Common DSA Problem-Solving Patterns

In Data Structures and Algorithms (DSA), many problems can be solved efficiently by recognizing recurring patterns. Here’s a cheat sheet of the most common ones you’ll encounter.

Two Pointers

Use two variables (pointers) to traverse the data structure in different ways (same direction, opposite ends, or fast/slow).
■ Common problems:
- Check if a string/array is a palindrome
- Two-sum in a sorted array
- Detect cycle in a linked list
- Find middle of a linked list
- Sliding window variations

Hash Map / Store-and-Search

Use a hash map to remember elements you’ve seen and quickly check if a matching element exists.
■ Common problems:
- Two-sum (unsorted array)
- Subarray Sum Equals K
- Longest substring without repeating characters
- Anagram detection

Sliding Window

Use two pointers to represent a window that expands and contracts based on conditions. Great for subarray/substring problems.
■ Common problems:
- Maximum sum subarray of size k
- Longest substring without repeating characters
- Minimum window substring

Binary Search Pattern

Divide the search space in half repeatedly. Works on sorted arrays or monotonic search spaces.
■ Common problems:

- Binary search in sorted array
- Search in rotated sorted array
- Find minimum in rotated sorted array
- Peak element in array

Greedy

Make the best local choice at each step, hoping it leads to the global optimum.
■ Common problems:
- Activity selection / Interval scheduling
- Coin change (minimum coins, if denominations allow greedy)
- Jump game

Dynamic Programming (DP)

Break a problem into subproblems, solve them once, and store results for reuse. Often involves memoization or tabulation.
■ Common problems:
- Fibonacci sequence
- House robber
- Longest common subsequence
- Knapsack problem

Backtracking

Try building a solution incrementally and backtrack when it fails. Great for combinatorial search problems.
■ Common problems:
- N-Queens
- Sudoku solver
- Permutations and combinations
- Word search

Graph Traversal

Explore nodes/edges in a graph using BFS, DFS, or Dijkstra’s algorithm depending on problem requirements.
■ Common problems:
- Number of islands
- Shortest path (Dijkstra, BFS)
- Topological sort
- Graph cycle detection