Bitflip's Leetcode Pattern Recognition Cheat Sheet (Version 1)

Print this out, keep it by you, but not during your interview 😉

Step 1: Check The Constraints

Small n (≤ 20):

- Brute force approaches are viable
- Backtracking and recursion
- Exponential time complexity (2^n, n!) is acceptable
- Try all possible combinations/permutations

Medium n (10^3 to 10^6):

- ❌ No brute force solutions
- Linear time O(n) or O(n log n) solutions
- Greedy algorithms
- Two pointers technique
- Heap-based solutions
- Dynamic programming

Large n (≥ 10^7):

- ❌ No linear time solutions
- O(log n) solutions only
- Binary search
- Mathematical formulas
- O(1) constant time approaches

Step 2: Analyze Input Format

Tree/Binary Tree/BST:

- Tree traversal (DFS/BFS)
- DFS for: all paths, recursive exploration, preorder/inorder/postorder

- BFS for: level-by-level, shortest path in unweighted tree
- Consider: tree properties, parent-child relationships

Graph (nodes + edges):

- BFS for shortest path
- DFS for connected components
- Union Find for "connected components" or "number of groups"
- Topological sort for dependencies

2D Grid/Matrix:

- DFS/BFS for "islands" problems
- Union Find for connected regions
- Dynamic programming for path problems
- Consider: 4-directional or 8-directional movement

Sorted Array:

- Two pointers technique
- Binary search
- Greedy approach

String:

- Two pointers for palindromes
- Sliding window for substrings
- Trie for word problems
- Stack for parentheses/brackets

Linked List:

- Two pointers (fast/slow)
- Dummy node techniques
- Cycle detection

Step 3: Analyze Output Format

List of Lists (combinations, subsets, paths):

- Backtracking is almost always the answer
- Generate all possibilities
- Use recursion with choice/no-choice pattern

Single Number (max/min profit, cost, ways, jumps):

- Dynamic Programming for optimization
- Greedy for local optimal choices
- Mathematical approach for counting

Modified Array/String (in-place operations):

- Two Pointers for in-place modifications

Ordered List (sorted sequence, valid task order):

- Sorting with custom comparators
- Topological Sort for dependencies
- Heap for maintaining order

Step 4: Keyword Pattern Recognition

Dynamic Programming Keywords:

- "Number of ways"
- "Maximum/minimum" + "sum/profit/cost"
- "Can you reach"
- "Longest/shortest subsequence"
- "Optimal" or "best"

Two Pointers Keywords:

- "Palindrome"
- "Sorted array"
- "Target sum"
- "Remove duplicates"

Heap Keywords:

- "K largest" or "K smallest"
- "Top K elements"
- "Median"
- "Priority"

Stack Keywords:

- "Parentheses" or "brackets"
- "Valid expression"
- "Nested structure"
- "Undo operations"

Monotonic Stack Keywords:

- "Next greater element"
- "Next smaller element"

HashMap Keywords:

- "Count frequency"
- "Find duplicates"
- "Anagram"

Trie Keywords:

- "Word search"
- "Word prefixes"

Greedy Keywords:

- "Minimum operations"

Union Find Keywords:

- "Connected components"
- "Number of groups"

Binary Search Keywords:

- "Kth element"

- "Search in sorted"
- "Minimize maximum"
- "First/last occurrence"

Bit Manipulation:

- "XOR" operations
- "Single number" problems
- “Power of 2”

Math/Geometry:

- "Greatest/Least Common Denominator"
- "Prime numbers"
- "Angle calculations"
- “Coordinate”

Game Theory:

- "Optimal strategy"
- "Win/lose scenarios"
- "Minimax"

Sliding Window:

- "Substring" with conditions
- "Subarray" with fixed/variable size
- "Maximum/minimum window"
- "Contains all"