PART 1: TIME & SPACE COMPLEXITY (FOUNDATION)

1️⃣ Time Complexity — Sub-Topics (IN THIS ORDER)

1.1 What is Time Complexity?

- Why we measure time
- Input size `n`
- Worst case vs average case
- Big-O notation (idea, not math)

1.2 Common Time Complexities

- O(1)
- O(log n)
- O(n)
- O(n log n)
- $O(n^2)$
- $O(2^n)$ (intro only)

1.3 How to Calculate Time Complexity

- Single loop
- Nested loops
- Loops with different variables
- Recursion (basic idea only)

1.4 Comparing Solutions

- Why O(n) is better than $O(n^2)$
- When slower solution is acceptable

2️⃣ Space Complexity — Sub-Topics

2.1 What is Space Complexity?

- Input space
- Auxiliary space
- In-place algorithms

2.2 Common Space Cases

- Constant space O(1)
- Linear space O(n)
- Recursion stack space

---

3 Complexity — What YOU Should Practice

DO NOW

- Identify complexity of given code
- Compare two solutions
- Predict growth as input increases

DO NOT

- Memorize formulas
- Overthink recursion space now

Complexity Learning Loop

```
Kunal lecture → self-explain → 5–8 small questions → DONE
```

Once done, **move on**. Don't get stuck here.

ESSENTIAL TIME COMPLEXITY PROBLEMS (DO ONLY THESE)

PART 1: LOOP-BASED COMPLEXITY (MOST IMPORTANT)

You don't need specific "questions" here — you need to be able to do **this skill**.

You MUST practice:

1. **Single loop**
2. `for (int i = 0; i < n; i++)`

→ O(n)

3. **Nested loops**

```
4. for (int i = 0; i < n; i++)
5.   for (int j = 0; j < n; j++)
```

→ O(n^2)

6. **Dependent loops**

```
7. for (int i = 0; i < n; i++)
8.   for (int j = i; j < n; j++)
```

→ O(n^2)

9. **Log loop**

```
10.  while (n > 1) n /= 2;
```

→ O(log n)

If you can analyze these confidently, you're good.

PART 2: ESSENTIAL RECURRENCE RELATIONS (FROM KUNAL)

From your screenshot, **DO ONLY THESE TYPES**:

MUST-DO RECURRENCES

1 Linear recursion

```
T(n) = T(n - 1) + c
```

→ O(n)

Purpose: understand simple recursion growth.

---

2 Linear + work

```
T(n) = T(n - 1) + n
```

→ O(n^2)

Purpose: see accumulation of work.

3️⃣ Divide & conquer (basic)

```
T(n) = 2T(n/2) + n
```

→ O(n log n)

Purpose: understand merge sort.

4️⃣ Single subproblem

```
T(n) = T(n/2) + n
```

→ O(n)

Purpose: binary search–style thinking.

5️⃣ Slight variation (OPTIONAL but useful)

```
T(n) = 2T(n/4) + n
```

→ O(n)

Purpose: intuition that not all D&C = n log n.

? 5 ESSENTIAL PRACTICE QUESTIONS (TIME COMPLEXITY)

□ Question 1: Single Loop (Baseline)

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

Tasks:

1. What is the time complexity?
2. What is the space complexity?
3. Does `sum` affect complexity?

□ This checks if you understand **linear growth**.

□ Question 2: Nested Loop (Classic)

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(i + j);
    }
}
```

Tasks:

1. Time complexity?
2. What if `j = i` instead of `0`?
3. Does printing change Big-O?

□ This checks **quadratic intuition**.

□ Question 3: Logarithmic Loop

```
int count = 0;
while (n > 1) {
    n = n / 2;
    count++;
}
```

Tasks:

1. How many times does the loop run?
2. Time complexity?
3. Space complexity?

□ This tests **log n thinking**.

□ Question 4: Linear Recursion

```
void func(int n) {
    if (n == 0) return;
    func(n - 1);
}
```

Tasks:

1. Write the recurrence relation.
2. Time complexity?
3. Space complexity? (IMPORTANT)

□ This introduces **recursion stack space**.

□ Question 5: Divide & Conquer (Key One)

```
void func(int n) {
    if (n <= 1) return;
    func(n / 2);
    func(n / 2);
}
```

Tasks:

1. Write the recurrence.
2. Draw the recursion tree (levels).
3. Time complexity?
4. Space complexity?

□ This connects directly to **merge sort logic**.

□ PART 2: ARRAYS — PROPER STRUCTURE WITH SUB-TOPICS

Arrays is **NOT one topic**. It is **a container of techniques**.

PHASE A: ARRAY BASICS (YOU START HERE)

A1. Fundamentals

- What is an array?
- Indexing (0-based)
- 1D arrays
- 2D arrays

A2. Traversal Techniques

- Forward traversal
- Reverse traversal
- Nested traversal (2D)

A3. Basic Operations

- Insert (conceptual)
- Delete (conceptual)
- Update
- Search (linear)

A4. Simple Patterns

- Max / min
- Count elements
- Sum / average

□ QUESTIONS FOR PHASE A

(Only these from Kunal sheet)

- Build Array from Permutation
- Concatenation of Array
- Running Sum of 1D Array
- Richest Customer Wealth
- Shuffle the Array
- Kids With the Greatest Number of Candies
- How Many Numbers Are Smaller Than the Current Number
- Create Target Array in the Given Order
- Count Items Matching a Rule
- Cells with Odd Values in a Matrix
- Matrix Diagonal Sum
- Flipping an Image
- Find Numbers with Even Number of Digits
- Transpose Matrix

- Add to Array-Form of Integer
- Maximum Population Year
- Reshape the Matrix
- Determine Whether Matrix Can Be Obtained by Rotation

PHASE B: TWO POINTERS (NEW TECHNIQUE)

B1. Concept

- What are two pointers?
- Left & right pointer
- When to move which pointer

B2. Patterns

- Shrinking window
- Swapping
- Partitioning

□ QUESTIONS FOR PHASE B

- Remove Duplicates from Sorted Array
- Sort Colors
- Rotate Array
- Minimum Cost to Move Chips to the Same Position

PHASE C: PREFIX SUM / KADANE

C1. Prefix Sum

- Prefix array
- Range sum
- Avoid recomputation

C2. Kadane’s Algorithm

- Max subarray sum
- Reset logic

□ QUESTIONS FOR PHASE C

- Maximum Subarray
- Product of Array Except Self

PHASE D: SLIDING WINDOW

D1. Fixed Window

- Window size constant

D2. Variable Window

- Expand / shrink window

□ QUESTIONS FOR PHASE D

- Jump Game (later)
- (More will come from curated list)

PHASE E: HASHING WITH ARRAYS

E1. HashMap Basics

- Frequency counting
- Seen before logic

E2. Prefix + Hashing

- Subarray problems

□ QUESTIONS FOR PHASE E

- Two Sum
- Number of Good Pairs
- Lucky Number in a Matrix
- Find N Unique Integers Sum up to Zero

PHASE F: NOT ARRAYS (SKIP NOW)

- House Robber → DP
- Find First and Last Position → Binary Search
- First Missing Positive → Advanced index mapping
- Max Value of Equation → Advanced optimization