📘 TIME & SPACE COMPLEXITY — FULL DETAILED NOTES.

🟦 1. What is Time Complexity?

Definition:

Time Complexity is **the rate at which the time taken by an algorithm increases with the increase in input size**.

❌ Not equal to actual time taken. Why?

Because **actual time depends on machine configuration**, example:

Machine	Time taken
Old Windows	2 sec
New MacBook	1 sec

Same code → different machines → different execution time.
➡ Therefore, we measure *how fast time grows*, not actual seconds.

🟦 2. Why do we need Time Complexity?

In interviews:

- They **never run your code on a machine**.
- They analyze:
  ✔ Time Complexity
  ✔ Space Complexity
  ✔ Logic & Optimality

Time Complexity allows us to compare codes **independent of hardware**.

🟦 3. Big-O Notation

Big-O describes the upper **bound** (worst case) of growth.

Format:

```
O(expression)
```

Example:

```
O(n), O(n²), O(log n), O(1) ...
```

We compute an **approximation** of growth — not exact steps.

🟦 4. Counting Steps: Basic Example

Code:

```
for(int i = 1; i <= 5; i++) {
    cout << "Raj";
}
```

Steps:

- Initialization
- Comparison
- Print
- Increment
  ➡ repeated 5 times → approx 3 × 5 operations.

Time complexity:

```
O(15)
```

But we **never** write constants → final:

```
O(1)  ❌
O(n) ✔ (because loop runs n times)
```

🟦 5. THREE IMPORTANT RULES for Computing Time Complexity

✔ Rule 1: Always consider Worst Case

Why?

- Systems must be designed for **maximum load**.
- E.g., linear search worst case = element not found.

✔ Rule 2: Avoid Constants

Example:

```
4n³ + 3n² + 8
```

For n = 10^5:

- $4n^3 = 4 \times 10^{15}$
- $3n^2 = 3 \times 10^{10}$ (irrelevant)
- 8 is negligible

Final:

$O(n^3)$

✔ Rule 3: Avoid Lower Order Terms

Example:

$n^3 + n^2 + n \rightarrow O(n^3)$

Reason: n^3 dominates heavily for large n.

🟦 6. Types of Time Complexity

🔹 Best Case:

Minimum operations
Example:
Marks = 10 → first if gets executed → 2 operations

🔹 Worst Case:

Maximum operations
Marks = 70 → last else executed → ~4 operations
➡ **Used in all complexity analysis**

🔹 Average Case:

```
(best + worst) / 2
```

Rarely used in coding interviews unless explicitly asked.

🟦 7. Bigger Notations (Theory Only)

Notation	Meaning
O(...)	Upper bound / Worst case

Ω(...)	Lower bound / Best case
Θ(...)	Tight bound / Average case

Interview Tip:
Striver says — *Only Big-O is asked in interviews.*

🟦 8. Time Complexity Examples

➤ Example 1 — Nested Loop

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        // constant time operation
    }
}
```

Explanation:

- Outer loop → n times
- Inner loop → n times for each i
  Total:

```
n × n = n²
Time Complexity = O(n²)
```

➤ Example 2 — Triangular Loop

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j <= i; j++) {
        // constant
    }
}
```

Runs like:

```
1 + 2 + 3 + ... + n = n(n+1)/2
```

Apply rules:

- Ignore constants → $n^2/2$
- Ignore 1/2 → n^2

Final:

```
O(n²)
```

🟦 9. Common Complexities (important list)

Complexity	Meaning
O(1)	Constant
O(log n)	Logarithmic
O(n)	Linear
O(n log n)	n log n
O(n²)	Quadratic
O(n³)	Cubic
O(2ⁿ)	Exponential
O(n!)	Factorial

You will see these throughout DSA.

🟥 10. SPACE COMPLEXITY

✔ Definition:

Total memory consumed =

```
Auxiliary Space + Input Space
```

🟩 A. Auxiliary Space

Extra space used to solve a problem
Example:

```
int a, b;
int c = a + b; // c is extra
```

Auxiliary space = 1 variable → O(1)

🟩 B. Input Space

Memory used to *store the input*

Example:

```
int a, b; // input variables
```

Input space = 2 variables → still **O(1)**
(Because a few variables don't depend on n)

If array input:

```
int arr[n] → O(n)
```

🟥 11. Very Important Interview Rule

❌ Never modify input directly

Instead of:

```
b = a + b; // modifies original input
```

Do:

```
int c = a + b;
```

Why?

- In real software systems, input data may be used elsewhere.
- Modifying input is considered **bad practice**.

➡ Interviewer may reject you if you mutate input without permission.

🟦 12. Space Complexity Examples

Example 1

```
int a, b, c;
```

Space Complexity:

```
O(1)
```

Example 2

```
int arr[n];
```

Space Complexity:

```
O(n)
```

Example 3

Recursion:

- Each recursive call uses stack space
- Example: factorial recursion → O(n)

🟧 13. Competitive Programming Tip

Servers can run:

```
1 second ≈ 10⁸ operations
```

If time limit = 1 sec
Your code must run in:

```
O(10⁸)
```

If time limit = 2 sec:

```
2 × 10⁸ operations
```

NOTE:
Do NOT do:

```
(10⁸)² → WRONG
```

🟩 14. FINAL SUMMARY (Perfect for revision)

✔ Time Complexity

- Not actual time
- Rate of growth
- Use Big-O
- Worst case always considered
- Ignore constants & lower-order terms

✔ Space Complexity

- Auxiliary + Input
- Never modify input
- Arrays → O(n)
- Recursion adds stack frames

✔ Important results

- Nested loops = multiply
- Triangular loops = n(n+1)/2
- Most servers = 10^8 operations/second