Code and Complexity's – Problem Pattern Recognition Guide

(Adapted for Interviews & Competitive Programming)

📌 Keep this handy while practicing — but remember, during interviews it's all about what's in your head!

Step 1 – Start with the Constraints

Small Input Size ($n \leq 20$)

- Brute force is fine here.
- Backtracking and recursive enumeration shine.
- Exponential complexity (2^n, n!) is acceptable.
- Explore all possible combinations or permutations without fear.

Moderate Input Size ($10^3 \leq n \leq 10^6$)

- ❌ Skip brute force — it'll be too slow.
- Aim for O(n) or O(n log n) solutions.
- Use techniques like two pointers, greedy algorithms, heaps, or dynamic programming.

Very Large Input Size ($n \geq 10^7$)

- ❌ Even O(n) might be too slow.
- Target O(log n) or O(1) solutions.
- Consider binary search, mathematical shortcuts, and precomputed formulas.

Step 2 – Decode the Input Format

Trees (General / Binary / BST)

- Use DFS (all paths, recursive, preorder/inorder/postorder) or BFS (level-order, shortest path in unweighted trees).
- Pay attention to parent-child relationships and special tree properties.

Graphs (Nodes + Edges)

- BFS → shortest path.
- DFS → connected components.
- Union-Find → connectivity checks, grouping problems.
- Topological sort → dependency resolution.

2D Grids / Matrices

- DFS/BFS for "islands" style problems.
- Union-Find for connected regions.
- DP for pathfinding and counting problems.
- Mind the movement rules (4-dir, 8-dir).

Sorted Arrays

- Binary search.
- Two pointers.
- Greedy choices.

Strings

- **Two pointers** → palindrome checks.
- **Sliding window** → substring problems.
- **Trie** → prefix/word problems.
- **Stack** → bracket/parentheses validation.

Linked Lists

- Fast/slow pointers for cycle detection.
- Dummy node tricks for cleaner code.

Step 3 – Understand the Output Type

List of Lists (paths, subsets, combinations)

- Backtracking is your friend.
- Generate all choices using recursion (Take, Not Take).

Single Value (max profit, min cost, number of ways)

- Dynamic Programming for optimization.
- Greedy for quick optimal picks.
- Math-based counting when applicable.

Modified Structure (in-place edits)

- Two pointers for space-efficient changes.

Ordered Output (sorted tasks, ranked items)

- Custom sorting.
- Topological sorting.
- Heaps for maintaining order dynamically.

Step 4 – Keyword Triggers for Patterns

- **Dynamic Programming** → "Number of ways", "Max/Min sum", "Can you reach", "Longest/Shortest subsequence", "Optimal solution".
- **Two Pointers** → "Palindrome", "Sorted array", "Target sum", "Remove duplicates".
- **Heap** → "K largest/smallest", "Top K elements", "Median", "Priority queue".
- **Stack** → "Parentheses/brackets", "Valid expression", "Nested structure", "Undo/Redo".
- **Monotonic Stack** → "Next greater/smaller element".
- **HashMap** → "Frequency count", "Find duplicates", "Anagram check".
- **Trie** → "Word search", "Prefix matching".
- **Greedy** → "Minimum operations".

- **Union-Find** → "Connected components", "Number of groups".
- **Binary Search** → "Kth element", "Search in sorted data", "Minimize maximum", "**First/last occurrence".**
- **Bit Manipulation** → "XOR trick", "Single number", "Power of 2 check".
- **Math/Geometry** → "GCD/LCM", "Primes", "Angles", "Coordinates".
- **Game Theory** → "Optimal strategy", "Win/Lose prediction", "Minimax".
- **Sliding Window** → "Substring match", "Fixed/variable size subarray", "Max/Min window".