Mini-Course: The Time Complexity Compendium

David Payne
Undergraduate in Computer Science
University of Florida
Gainesville, Florida
dpayne1@ufl.edu

***Abstract*—The Time Complexity Compendium is aimed at late-high school or early undergraduate Computer Science students who already have some programming experience (such as Programming Fundamentals 1 or similar). The course is designed to give students an introduction to Time Complexity and Big O Notation. Most importantly, the course teaches students how to analyze the time complexity of algorithms under various circumstances, such as loops, recursion, and data structures.**

Keywords*—*Time Complexity, Website, Loops, Recursion, Undergraduate

I. Introduction

The Time Complexity Compendium is intended to be a 1 credit hour mini-course accessed entirely on a free website consisting of 7 modules related to teaching time complexity evaluation with Big-O notation.

The course can be accessed with the following link: https://www.cise.ufl.edu/~dpayne1/TimeComplexity/index.html

The motivation behind the course is due to identifying a need among myself and my peers for a more comprehensive teaching of time complexity. Time complexity is briefly mentioned in courses such as COP3530 Data Structures and Algorithms and COT3100 Discrete Structures, but is only dedicated a few lectures in the broader scope of those courses. As a result, many students struggle with time complexity and evaluating the time complexity of code snippets, and even their own code.

Though time complexity is poorly understood amongst students, it still is an extremely important concept to the overall computer science curriculum because employers expect their developers to build efficient algorithms. Additionally, programmers must always be thinking about optimizing their code when they are developing software. A poor grasp of time complexity will be a detriment to our future software engineers, which is why The Time Complexity Compendium aims to resolve this illiteracy.

Students who take this course will have the advantage of being more competitive in the workplace due to increased awareness of efficiency in the code they write. They will be able to compute the time complexity of any given code snippet, a tool that will help them in their future courses as well as in coding interviews.

II. Description of Course

As previously mentioned, the course is hosted entirely online a website is available at the link above to browse. The course is split into 7 modules, each covering a new topic in time complexity and expanding on the previous topic. Each module has its own webpage and is easily accessed through the module table at the top of the webpage. Each module will have instructional content via text, visuals, code snippets, and graphs. Each module will also have one or more activities and an assessment at the end of each module.

A. Learning Objectives

There are four main learning objectives for The Time Complexity Compendium:

- Understand the mathematical functions and growth rates that underlie time complexity in software engineering.
- Understand and apply the fundamentals of Big-O notation.
- Analyze code snippets involving loops, recursion, data structures, and special cases using Big-O notation.
- Compare and evaluate the time complexity of different code snippets to determine which is more efficient.

B. Topics/Sequence Outline

The course is split into 7 distinct modules, each module expanding from the last and introducing more concepts for students to understand. The modules are intended to cover everything there is to know about time complexity and evaluating the time complexity of code snippets. The modules are as follows:

Module 1: Functions and Growth Rates
Module 2: Big O Notation Introduction
Module 3: Simple Loops and Nested Loops
Module 4: Loops Special Cases
Module 5: Recursion
Module 6: Recursion Special Cases
Module 7: Data Structures

For a breakdown of each module, its learning objectives, activities, learning challenges, and standards alignment, please view the course schedule section of the website.

C. Expected Learning Outcomes

After completing this course, students should be able to evaluate the time complexity of code snippets that involve loops, nested loops, recursion, and data structures using Big-O notation, and be able to evaluate which code snippets are more efficient in terms of growth rates of their Big-O notation. Through this, students will have an increased awareness of the efficiency of the code that they write in their future courses and in the workforce.

D. *Course Target Population*

The target audience for The Time Complexity Compendium is beginning programmers. As such, the course is aimed at undergraduate students who are interested in computer science and have prior programming experience. Ideally, these students will have just finished Programming 1 or 2 and have not taken their Data Structures and Algorithms course yet. High schoolers with programming experience are also welcome to take the course, though the course is geared more toward undergraduates.

1) *Ideal Learner*

The Ideal Learner is a student who is passionate about and enjoys programming. They are typically high achievers and have a strong programming background and conceptual understanding. It is important that they also have a good mathematical background as they will need to understand growth rates and compare functions for this course. Fueled by a strong desire to learn computer science, they will enjoy the course and understand abstract concepts like time complexity easily.

2) *Typical Learner*

A typical learner enjoys programming to a moderate degree, but they are not incredibly passionate about it nor do they have the strongest possible understanding of programming, although passing Programming 1. Despite this, they are motivated by a desire to attain their degree in computer science and are possibly interested in honing their skills in programming. They may not remember the mathematical functions necessary for the course, but they do not find the course difficult, while still providing an adequate challenge for them.

3) *Non-Typical Learner*

A non-ideal learner is motivated by the money and success that can come from a computer science degree. They dislike programming and do not understand programming concepts well. They are only taking this course for the degree and are unconcerned about understanding the content.

E. *Frequency of Course*

The mini-course is a self-paced course. This means that students can complete the modules at their own pace in accordance with their understanding of each topic. Since modules build on each previous topic, it is highly recommended that students take and pass the summative evaluation assessment at the end of each module before moving on to the next, even though all modules are unlocked from the start. Because this course is online, students can always reference and review previous modules if they fail to understand a certain topic.

We recommend students spend approximately 1 hour per day on this course for approximately 2 weeks, doing about 1 module per weekday. However, students can complete the course at whichever pace they desire.

F. *Session Structure*

There are 7 modules, each intended to take approximately 45 minutes, including the summative evaluation assessment. The modules will be a mix of instructional content via paragraphs, diagrams, visuals, and code examples. Interspersed between the instructional content will be activities that check the student's understanding of the content that came before it. These activities give students the opportunity to think critically about the content and reinforce learning by allowing them to apply what they have learned.

At the end of each module, there will be a self-evaluation assessment consisting of questions related to the module. The summative evaluation assessment will be a mix of multiple choice, true/false, and short answer questions. The summative evaluation assessment will be graded automatically, and students will be able to see their scores immediately after submitting the assessment with feedback. Students will be able to retake the summative evaluation assessment as many times as they want, and the highest score will be recorded.

G. *Course Placement in the undergrad curriculum*

Ideally, this course should be taken after COP3502C Programming Fundamentals 1, however, it can be taken after COP3502C Programming Fundamentals 2. It should be taken before COP3530 Data Structures and Algorithms. It is intended to be a lightweight online course, and students may choose to take it alongside Programming Fundamentals 2 or in the summer before taking Data Structures and Algorithms.

H. *Required prior knowledge and pre-requisite classes*

The only prerequisite for this course is COP3502C Programming Fundamentals 1, or an equivalent introductory programming course. High schoolers opting to take this course must have prior programming experience.

Prior to taking this course, students should be familiar with C++, as the code snippets used in the course will be written in C++. Students should be familiar with loops, nested loops, and recursion in C++ and be able to trace a program from start to finish. A strong background in math and mathematical functions is recommended but not required for understanding the functions that underlie the time complexity of code snippets.

I. *Instructional Approach*

The course is designed as a self-paced online course. Each module contains a written lecture with code snippets and other relevant visuals with activities interspersed between lecture content, and an assessment at the end of each module.

An online, self-paced course is ideal for teaching time complexity because it is a difficult concept that relies on the understanding of the student to progress and expand on topics. For example, a student can make sure they understand Big-O notation before moving on to using Big-O notation to evaluate the time complexity of code snippets. Being online is intended for the convenience of students.

J. *Technology and Resources Used in the Course*

To complete this course, students only need to have an internet connection and access to a web browser on a device capable of connecting to the internet. This was done to increase the accessibility of the course and allow a wide variety of students to access the course. They will need to understand C++, but the course does not require writing or compiling any code, and therefore does not require additional technology.

The course makes use of HTML, CSS, JavaScript, and PHP to create the website, but the student does not have to be familiar with these languages or frameworks to use the course. PHP and JavaScript are used to grade students' answers on the activity or assessment, and HTML and CSS are used to display the information on the webpage.

K. *Prior Research considerations*

The course reduces the cognitive load in each module by chunking information into subsections and further into activities that pertain to that section [1]. To elaborate, rather than having one activity at the end of the module that covers all of the course content, activities are interspersed in the subsections of the course module, and they only pertain to the content that was covered in that subsection of the module. This is helpful in reducing the cognitive load of information, by reinforcing small chunks of information at a time rather than the entire module. Experts tend to reduce cognitive load by chunking information as well [1].

Additionally, another design principle that has been identified is allowing students to control the amount of scaffolding that they receive [2]. In this course, this design principle is implemented within some of the activities, such as Activity 3.2. This is because the student has the option to press a button to get hints on the question and has another option to press a button to get the answer to the question. The student may opt to not use the hints, use the hints but not the answers, or use the hints and the answers. In this way, the amount of scaffolding is set by the students, and they can change the amount of scaffolding that they receive at any time.

One other element of CS Education Teaching Best Practices is the implementation of feedback into assignments and activities [3]. The Time Complexity Compendium incorporates feedback into all its activities and assessments. When a student submits a question on the activity, or submits their assessment, they automatically receive feedback about their answer, the correct answer, and how to arrive at the correct answer or hints for the student. This feedback is instrumental in allowing students to understand proper time complexity evaluation and Big-O notation.

Another one of the best practices addressed by this mini course is the principle of accessibility [3]. The course promotes accessibility by being hosted online, and not requiring any other software or setup other than a standard web browser and internet connection that students already have. The online course is highly accessible and no external resources, such as a textbook or other software, are needed. The Time Complexity Compendium is therefore highly accessible.

L. *Broadening Participation in Computing Issues and Challenges*

The course addresses challenges in broadening participation in computing first by allowing the entire course to be read through the use of screen readers for the visually impaired. Since the course is a website, disabled students can use their screen readers to listen to the course content out loud and still complete the course in its entirety.

Additionally, the course content is written with accessible diction and relatively informal vocabulary. This was intended to engage students with its accessibility while also being approachable for students who do not have English as their first language. Therefore, students of many different backgrounds should be able to understand and engage with course content even though the course is written entirely in English.

III. Course Content

There are 7 modules of course content. Each module consists of lecture content delivered through text, code snippets, and other relevant visuals with activities peppered throughout the content. There is an assessment at the end of each module.

This section will cover an example lesson in Module 3, Simple Loops and Nested Loops.

A. *Selecting course topics*

Selecting course topics for The Time Complexity Compendium involved starting from the highest-level application of time complexity, data structures, and working backward to understand what students will need to know before building up to analyzing the time complexity of data structures. I wanted to strip Time Complexity down to its core components in an effort for students to understand time complexity better. As a result of this, I realized that the course needs to start at the absolute beginning level, with the first module only covering the common mathematical functions that comprise the time complexity of common code snippets. This is because this course intends to avoid any possible confusion regarding time complexity, and understanding and memorizing the growth rates of functions is instrumental in understanding and comparing time complexity.

From there, I thought that the next logical progression in teaching time complexity would be to introduce Big-O notation, without code at first. This was to get students used to Big-O notation before relating it to code snippets, to reduce cognitive load. Then, I introduce code snippets with loops in the next module. From there, I cover loop special cases, introduce recursion, recursion special cases, and finally cover data structures. By ending with data structures, students are ready to enter their Data Structures and Algorithms course as they will understand the mechanism behind evaluating the time complexity of specific data structures.

B. *Example Lesson*

This example lesson covers Module 3: Simple Loops and Nested Loops. To access this lesson, click the link below.

https://www.cise.ufl.edu/~dpayne1/TimeComplexity/modules/module3/

The example lesson starts with a section titled "Simple Loops." In this subsection, students are introduced to connecting code snippets with for loops and while loops. The three basic for loop and while loop time complexities are introduced, O(n), O(log(n)), and O(1) loops, which are vital for students to be able to recognize. Code snippets are shown as examples of these common time complexity loops. To solidify the recognition of these loop conditions for students, a matching activity is given at the end of the section for students to use what they have learned and recognize the time complexity of each loop snippet. After this activity is a table with common loop conditions and their corresponding time complexities that the student can study and memorize.

The next section of the lesson is titled "Non-nested multiple loops," and this section teaches the rules of when a student is presented with multiple non-nested loops via examples and rules. The three rules mentioned are 1) add non-nested terms, 2) drop constant multipliers, and 3) drop lower order terms, and an example is given for each. After this section is the second activity of the lecture, which is a scaffolded free-response activity consisting of 5 questions. The student must analyze the multiple non-nested loop code snippets and write the time complexity of the code snippet. This activity is scaffolded with hint buttons and the ability to reveal the answer if the student is stuck. The question gives feedback for correct and incorrect answers.

The last section is titled "Nested Loops," and it teaches students to multiply nested terms when given a nested loop. This section provides multiple examples of this concept.

The summary is titled "Play by the rules" and gives a short summary of the rules for evaluating the time complexity of loops. The student is then guided by the webpage to take the assessment by clicking the link at the bottom of the webpage.

The assessment is a summative assessment consisting of 7 questions related to course content. The assessment questions are more difficult than the activity questions, and feedback is given after the student submits their answers, with their auto-graded score and the correct answers with their explanations.

C. *Learning Objectives of the Lesson*

The learning objectives of module 3: Simple Loops and Nested Loops are as follows:

- Compute the time complexity of code snippets involving simple loops and nested loops with Big O notation.
- Compare code snippets involving loops using Big O notation and evaluate which is more efficient.
- Understand that the loops that students write can affect performance.

D. *Larger Course Goals*

This lesson teaches students the basics of evaluating the time complexity of loops. This connects to the larger course goal of students understanding how to evaluate the time complexity of code snippets because they will be writing loops in their own code, and they will need to know how to evaluate the efficiency of their code to write more efficient code. Loops are also the main mechanisms of creating time complexity in code, which makes this lesson one of the most important of the course.

E. *Connection to CS2023 Framework*

This lesson and its objectives connect to the following National Standards outlined by the CS2023 Framework:

Al/Basic Analysis - Use big O notation formally to give asymptotic upper bounds on time and space complexity of algorithms.

Al/Basic Analysis - Use big O notation formally to give expected case bounds on time complexity of algorithms.

This is because assessing the time complexity of algorithms involving loops is taught in this module, and students give the expected time complexity of these algorithms using Big-O notation. Asymptotic upper bounds in this case means the mathematical functions that are identified in the Big-O notation format, such as O(n) or O(log(n)). Understanding how to compute the time complexity of loops will be instrumental in evaluating the time complexity of algorithms.

F. *Addressing Learning Challenges*

A challenge that I identified for the course is the struggle of helping students on an online course if they are stuck. This is a challenge for online courses because the course content is rigid, and students do not have an opportunity to ask questions or ask the teacher to elaborate on concepts.

The solution to this is to create a hint system and answer system scaffold. This scaffold allows the student to have the option to get a hint on a question in the activities if the student is struggling or needs extra help. The hint will be relevant to each question and give tips on how to evaluate the time complexity of the code snippet given, or another relevant hint to the question. If the student is still stuck, they can reveal the answer for the activities. Because this is a scaffold, this aid goes away during the assessments. Though students cannot get hints or the answer on the assessment as they complete the assessment, they will still receive feedback and how to approach the problems after they have submitted it.

Another challenge is teaching students how to analyze the time complexity of code snippets while keeping them engaged. Typical and non-typical learners tend to like learn-by-doing approaches and the instructor walking through their problem-solving techniques as they approach new questions.

To solve this problem, a scaffold is given where multiple examples are given during the instructional content that involves walking through example problems and showing typical and non-typical learners specifically how to approach problems. Engagement is satisfied through the inclusion of activities that break up the stream of lecture content, and therefore reduces cognitive load by following the general model of learning a subsection, doing an activity based on that subsection, and then moving on to the next subsection. In this way, students are not constantly reading and are engaged and applying the content they are learning.

A challenge for ideal learners is that they want to be challenged by the course content and may find typical content to be too easy. To solve this, questions in the assessments are created to be more challenging than the questions in the activities. This will keep ideal learners engaged in course content by challenging them and therefore they will still enjoy the course and pay attention.

G. *Broadening Participation in Computing Considerations*

The sample lesson addresses broadening participation in computing issues in multiple ways. First, the language used within the lesson is accessible and basic enough for students who have English as their second language to understand. The language used is informal and easy to interpret, which makes the course inviting for students of different cultures.

Second, since the course is online, it supports screen readers for students who are visually impaired but interested in computer science. Screen readers can read course content out loud, therefore making the entire lesson accessible to the visually impaired.

IV. COURSE ASSESSMENTS

A sample assessment can be completed by clicking on the following link:

https://www.cise.ufl.edu/~dpayne1/TimeComplexity/modules/module3/assessment.php

There are 7 assessments in this course, with one being at the end of each module. These assessments are all summative assessments, and are therefore all multiple choice, true/false, short answer, or select all that apply questions based on the course module that the student has just completed. The summative evaluation assessment will be graded automatically, and students will be able to see their scores immediately after submitting the assessment along with feedback. Students will be able to retake the summative evaluation assessment as many times as they want, and the highest score will be recorded. The questions will pull from a question bank, so that the student will typically get new questions for each attempt to prevent students from simply memorizing answers.

A. *Supporting Course Goals and Learning Outcomes*

These assessments support the overall course goals as performing well on the assessments shows mastery of course topics. For example, a student performing well on the assessment for simple loops and nested loops would then be considered to have mastery in evaluating the time complexity of simple loops and nested loops. Since this is part of the course's overall goal of analyzing the complexity of a given code snippet, assessments inherently support the course goals.

Additionally, the assessments relate to the learning outcomes as in each assessment there will be questions related to comparing the time complexity of different code snippets. Since one of the learning outcomes mentioned is increased awareness of the efficiency of code students write, comparing two code snippets will help students achieve this outcome by being able to select which code snippet is the more efficient one. The student will then experience a "knowledge transfer" when they write their own code. For example, when a student is coding, they may think of two potential approaches to solve a given coding problem. Because of these assessments, they can evaluate which approach is more efficient and implement the more efficient one within their code.

B. *Student Knowledge and assessing the difference between high, medium, and low-performing students*

Because these are summative assessments and contain one right answer per question, assessing what students know should be relatively simple as the instructor will be able to see student's scores and which questions they got wrong. Since each question focuses on a different topic, the instructor and the student can tell which areas of the module that they need to study more and go back to study or relearn those sections of the module. An advantage of using summative assessments is that students are presented with automatic feedback, and it can help them determine what parts of the module that they struggle with. They can then go back to that part of the module and study it until they believe they have mastered it. Additionally, student knowledge can be modeled on a scale from 0 to the max number of points possible, for each assessment, which is convenient for the instructor of the course.

By extension, proficiency can be measured based on student scores. A score of 50% or below can be considered low proficiency. A score of 90-51% can be considered medium proficiency. A score of 100% can be considered high proficiency. This is because the assessments are challenging, and typical learners are not expected to achieve 100% on their first attempt at the assessment. 100% on a student's first attempt will typically be achieved only by ideal learners, and at this point, we can classify them as high-performing students. Consistent scores within the medium proficiency range would classify a student as medium-performing, and consistent scores within the low-performing range would classify a student as low performing.

The expectation for students on these assessments is to keep improving on each of their attempts at the assessment until they achieve a perfect score. There will be a question bank for each assignment so that students simply do not memorize answer choices. This makes assessments more of a challenge, however, there is low stress for students due to having multiple attempts on each assessment.

C. *Authenticity of Assessments*

The type of authenticity that these assessments align most with is disciplinary authenticity, which is a type of educational authenticity that David Shaffer has identified in his work [4].

Disciplinary authenticity refers to an educational task being representative of the discipline that the educational task is preparing the learner for. The assessments in this course align with this type of authenticity because as students become software engineers, they will have the evaluate the time complexity of their own code and compare different approaches and implementations to figure out which is the most efficient algorithm to implement. This is exactly what the questions on the assessments will ask, so high disciplinary authenticity is present.

Another type of authenticity that the assessments align with is authentic assessment, which is the idea that students should learn something while taking the test. This is something that occurs in our assessments due to the challenge of our assessments making the students think beyond what they have learned to evaluate the time complexity of code snippets or solve a given problem, and therefore the students are learning something as they complete the assessment.

V. Peer Feedback

The feedback that I received from my peers was mostly positive. My peers were impressed that I made a whole website demo for this project and all activities were on the website ready to go with feedback and scaffolding. They praised the layout and organization of the website as being easy to navigate and understand, which is something I considered while making the website because navigation and organization are important for an online course as students do not have someone to help them if they get stuck. They also felt as though the topic of my mini-course was important for beginning programmers, which is ideal because I am glad my course focuses on something important and needed in the field. They also felt that the lecture was easy to understand and liked that I walked through examples of how to solve individual problems as part of the learning content.

The weaknesses of the sample lecture and the suggestions for improvement came from the activities that were hosted on the website. One of my peers mentioned that the matching activity was not as functional as it could be. For example, it does not let you undo a match and does not tell you if you got a match right or wrong. Additionally, another peer stated that the second activity with the scaffolded loop free-response should have had more challenging questions and could have been longer.

To address the feedback on the matching, I can make several changes to the activity to increase user experience. I will have to implement more JavaScript functions within the code that allow the user to undo their previous actions as well as trigger a prompt to appear which tells them if their match was right or wrong. This will help students understand the activity better and reduce potential frustration. I will also elaborate more in the instructions for the activity so there is as little confusion as possible.

To address the feedback on the second activity that involved free-response questions, I plan on increasing the length of my activities in the future, and making the questions in the activities progressively get more challenging. I will also increase the variety of questions, like the variety seen in my assessment. This can involve comparing two code snippets and determining which is more efficient based on time complexity, for example. I think it was an important point to bring up that this activity could have been longer, because I believe that for students to remember a concept, they need to experience repetition of the concept. This is especially the case with the time complexity evaluation of code snippets.

VI. Next Steps

The next steps for this mini-course involve fleshing out the rest of the modules, which involves creating each module's lecture content, activities, and assessments. Once this is done, I will need to implement a database to store student grades and completion of activities so they can receive a final grade in the course. Additionally, the content will have to go through several stages of review, testing, and feedback, with the content being tweaked after each stage of review before this can become an actual course offered at a university.

Unfortunately, I am not too interested in continuing nor will I be teaching this course in the near future due to my interests lying in software engineering rather than Computer Science education. However, I sincerely hope that the University of Florida can one day offer a course similar to this one so incoming software developers can have a greater understanding of time complexity.

References

[1] National Research Council. 2000. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. Washington, DC: The National Academies Press. https://doi.org/10.17226/9853.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed.,

[2] Elliot Soloway, Mark Guzdial, and Kenneth E. Hay. 1994. Learner-centered design: the challenge for HCI in the 21st century. interactions 1, 2 (April 1994), 36–48. https://doi.org/10.1145/174809.174813

[3] Guzdial, Mark. (1999). Software-Realized Scaffolding to Facilitate Programming for Science Learning. Interactive Learning Environments. 4. 10.1080/1049482940040101.

[4] Shaffer, D. W., & Resnick, M. (1999). " Thick" Authenticity: New Media and Authentic Learning. *Journal of interactive learning research, 10*(2), 195.