This pattern recognition cheatsheet will help you solve 90% of all DSA problems in interviews (based on my interview experience with Uber, Google, Meta, Microsoft and Amazon...)

Most people think just knowing the DSA patterns will automatically make them a genius but problems don't come with pattern tags, you have to identify them yourself. Here's a better approach than generic cheatsheets you'll find out there:

1// When should you use Two Pointers?

→ If the question involves a sorted array or string, and you need to process or compare elements from both ends, especially when O(1) extra space is required.

→ Triggers: "Find pairs with sum X," "reverse in place," "remove duplicates," "partition array," or "merge sorted arrays."

→ You're usually aiming for O(n) time by moving left/right pointers, not doing brute-force nested loops.

2// When does Binary Search make sense?

→ If you're given a sorted array/list or need to find an optimal value/position quickly (much faster than O(n)), and the brute force is linear scans.

→ Triggers: "Find boundary/threshold," "first/last occurrence," "smallest/largest," or any "can you achieve X in minimal steps?"

→ If you're narrowing down search space each step (halving it), aim for O(log n) time.

3// When are Sets & Maps (Hashing) the right tool?

→ When you need constant-time lookups, quick frequency counts, or have to check for duplicates/pairs/groups.

→ Triggers: "Find unique," "detect duplicates," "group by," "anagrams," or "count frequency."

→ These help reduce O(n^2) scans to O(n) with efficient data access.

4// When does Sliding Window help?

→ If you're asked for the "longest/shortest/maximum/minimum" subarray or substring with certain properties, especially when the elements are consecutive or need to be processed in a range.

→ Triggers: “Find the max/min/unique in subarray of size k,” “longest substring with X property,” “contiguous subarray sum.”

→ You’re replacing nested loops (O(n^2)) with a moving window—usually O(n).

5// When to use Stack or Queue?

→ When the problem has nested/paired relationships, "next greater" or "previous smaller" logic, or simulates undo/redo, order of processing, or balanced symbols.

→ Triggers: “Evaluate expression,” “balance parentheses,” “next greater element,” “undo action,” or “process in order received.”

→ You typically need O(n) time with a linear pass and auxiliary structure.

6// When do you use Recursion?

→ When the data or the problem has a self-referential structure (trees, nested lists, problems defined in terms of smaller subproblems), or the process naturally splits into similar sub-tasks.

→ Triggers: “Divide into smaller parts,” “tree traversal,” “nested structure,” “Fibonacci,” “calculate X for left and right,” or “process all combinations.”

→ If you’d use a stack or see yourself writing nested loops for variable depths, recursion can get you O(depth) space and often O(n) time for trees.

7// When are Linked Lists the best fit?

→ If the input is already a linked list or you need to maintain order but also need O(1) insertion/deletion at head or tail, or require list manipulation without using extra space.

→ Triggers: “Delete node without head,” “reverse a list,” “find intersection,” “detect cycle,” “rearrange nodes.”

→ Optimizes problems where array indexing isn’t helpful and in-place operations matter.

8// When do Graphs show up?

→ If the input involves relationships between entities, networks, grids, pairwise connections, or anything talking about “paths,” “reachability,” “shortest/longest route,” or “can you get from A to B.”

→ Triggers: “Edge list,” “adjacency,” “island,” “maze,” “friends/connections,” “travel from/to,” “steps to goal.”

→ Expect O(V+E) time for traversals (BFS/DFS), and potentially higher for shortest paths, depending on the algorithm.

9// When does Heap/Priority Queue help?

→ When you need to repeatedly access the min/max element efficiently, merge sorted streams, maintain top-K items, or dynamically track order/statistics.

→ Triggers: “Find Kth largest/smallest,” “keep top/bottom K,” “running median,” “schedule tasks,” “merge K lists.”

→ Typically achieves O(log n) insertion/deletion and O(1) access to top element.

10// When is Sliding Window your friend?

→ When you need to find max/min/unique/target sum in subarrays or substrings of an array/string, with a focus on contiguous elements and optimizing from brute-force O(n^2) to O(n).

→ Triggers: “Longest/shortest subarray/substring,” “contiguous range,” “sum/product/window size,” “exactly/at most/at least K elements.”

11// When does Backtracking work best?

→ When you have to generate all possibilities, arrangements, permutations, or combinations—especially in puzzles or subset problems with exponential solution space.

→ Triggers: “Find all solutions,” “permutations/combinations/subsets,” “valid boards,” “choices at each step,” “undo and try next.”

→ Suits small inputs (n <= 15) and problems where you need to enumerate everything (NP-complete, like N-Queens, Sudoku).

12// When does Dynamic Programming (DP) apply?

→ If you need to make a series of choices, count ways, maximize/minimize results, and brute force has lots of repeated calculations, look for overlapping subproblems and optimal substructure.

→ Triggers: “Count number of ways,” “max/min value,” “partition,” “longest increasing subsequence/path,” “edit distance,” “minimum cost.”

→ Replaces exponential recursion with O(n) or O(n^2) solutions by caching/memoization.

13// When to use Greedy Algorithms?

→ When you need the optimal answer but can build it step by step by always choosing what looks best at the moment, without revisiting past choices.

→ Triggers: “Minimize/maximize cost/number,” “choose earliest/latest,” “intervals/activities,” “cover all with fewest.”
→ Works for interval scheduling, coins, jumps, and when local optimum leads to global optimum.

14// When do you need Topological Sort?

→ When you’re dealing with ordering or scheduling tasks with dependencies, especially in directed acyclic graphs (DAGs), or when something needs to be done before something else.

→ Triggers: “Prerequisite,” “dependency,” “order of tasks,” “build sequence,” “can you finish all?”
→ O(V+E) time, identifies cycles or valid orderings in tasks.

15// When does Prefix Sum (and Suffix Sum) shine?

→ When you need to efficiently compute sums or aggregates over subarrays or ranges—especially for repeated range queries.

→ Triggers: “Sum in a range,” “update/query,” “intervals,” “multiple subarrays,” “difference between partitions.”

→ Turns O(n^2) repeated computation into O(n) preprocessing + O(1) queries.

16// When to use Union Find (Disjoint Set)?

→ When the problem is about merging groups, tracking connectivity, or finding cycles/components, especially in dynamic connectivity questions.

→ Triggers: “Are X and Y connected?”, “how many groups?”, “merge/find leader/root,” “detect cycle in undirected graph.”

→ Best for O(α(n)) union/find, which is near constant with path compression.