## **Universal Coding**

| **Step** | **What to Ask Yourself** | **Action** | **Why This Matters** |
| --- | --- | --- | --- |
| **1. Identify Problem Type** | Is it about **searching**, **sorting**, **counting**, **optimizing**, or **combining data**? | Categorize it:  - Search problem → Think Hash/Two Pointers/Binary Search  - Order required → Sorting or Heap  - Frequency/count → Hash Map  - Range/sum queries → Prefix Sum / Sliding Window  - Dynamic decisions → DP | Problem type narrows the pool of possible algorithms instantly. |
| **2. Spot Constraints & Data Size** | What is **n**? How big is the input?  - n ≤ 100 → O(n²) might be fine  - n ≤ 10⁵ → Aim for O(n log n) or better  - n ≤ 10⁶ → O(n) or O(log n) only | Compare input size vs. time limit (≈ 10⁸ ops/second). | Quickly rules out slow algorithms. |
| **3. Plan Brute Force First** | Can I solve it in the simplest possible way without worrying about complexity? | Write the brute force mentally (O(n²) or worse). | This gives a baseline to optimize from. |
| **4. Optimize by Reducing Loops** | Can I replace nested loops?  - Replace search in loop → Hashing (O(1))  - Replace multiple scans → Prefix Sum / Sliding Window (O(n))  - Need min/max while processing → Heap / Monotonic Stack | Choose algorithm with least complexity that still works. | Turns O(n²) into O(n log n) or O(n). |
| **5. Pick Final Approach** | Which is the **best balance** between time complexity and implementation ease? | - Justify to interviewer:  1️⃣ Brute force complexity  2️⃣ Why it fails for large n  3️⃣ Chosen optimal complexity  4️⃣ Trade-offs | Shows structured thinking, not guesswork. |

## 

## 

## 

## 

## 

## 

## 

## **Problem-Solving Flow**

### **Quick Complexity Cheat Table**

| **Problem Trait** | **Likely Optimal Approach** | **Complexity Target** |
| --- | --- | --- |
| **Finding max/min/kth** | Linear scan or Heap | O(n) or O(n log k) |
| **Checking duplicates** | Hash Set | O(n) |
| **Sorted data needed** | Sort (Quick/Merge) | O(n log n) |
| **Range sums/averages** | Prefix Sum / Sliding Window | O(n) |
| **Frequent lookups** | Hash Map / Binary Search | O(1) / O(log n) |
| **Graph traversal** | BFS/DFS | O(V+E) |
| **Shortest path** | Dijkstra/Floyd/Bellman | O(E log V) or O(V³) |
| **Dynamic decisions** | DP | O(n) to O(n²) |