International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Understanding Time and Space Complexity in Algorithms

Naga Sai Krishna Mohan Pitchikala

Masters in Computer Science, University of Texas at Dallas, Texas, USA
Email: *nxp180022[at]utdallas.edu*

Abstract: ***The important concept in algorithms is time and space complexities. This is the critical aspect in choosing the one algorithm over the other. Calculating the time and space complexities are important as they are the measures of efficiencies of the algorithms and based on them certain algorithms are chosen over the other. Time complexity refers to the amount of time the program takes to run according to the input size, whereas the space complexity refers to the number of additional resources that are needed to run the program. In this paper we will define each of them theoretically and explain why they are important in data structures and algorithms and how to understand them practically.***

Keywords: time complexity, space complexity, algorithm efficiency, data structures, program resources

1. Introduction

An Algorithm is a well - defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output[1]. This means that an algorithm is a clear step by step process that we define to solve a problem. It is a formal way of writing the steps to solve a program, which are sometimes written in a language understandable by computer. They should be detailed correctly to be implemented in a computer. Once implemented the computer program will take any input value and will give out the desired output for that input value. This is where the whole logic to solve the problem lies in. Based on the the algorithms pseudo code is written before implementing the actual program. Pseudo code is an informal way to telling how we will be writing the program code. Pseudo code is in the human readable format and any person can write the actual code from the pseudo code by following the syntax of the specific chosen programming language. While algorithm is a decoding the logic that maps every input to the desired output, Data structures are the ones which indicate the type of storage used in program to store the variables. Now, when writing the program choosing the right algorithm and data structures is very important to write an efficient code that works as expected in all scenarios.

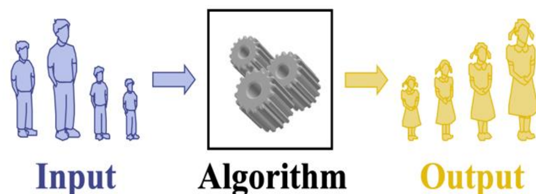

Figure 1: Definition of an algorithm[2]

So, measuring the performance of the algorithm used in program as well as the data structures is important to determine the efficiency of the program. This is where time and space complexity come in. The time and space complexity are considered as the fundamental metrics in measuring the time and space complexity of the algorithm. While time complexity measures the number of operations an algorithm performs as a function of input size, space complexity measured the amount of additional memory needed to execute the program. We will now discuss each of them in detail.

2. Time Complexity

Time complexity is defined as the amount of time taken by an algorithm to complete the process in terms of input size. Time complexity is measured in 3 cases. Best case, Average case and in worst case scenarios.

Best Case: The best - case time complexity denotes the time taken by the algorithm to complete execution in best case scenario. This is denoted by Ω (N)

Average Case: The average - case time complexity denotes the time taken by the algorithm to complete execution in typical case. This is denoted by Θ (N).

Worst Case: The worst - case time complexity refers to the amount of time taken by the algorithm to complete execution in worst case scenario. This is denoted by O (N).

For time complexity we will be mostly looking at the best - case time complexity and the worst - case time complexity. These two metrics will tell us how long the program or algorithm will take to execute based on the input length. There are like 7 different values for each of them ranging from 1, $\log^n$, n, $n\log^n$, n^2, 2^n, n!.

While they look and sound something strange, it is very clear to understand them. They represent how the output grows with respect to the input. Let us consider each of them in detail.

1) O (1): Constant Time. This indicates that the time taken by the program to run is constant and does not change with the input size.
2) O ($\log^n$): Logarithmic Time. This indicates that the time taken by the program to run grows at the rate that follows a logarithmic pattern according to the input.
3) O (n): Linear Time. This indicates that the time taken by the program to run grows at the rate that is proportional to the input size.
4) O ($n\log^n$): This indicates that the time taken by the program to run grows at the rate that is proportional to the

Licensed Under Creative Commons Attribution CC BY

multiplication of input and its logarithmic. Which is basically larger than the liner time.

5) O (n^2): Quadratic Time. This indicates that the time taken by the program to run grows at the rate that is proportional to the quadratic of the input size.
6) O (2^n): Exponential Time. This indicates that the time taken by the program to run grows at the rate that is exponential of the input size.
7) O (n!): Factorial Time. This indicates that the time taken by the program to run grows at the rate that is factorial of the input size.

O (1) is considered as the good value for the time complexity metric while the O (n!) is considered as the bad value for the time complexity measured in any case (best, worst, average)

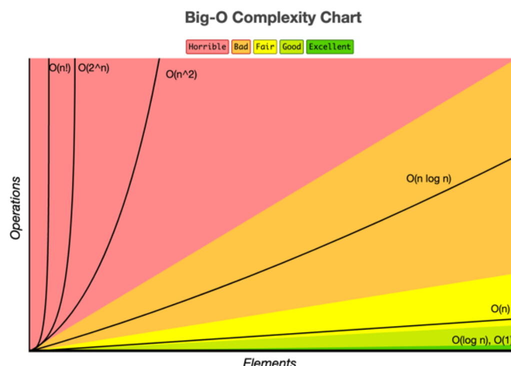

Figure 2: Big - O complexity chart[2]

For every algorithm that is being implemented based on the existing inventions or even if there is a new algorithm that is invented to solve a problem, time complexity is calculated first. This would inform the user about performance of the algorithm in real world scenarios. For example, when a new sorting algorithm is proposed by a person, the immediate action that they do is to evaluate the time complexity of them algorithm. If the proposed algorithm does not improve the time complexity over the ones that are already in market, then the proposed algorithm is not solving any problem for us efficiently, and it will not be materialized.

Calculating Time Complexity

Time complexity can be measured by the analyzing how the different parts of an algorithm work, like loops, recursion, and if - else statements. For example:

- In a loop for a single input value the loop will run 'n' times. 'n' being the size of the input. This means that the time complexity if the for loop is O (n), meaning for the input of size 'n' the runtime grows at the rate that is proportional to the size of the input.
- Now, consider a nested for loop. So, for each value in the outer loop, the inner loop will run 'n' times. This means that in total the time complexity of the nested for loop is O (n^2).

By analyzing how different parts of algorithm and how they run according to the size of input gives a clear idea on the run time of each individual part and the runtime of the whole algorithm.

3. Space Complexity

While time complexity is the primary performance metric in algorithms, the other one is the space complexity. This is also equally important as it tells us how much storage is required to execute the algorithm and in the places where memory has limitation, this plays a vital role.

Space complexity measures the amount of additional memory needed for the algorithm to complete execution based on size of the input it receives. When we say additional memory, we focus on the extra memory the algorithm requires to run instead of the memory needed to store the input itself. While we generally measure the memory in bytes, it is often easier to think about it in terms of how many integers the algorithm uses. The final measure does not depend on the exact number of bytes but on the overall pattern of memory usage.

People sometimes overlook space complexity because the memory used may be small or straightforward. However, in some cases, the memory usage can be just as important as the time it takes for the algorithm to run. Space complexity is usually measured in Big - O notation. This is similar to the measure of worst - case time complexities. When the space complexity is O (1) it means that the space complexity is constant and is not dependent on the input size. Similarly, when we say that the space complexity is O (n) it means that the memory required to execute the algorithm grows proportionally with the input size, and similarly other metrics in O (n)

Calculating Space Complexity

Space complexity cab be measured by analyzing the amount of additional memory an algorithm needs to run, beyond the memory required to store the input data itself. For example:

- Memory used for variables like integers, booleans, floats, and other basic data types is constant, regardless of the size of the input. This is known as O (1) space complexity, meaning it uses a constant amount of space irrespective of the data size
- If an algorithm uses an array to store data for each element of input, the space complexity would be O (n), meaning that the size of the array grows according to the size of the input.

By understanding how data is stored in different data structures in an algorithm we can calculate the memory needed by individual data objects in the algorithm and the memory needed to run the program.

4. Understanding the practicality of space and run time complexities

In real world scenarios, programs usually start with handling small amounts of data but gradually they often need to handle large amounts of data confined with the limitations from both time and memory. When a program uses an algorithm with high run time complexity, like for example O (n^2) (where time grows quadratically as the input increases) or O (2^n) (where time grows exponentially with input size), it can become very slow as the input size gets large. This means that for tasks like sorting or searching through a big dataset, using a slow

Licensed Under Creative Commons Attribution CC BY

algorithm can lead to noticeable delays, which will degrade the user experience and reduce the system's overall efficiency. Space complexity is also a big factor, especially in environments with limited memory, like smartphones, embedded devices, or smaller computers. If an algorithm requires too much memory, it can cause the program to crash or make the system run slowly as this program consumes all the available memory. Even if the algorithm is fast, it is not desirable if it needs more memory than what is available in the device.

By understanding the run time complexity and space complexity programmers can make smart decisions about which algorithms to use. This will help them in choosing the right algorithms to run the program efficiently both in terms of memory and space. When we choose the right algorithms based on the limitations and the complexities, the programs become more efficient and can handle larger tasks which are better suited for real - world applications.

5. Conclusion

Time and space complexity are fundamental concepts in computer science that will help us in figuring out how efficient an algorithm is. Time complexity tells us how long an algorithm takes to run (especially in the worst - case scenario) while space complexity tells us how much memory the algorithm will need. Knowing how to measure these complexities is important for picking the right algorithms that run fast and don't use too much memory, which is crucial when developing applications that will have a constant increase in the input data and when working with devices that have limited resources.

Sometimes, making a program faster may require using more memory, or reducing memory usage might slow the program down. Programmers need to evaluate the trade - off between time and space limitations, so they can create a software that works well. Finding the right balance between these two metrics is essential for building programs that are both quick and efficient with memory.

References

[1] Introduction to Algorithms by Thomas H. Cormen
[2] Data Structures and Algorithms in Java by Roberto Tamassia and Michael T. Goodrich
[3] https: //www.bigocheatsheet. com/
[4] Data Structures and Algorithms in Java by Michael T. Goodrich

Licensed Under Creative Commons Attribution CC BY