steps needed to perform algorithm analysis, skipping over the action stage in APOS theory (Kay and Wong, 2018). Kolb's theory of experiential learning shows also shows that there are stages of the learning process and learning styles that need to be better addressed. Felder's learning styles reveal ways that the presentation style of course materials could be improved. Students can be more successful at learning algorithm run-time complexity analysis if the teaching material accommodates varying learning styles and addresses all stages of learning.

References

Beaubouef, T. (2002). Why computer science students need math. 34(4):57–59.

Cater, M. (2011). Incorporating learning styles into program design. `https://lsuagcenterode.wordpress.com/2011/08/16/incorporating-learning-styles-into-program-design/`.

Dubinsky, E. and McDonald, M. A. (2001). APOS: A constructivist theory of learning in undergraduate mathematics education research. In *The Teaching and Learning of Mathematics at University Level*, pages 275–282. Springer.

Farghally, M. F., Koh, K. H., Shahin, H., and Shaffer, C. A. (2017). Evaluating the effectiveness of algorithm analysis visualizations. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 201–206.

Felder, R. M. and Soloman, B. A. (2000). Learning styles and strategies. `https://www.engr.ncsu.edu/wp-content/uploads/drive/1WPAfj3j5o5OuJMiHorJ-lv6fON1C8kCN/styles.pdf`.

Fouh, E., Farghally, M., Hamouda, S., Koh, K. H., and Shaffer, C. A. (2016). Investigating difficult topics in a data structures course using item response theory and logged data analysis. *International Educational Data Mining Society*.

Harris, J. (2006). Using computer algebra systems in the teaching of analysis of recursive functions. *Journal of Computing Sciences in Colleges*, 21(5):116–122.

Hein, G. E. (2019). Constructivist learning theory. `https://www.exploratorium.edu/education/ifi/constructivist-learning`.

Howard, R. A., Carver, C. A., and Lane, W. D. (1996). Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: Tying it all together in the CS2 course. In *Proceedings of the twenty-seventh SIGCSE Technical Symposium on Computer science Education*, pages 227–231.

Huang, S. (2020). What is big o notation explained: Space and time complexity. `https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c`.

Kallia, M. and Sentance, S. (2017). Computing teachers' perspectives on threshold concepts: Functions and procedural abstraction. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pages 15–24.

Kay, A. and Wong, S. H. S. (2018). Discovering missing stages in the teaching of algorithm analysis: an APOS-based study. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, pages 1–5.

Mcleod, S. (2017). Kolb's learning styles and experiential learning cycle. `https://www.simplypsychology.org/learning-kolb.html`.

Parker, M. and Lewis, C. (2014). What makes big-o analysis difficult: understanding how students understand runtime analysis. *Journal of Computing Sciences in Colleges*, 29(4):164–174.

Shi, H. and Shi, H. (2009). Introducing formal derivation into the design and analysis of algorithms. In *2009 4th International Conference on Computer Science & Education*, pages 1322–1324. IEEE.