**✅ Theory-Based Questions on Time & Space Complexity**

1. **What is time complexity? Why is it important?**  
   → Time complexity describes how the runtime of an algorithm grows with input size n. It helps compare algorithms' efficiency.
2. **What is space complexity? How is it different from time complexity?**  
   → Space complexity is the amount of memory an algorithm needs to run. Time = CPU cycles; Space = memory used.
3. **Explain Big O notation.**  
   → Big O represents the worst-case scenario — how fast time/space grows as input increases.
4. **What do Ω (Omega) and Θ (Theta) represent?**  
   → Ω: Best-case complexity  
   → Θ: Average-case complexity (tight bound)
5. **Why do we consider worst-case time complexity in Big O?**  
   → Because it ensures the algorithm never performs worse than that in any case.
6. **Compare O(1), O(log n), O(n), O(n log n), O(n²) with examples.**  
   → O(1): Accessing array index  
   → O(log n): Binary Search  
   → O(n): Linear Search  
   → O(n log n): Merge Sort  
   → O(n²): Bubble Sort
7. **What is the time complexity of searching in a sorted array using Binary Search?**  
   → O(log n)
8. **What’s better: O(n log n) or O(n²)? Why?**  
   → O(n log n), as it grows slower than O(n²) with large inputs.
9. **If an algorithm has two nested loops running n times, what is the time complexity?**  
   → O(n²)
10. **If you call a function recursively n times, and each call does O(1) work, what is the time complexity?**  
    → O(n)

**💻 Code-Based Time Complexity Analysis**

1. **What is the time complexity of this loop?**

java

CopyEdit

for(int i = 0; i < n; i++) {

System.out.println(i);

}

→ **O(n)**

1. **What’s the complexity here?**

java

CopyEdit

for(int i = 0; i < n; i++) {

for(int j = 0; j < n; j++) {

System.out.println(i + "," + j);

}

}

→ **O(n²)**

1. **What is the time complexity of Binary Search?**

→ **O(log n)**

1. **Find the time complexity:**

java

CopyEdit

for(int i = 1; i < n; i = i \* 2) {

System.out.println(i);

}

→ **O(log n)**

1. **What is the time complexity of Merge Sort?**

→ **O(n log n)**

1. **What is the space complexity of the Merge Sort algorithm?**

→ **O(n)** (due to temporary arrays created during merge)

1. **What is the space complexity of an in-place Quick Sort?**

→ **O(log n)** (due to recursion stack in best/average case)

1. **If a function takes an int[] arr of size n and uses a second array of same size, what is the space complexity?**

→ **O(n)**

1. **What is the space complexity of the following recursive function?**

java

CopyEdit

void fun(int n) {

if(n == 0) return;

fun(n - 1);

}

→ **O(n)** (due to recursion stack)

1. **Which is more efficient: O(n log n) time with O(n) space or O(n²) time with O(1) space?**

→ Depends on the problem. Usually, O(n log n) time is better — we often trade space for time.