

Level 1: Time and Space Complexity

1. Why Do We Need Time and Space Complexity?

When multiple algorithms solve the same problem, they may not perform equally well. Some algorithms are fast but consume more memory, while others are slow but memory-efficient. Time and space complexity help us compare algorithms independently of hardware and programming language. They allow us to predict performance when input size becomes very large.

2. What is Time Complexity?

Time complexity describes how the execution time of an algorithm increases as the size of input increases. It does not measure actual time in seconds. Instead, it focuses on the growth rate of operations. The goal is to understand scalability, not exact runtime.

3. Big-O Notation

Big-O notation expresses the upper bound of an algorithm's running time. It represents the worst-case scenario, ensuring that the algorithm will never perform worse than this bound. Big-O ignores constants and lower-order terms to simplify analysis.

4. Common Time Complexities

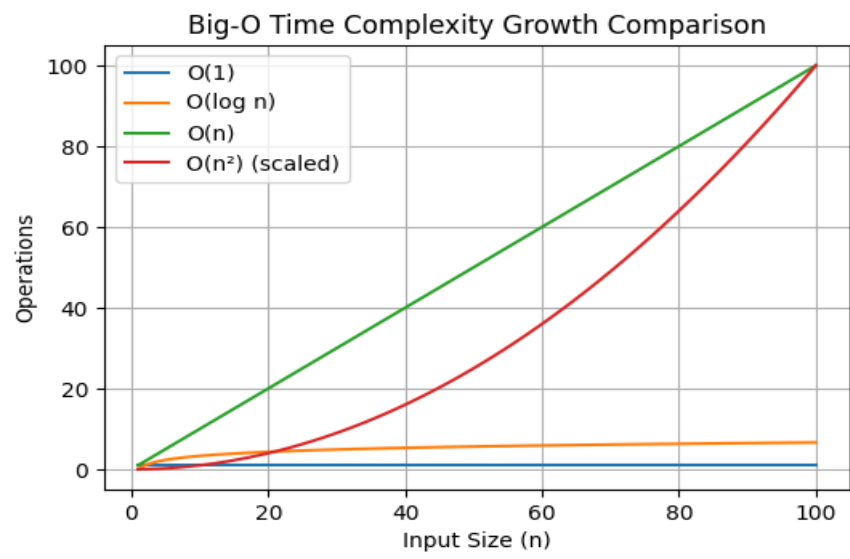
$O(1)$ – Constant Time: The algorithm takes the same amount of time regardless of input size. Example: accessing an element in an array by index.

$O(\log n)$ – Logarithmic Time: The input size is reduced at each step. Example: binary search in a sorted array.

$O(n)$ – Linear Time: Execution time grows proportionally with input size. Example: traversing a list once.

$O(n^2)$ – Quadratic Time: Execution time grows with the square of input size. Example: nested loops over the same data.

5. Visual Comparison of Time Complexities



6. How to Calculate Time Complexity

To calculate time complexity, follow these rules: Drop constant factors, keep the dominant term, and multiply complexities for nested loops. Always consider the worst-case input scenario.

7. Space Complexity

Space complexity measures the amount of extra memory an algorithm uses relative to input size. If an algorithm uses a fixed number of variables, it has $O(1)$ space complexity. If it creates additional data structures proportional to input size, it has $O(n)$ space complexity.

8. Time vs Space Tradeoff

Often, improving time efficiency requires using extra memory. For example, hash tables trade space for faster lookups. Understanding this tradeoff is crucial when designing real-world systems.