🔷 What is Time Complexity?

Time Complexity means **how much time** a program or function takes **as the input size grows**.

✅ Real-life Example:

Imagine you're searching for your name in a list of people.

- If the list has 5 names, you may find it quickly.
- If the list has 10,000 names, it will take longer.

Time Complexity helps estimate **how much longer it takes as the list gets bigger**.

🔷 What is Space Complexity?

Space Complexity means **how much memory (RAM)** a program uses **as the input size grows**.

✅ Real-life Example:

Think of writing names in a notebook.

- If you write 1 name, it takes 1 line.
- If you write 1000 names, you need more pages.

Space Complexity tells how much **extra memory** your code needs when input increases.

📊 Common Time Complexities (From Best to Worst):

Complexity	Name	Example	Meaning
O(1)	Constant Time	Accessing `arr[0]`	Always takes same time no matter input size
O(log n)	Logarithmic Time	Binary Search	Cuts the problem in half each time
O(n)	Linear Time	Loop through a list	Time grows linearly with input size
O(n log n)	Log-linear Time	Merge Sort, Quick Sort	Faster than quadratic but slower than linear
O(n²)	Quadratic Time	Nested loop (e.g., bubble sort)	Time grows **very fast** with input size
O(2ⁿ)	Exponential Time	Recursive Fibonacci	Time doubles every time; very slow
O(n!)	Factorial Time	Solving permutations	Extremely slow as input grows

---

📦 Common Space Complexities:

Complexity	Meaning	Example
O(1)	Uses same amount of space	Swapping variables
O(n)	Space grows with input	Storing input in an array
O(n²)	Space grows as square of input	2D matrix storage

✅ Code Examples:

🧠 Example 1: O(1) Time

```
def get_first_element(arr):
    return arr[0]
```

- No matter how big `arr` is, it just returns the first item → **O(1)**

🧠 Example 2: O(n) Time

```
def print_all(arr):
    for item in arr:
        print(item)
```

- Goes through the entire list → **O(n)**

🧠 Example 3: O(n²) Time

```
def print_pairs(arr):
    for i in arr:
        for j in arr:
            print(i, j)
```

- Every element pairs with every other → **O(n²)**

🧠 Example 4: O(log n) Time (Binary Search)

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

- Cuts the array in half each time → **O(log n)**

🧮 Tips to Analyze Time & Space

1. **Loops**: Each loop = O(n). Nested = O(n²).
2. **Recursion**: Consider how many times it recurses.
3. **Auxiliary space**: Extra arrays, hash maps, etc.
4. **Ignore constants**: O(2n) = O(n), O(5n²) = O(n²)

🧠 Summary

Term	Meaning
Time Complexity	How long your code takes
Space Complexity	How much memory your code uses
O(1)	Constant, super fast
O(n)	Grows with input
O(n²)	Bad for big inputs
O(log n)	Good! Cuts problem in half
