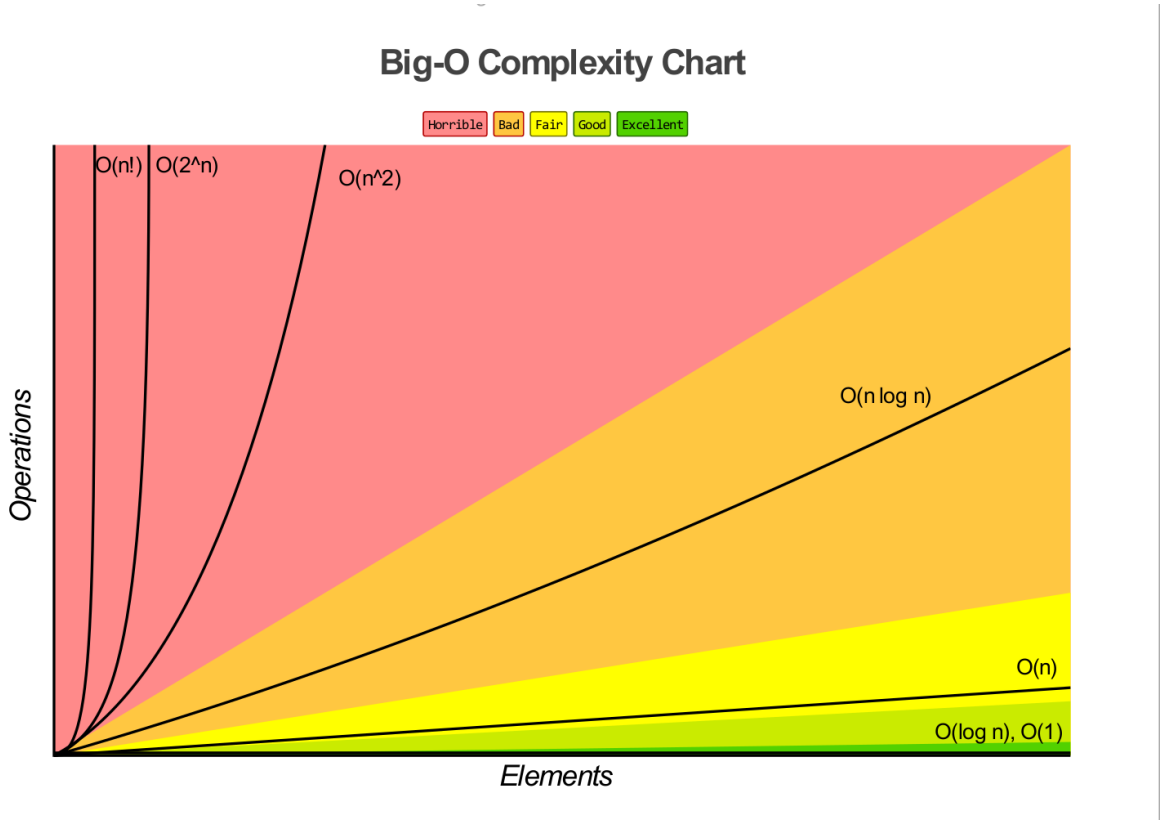
Space and Time complexity Analysis

Created time	@February 20, 2023 3:58 AM
Last edited time	@February 21, 2023 8:08 PM
Status	Completed
URL	



What is Time And Space complexity

Space and time complexity are two measures used to analyze the performance of an algorithm.

ℹ️ **Note: Time complexity ≠ Time taken**

The phrases "the time required" or “Time taken” may be more intuitive to some people, as it directly implies that the time complexity of an algorithm is a measure of the amount of time it takes to complete. However, the phrase "the measure of time required" is more precise, as it emphasizes that time complexity is not an absolute measure of time, but rather a relative measure of the number of operations performed by an algorithm as a function of the size of the input data.

What is Time Complexity?

Time complexity is a measure of how much time an algorithm takes to run as the input size increases. It is usually expressed in terms of the number of operations an algorithm performs, as a function of the input size. The goal is to analyze how the running time of the algorithm scales as the size of the input grows.

What is Space Complexity?

Space complexity is a measure of how much memory an algorithm uses as the input size increases. It is also expressed in terms of the input size, and the goal is to analyze how the memory usage of the algorithm scales as the input grows.

How to calculate?

The space and time complexity is often expressed using “Big O notation” This notation provides an upper bound on the growth rate of the algorithm as the input size increases.

Steps to calculate time and space complexity
To calculate the time and space complexity of an algorithm,

you can follow these general steps:

1. Identify the input size: This is the variable that determines the size of the input to the algorithm. For example, if you're analyzing a sorting algorithm, the input size might be the number of elements in the array to be sorted.
2. Count the number of operations: This involves identifying the number of basic operations that the algorithm performs as a function of the input size. For example, for a sorting algorithm, you might count the number of comparisons, swaps, or other basic operations that the algorithm performs.
3. Express the growth rate: Use Big O notation to express the growth rate of the algorithm as the input size increases. This involves simplifying the expression for the number of operations and identifying the most significant term in the expression.
4. Analyze the space complexity: Follow the same steps as above, but count the amount of memory used by the algorithm, rather than the number of operations.

what does it mean by relatively insignificant?

suppose we are given an unsorted “n” size array and we have to perform an operation that requires us to first linear search a key and then do some operation according to it to do that “operation” we are running a nested loop of 1 to n so the

equation we will get polynomial i.e. O(x + x^2) as we are mostly optimizing our algorithm to perform well in the worst case we have to take bigger input in consideration as the input grows we can see the significance of each terms dominating to at point that including lesser doesn't even matter like in x + x^2 for the input array

with size 1, we get (1 + 1^2) = (1 + 1) = 2

with size 10, we get (10 + 10^2) = (10 + 100) = 110

with size 100, we get (100 + 100^2) = (100 + 10000) = 10100

with size 10000 we get(10000 + 10000^2) = (10000 + 100000000) = 100010000

with size 100000000 we get (100000000 + 100000000^2) = 10000000100000000

you get the idea the more number will grow the less important the less significant term will become so as a generalized way we only care about the highest significant term.

Why do we use Big O notation?

Big O notation is a commonly used way of expressing the time and space complexity of algorithms. One of the main reasons it's used is that it provides a simple way of expressing the upper bound on the growth rate of the algorithm as the input size increases. This makes it easy to compare the performance of different algorithms, even if they have different implementations.

Another reason for using Big O notation is that it's a relatively simple and standard way of expressing complexity, which allows for easier communication and collaboration among developers and researchers. While other notations and methods exist for expressing algorithmic complexity, Big O notation is widely used and understood in the computer science community.

Other ways to express algorithmic complexity.

While Big O notation is a commonly used way to express algorithmic complexity, there are other notations and methods that can be used as well. i.e.

- Big Omega (Ω) notation: This provides a lower bound on the growth rate of an algorithm, as the input size increases. It is often used in situations where we want to

show that an algorithm performs at least as well as a certain level, regardless of the input size.

- Big Theta (Θ) notation: This provides both an upper and lower bound on the growth rate of an algorithm, as the input size increases. It is often used to show that an algorithm has a specific, well-defined complexity.

note: while identifying the number of operations we mostly get very complicated mathematical equations to take a general idea of growth, we ignore the constant and lesser significant terms This is because, as the input size grows, the most significant terms dominate the behaviour of the function or algorithm, while the less significant terms become relatively insignificant.

Conclusion:

space and time complexity is a measure used to analyze the space and time complexity of an algorithm, which helps us to compare different algorithms used for the same operation and also helps us to optimize and design our algorithms for the worst, average and best-case input to show those we use the asymptotic notation i.e Big O notation(Worst case), Theta notation(average case) and Omega notation(best case). these three provide us with the upper bound, the lower bound and the average growth of a function. having all of this notation in our pocket mostly when we refer to the space or time complexity of a function we often talk about the worst case (big O notation) because it gives us a general idea that how our function will grow in the big picture

(src) made in collaboration with ChatGPT