Day 3 : Data Structures & Algorithms (DSA) with Java – Full Course

Time and Space Complexity (Big O Notation Explained)

@drnimishadavis

Why Analyze an Algorithm?

- To compare two algorithms that solve the same problem.
- Helps identify which one is faster or uses less memory.
- Hardware, compiler, and OS may differ — so we use mathematical analysis instead of time in seconds.

What is Time Complexity?

- Time complexity = Total number of basic operations executed by an algorithm.
- Expressed as a function of input size n.
- Focuses on growth rate, not actual execution time.

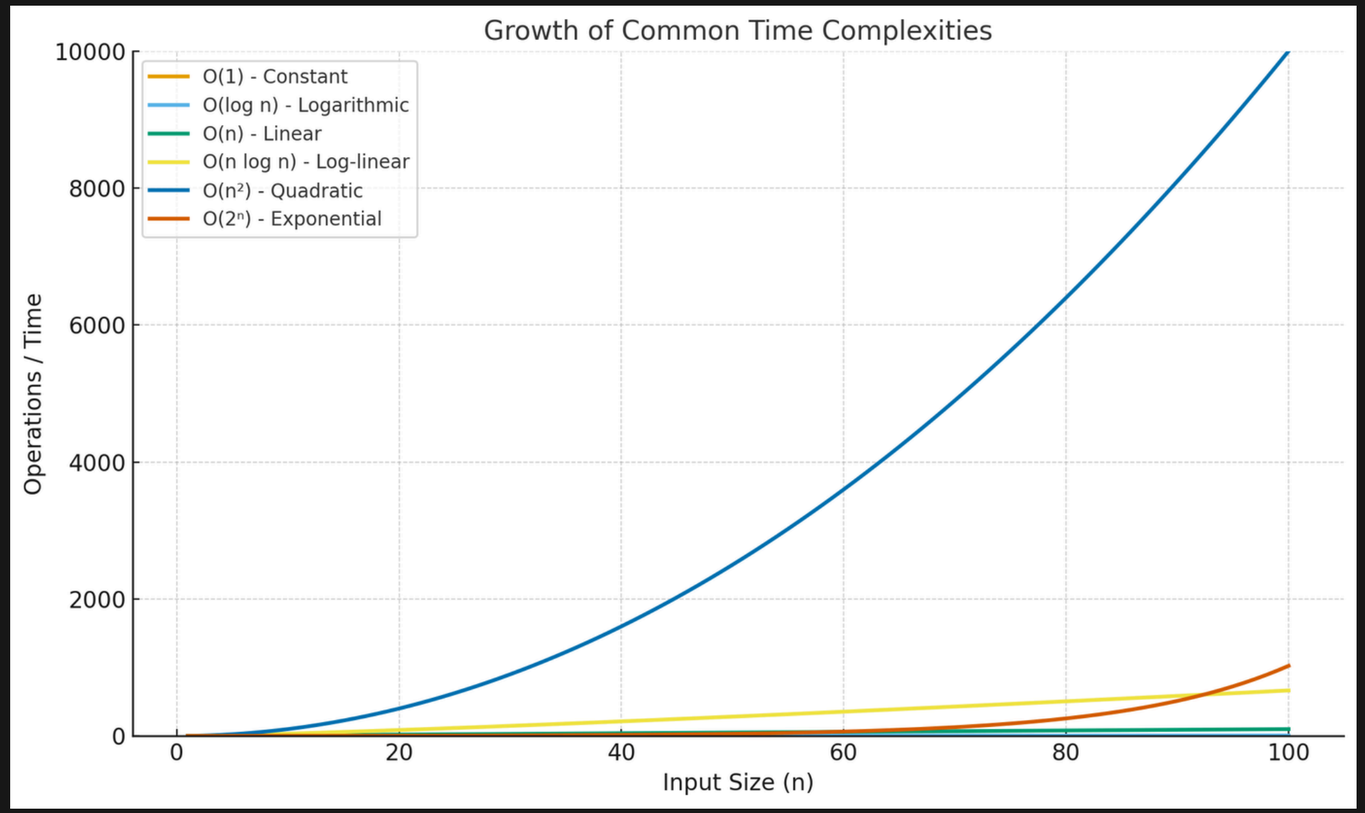
Example:

If an algorithm takes 3n + 5 steps → Time complexity is O(n).

Common Time Complexities

Complexity	Name	Example
O(1)	Constant Time	Accessing array element
O(log n)	Logarithmic	Binary Search
O(n)	Linear	Linear Search
O(n log n)	Log-Linear	Merge Sort, Quick Sort
$O(n^2)$	Quadratic	Bubble Sort, Insertion Sort
$O(2^n)$	Exponential	Recursion on subsets
O(n!)	Factorial	Traveling Salesman Problem

Visual Growth Graph



@drnimishadavis

Example: Linear Search

```
int linearSearch(int arr[], int x) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == x)
            return i;
    }
    return -1;
}
```

Example: Binary Search

```
int binarySearch(int arr[], int x) {
    int low = 0, high = arr.length - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == x)
            return mid;
        else if (arr[mid] < x)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

Types of Time Complexity

- Best Case: Minimum time (ideal scenario).
- Average Case: Expected time for random input.
- Worst Case: Maximum time taken (guaranteed upper bound).

What is Space Complexity?

- Space complexity = Total memory used by an algorithm.
- Includes input, variables, recursion stack, temporary storage.
- Expressed as a function of input size n.

Example:

- An algorithm using an array of size n → O(n) space.
- Fixed variables only → O(1) space.

Example: Space Complexity

```
int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}
```

Big O, Big Ω, Big Θ

- O (Big O): Upper bound → Worst case.
- Ω (Big Omega): Lower bound → Best case.
- Θ (Big Theta): Tight bound → Average case (both upper & lower same).

Comparison Table

Algorithm	Best Case	Average Case	Worst Case	Space
Linear Search	O(1)	O(n/2)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)

Practical Tip: How to Analyze Code

- Count loops → each loop = O(n).
- Nested loops → multiply complexities.
- Sequential statements → take the highest O().
- Ignore constants and lower terms.

Summary

✅ Time complexity → how fast your algorithm runs.
✅ Space complexity → how much memory it uses.
✅ Big O notation → standard for comparing efficiency.
✅ Always aim for algorithms with lower growth rate.

Practice Task

Write algorithms and find time complexities for:

1. Sum of all elements in an array.
2. Checking if a string is palindrome.
3. Finding factorial (loop vs recursion).

Thank You

@drnimishadavis