Time and Space Complexity – Summary (Apna College Lecture 12)

1. What is Time Complexity?

Time complexity measures how the execution time of an algorithm increases with input size. It focuses on the number of operations rather than actual runtime.

2. Big O, Omega, Theta Notations

• Big O (O): Worst-case scenario (upper bound)
• Big Ω (Omega): Best-case scenario (lower bound)
• Big Θ (Theta): Average-case or tight bound

Most common in interviews: Big O.

3. Types of Time Complexities

O(1): Constant – Accessing an array index
O(n): Linear – Traversing an array
O(log n): Logarithmic – Binary Search
O(n^2): Quadratic – Nested loops
O(2■): Exponential – Recursive Fibonacci
O(n!): Factorial – Brute-force permutations

4. Time Complexity in Loops

• Single loop → O(n)
• Nested loops → O(n^2)
• Loop with i *= 2 → O(log n)

5. Recursion Time Complexity

• Based on recurrence relations
Example:

```
int fact(int n) {
if(n == 0) return 1;
return n * fact(n - 1);
} // O(n)
```

Fibonacci recursive → O(2■)

6. Space Complexity

Space used by algorithm includes variables, data structures, and call stack.
• Array of size n → O(n)
• Iterative with few vars → O(1)
• Recursion → O(stack depth)

7. Why It Matters?

- Predict performance under constraints
- Required in interviews
- Optimization = Better real-world efficiency

Key Takeaways

- Always check input size constraints.
- Use Big O for clear communication.
- Optimize both time and space.