The Ultimate DSA Patterns Deep Dive

Mastering the 20 Patterns that Solve 95% of Interview Questions

---

1. Sliding Window

Concept:
Instead of re-calculating a subarray from scratch, you "slide" a window over the array. You add the new element entering the window and remove the element leaving it. This turns nested loops (O(N²)) into a single loop (O(N)).

Core Logic:

- **Expand** `windowEnd` to include elements.
- **Shrink** `windowStart` when the window violates a condition (e.g., sum > target).
- Maintain a running `currentSum` or `count`.

Top Practice Questions:

1. **Maximum Sum Subarray of Size K** (Easy)
2. **Smallest Subarray with a given sum** (Medium)
3. **Longest Substring with K Distinct Characters** (Medium)
4. **Fruits into Baskets** (Medium)
5. **Longest Substring Without Repeating Characters** (Hard)

---

2. Two Pointers

Concept:
Use two pointers to iterate through a sorted array or list. One pointer starts at the beginning, the other at the end (or both at the start moving at different speeds). This avoids nested loops for finding pairs.

Core Logic:

- **Sorted Array:** If `sum > target`, decrement `end`. If `sum < target`, increment `start`.
- **In-place operations:** One pointer reads, one pointer writes (for removing duplicates).

Top Practice Questions:

1. **Pair with Target Sum** (Easy)
2. **Remove Duplicates from Sorted Array** (Easy)
3. **Squaring a Sorted Array** (Easy)
4. **3Sum** (Medium)
5. **Subarrays with Product Less than a Target** (Medium)
6. **Dutch National Flag Problem** (Medium)

---

3. Fast & Slow Pointers (Tortoise & Hare)

Concept:
Use two pointers moving at different speeds (usually 1x and 2x). Useful for detecting cycles in linear data structures or finding the middle element.

Core Logic:

- `slow = slow.next`
- `fast = fast.next.next`
- If `slow == fast`, there is a cycle.

Top Practice Questions:

1. **LinkedList Cycle** (Easy)
2. **Start of LinkedList Cycle** (Medium)
3. **Happy Number** (Medium)
4. **Middle of the LinkedList** (Easy)
5. **Palindrome LinkedList** (Medium)

---

4. Merge Intervals

Concept:
Deal with overlapping intervals. The key is to **sort** the intervals by their start time first.

Core Logic:

- Sort intervals by `start`.
- Iterate: If `current.start < previous.end`, they overlap. Merge them by taking `max(current.end, previous.end)`.

Top Practice Questions:

1. **Merge Intervals** (Medium)
2. **Insert Interval** (Medium)
3. **Intervals Intersection** (Medium)
4. **Conflicting Appointments** (Medium)
5. **Meeting Rooms II** (Hard)

---

5. Cyclic Sort

Concept:
If you are given an array of numbers ranging from `1` to `N` (or `0` to `N`), you can sort it in O(N) time without extra space.

Core Logic:

- Iterate through the array.
- If `nums[i]` is not at the correct index (i.e., `nums[i] != i + 1`), swap it with the number at its correct index.

Top Practice Questions:

1. **Cyclic Sort** (Easy)
2. **Find the Missing Number** (Easy)
3. **Find all Disappeared Numbers** (Easy)
4. **Find the Duplicate Number** (Medium)
5. **First Missing Positive** (Hard)

---

6. In-place Reversal of a LinkedList

Concept:
Reverse a linked list without using extra memory. You need to carefully manipulate `prev`, `curr`, and `next` pointers.

Core Logic:

- Save `next` node.
- Point `curr.next` to `prev`.
- Move `prev` to `curr`.
- Move `curr` to `next`.

Top Practice Questions:

1. **Reverse a LinkedList** (Easy)
2. **Reverse a Sub-list** (Medium)
3. **Reverse Every K-element Sub-list** (Medium)
4. **Rotate List** (Medium)

---

7. Tree Breadth First Search (BFS)

Concept:
Traverse a tree level-by-level. Use a **Queue** to store nodes of the current level.

Core Logic:

- Push root to Queue.
- While Queue is not empty:
  - Get size of Queue (elements in current level).
  - Process all nodes in this level.
  - Add their children to the Queue.

Top Practice Questions:

1. **Binary Tree Level Order Traversal** (Easy)
2. **Reverse Level Order Traversal** (Easy)
3. **Zigzag Traversal** (Medium)
4. **Level Averages in a Binary Tree** (Easy)
5. **Connect Level Order Siblings** (Medium)

---

8. Tree Depth First Search (DFS)

Concept:
Traverse a tree by going as deep as possible before backtracking. Use **Recursion** (Stack).

Core Logic:

- Base case: If node is null, return.
- Recursive step: Process node, then call DFS on left child, then right child.

Top Practice Questions:

1. **Path Sum** (Easy)
2. **Path Sum II (All paths)** (Medium)
3. **Sum of Path Numbers** (Medium)
4. **Path With Given Sequence** (Medium)
5. **Count Paths for a Sum** (Medium)

---

9. Two Heaps

Concept:
Divide a set of numbers into two parts: a "Small" half and a "Large" half. Use a **Max-Heap** for the small half and a **Min-Heap** for the large half. This gives you instant access to the median.

Core Logic:

- Max-Heap stores the smaller half of numbers.
- Min-Heap stores the larger half.
- Balance the heaps so their size difference is at most 1.

Top Practice Questions:

1. **Find Median from Data Stream** (Hard)
2. **Sliding Window Median** (Hard)
3. **Maximize Capital** (Hard)

---

10. Subsets (Backtracking)

Concept:
Build a solution step-by-step. If a step leads to an invalid solution, **backtrack** (undo the step) and try a different path.

Core Logic:

- Use recursion.
- At each step, decide whether to include an element or not.
- Base case: Reached end of input.

Top Practice Questions:

1. **Subsets** (Medium)
2. **Subsets II (Duplicates)** (Medium)
3. **Permutations** (Medium)
4. **String Permutations by changing case** (Medium)
5. **Balanced Parentheses** (Hard)

---

11. Modified Binary Search

Concept:
Use Binary Search on data that isn't perfectly sorted (e.g., rotated) or infinite.

Core Logic:

- Find `mid`.
- Determine which half of the array is sorted.
- Decide if the target lies in the sorted half or the other half.

Top Practice Questions:

1. **Order-agnostic Binary Search** (Easy)
2. **Ceiling of a Number** (Medium)
3. **Next Letter** (Medium)
4. **Search in a Sorted Infinite Array** (Medium)
5. **Search in Rotated Sorted Array** (Medium)

---

12. Bitwise XOR

Concept:
XOR rules: `a ^ a = 0` and `a ^ 0 = a`. This is perfect for finding unique elements when everything else appears twice.

Core Logic:

- XOR all elements in the array.
- Duplicates cancel out (`a ^ a = 0`).
- The result is the single unique number.

Top Practice Questions:

1. **Single Number** (Easy)
2. **Two Single Numbers** (Medium)
3. **Complement of Base 10 Number** (Medium)
4. **Flip Image** (Easy)

---

13. Top 'K' Elements

Concept:
Find the K largest, smallest, or most frequent elements. Use a **Heap** (Priority Queue).

Core Logic:

- To find Top K Largest: Use a **Min-Heap** of size K.
- Add elements. If heap size > K, remove the smallest (root).
- The heap will contain the K largest elements.

Top Practice Questions:

1. **Top K Frequent Elements** (Medium)
2. **Kth Largest Element in an Array** (Medium)
3. **K Closest Points to Origin** (Medium)
4. **Connect Ropes** (Easy)
5. **Frequency Sort** (Medium)

---

14. K-way Merge

Concept:
Merge K sorted lists into one sorted list.

Core Logic:

- Push the first element of each list into a **Min-Heap**.
- Pop the smallest (root), add to result.
- Push the next element from the same list that the popped element came from.

Top Practice Questions:

1. **Merge K Sorted Lists** (Hard)
2. **Kth Smallest Number in M Sorted Lists** (Medium)
3. **Smallest Number Range** (Hard)

---

15. 0/1 Knapsack (Dynamic Programming)

Concept:
You have items with weights and values. You have a knapsack with capacity C. Maximize value.

Core Logic:

- `dp[i][c]` = Max value using first `i` items with capacity `c`.
- Decision: Either exclude item `i` OR include item `i` (if weight allows).
- `dp[i][c] = max(dp[i-1][c], value[i] + dp[i-1][c-weight[i]])`.

Top Practice Questions:

1. **0/1 Knapsack** (Medium)
2. **Equal Subset Sum Partition** (Medium)
3. **Subset Sum** (Medium)
4. **Minimum Subset Sum Difference** (Hard)

---

16. Topological Sort (Graph)

Concept:
Find a linear ordering of nodes in a Directed Acyclic Graph (DAG) such that for every edge U -> V, U comes before V.

Core Logic:

- Calculate **in-degrees** (number of incoming edges) for all nodes.
- Add nodes with `in-degree = 0` to a Queue (Sources).
- Process queue: Remove node, add to result, decrement in-degree of neighbors. If neighbor's in-degree becomes 0, add to Queue.

Top Practice Questions:

1. **Course Schedule** (Medium)
2. **Course Schedule II** (Medium)
3. **Alien Dictionary** (Hard)

---

17. Union Find (Disjoint Set)

Concept:
Efficiently track which set an element belongs to and merge sets.

Core Logic:

- `find(x)`: Find the root/parent of x.
- `union(x, y)`: Make the root of x point to the root of y.
- Use **Path Compression** and **Union by Rank** for O(1) amortized time.

Top Practice Questions:

1. **Number of Provinces** (Medium)
2. **Redundant Connection** (Medium)
3. **Number of Operations to Make Network Connected** (Medium)

---

18. Monotonic Stack

Concept:
Keep elements in the stack sorted (increasing or decreasing). Useful for finding "Next Greater Element".

Core Logic:

- While `stack.top() < current_element`:
    - `stack.pop()` (The current element is the "Next Greater" for the popped element).
- `stack.push(current_element)`.

Top Practice Questions:

1. **Next Greater Element I** (Easy)
2. **Daily Temperatures** (Medium)
3. **Largest Rectangle in Histogram** (Hard)

---

19. Trie (Prefix Tree)

Concept:
Tree-like structure for storing strings. Each node represents a character.

Core Logic:

- Root is empty.
- Each node has 26 children (for 'a'-'z').
- Useful for prefix matching.

Top Practice Questions:

1. **Implement Trie (Prefix Tree)** (Medium)
2. **Design Add and Search Words Data Structure** (Medium)
3. **Word Search II** (Hard)

---

20. Matrix Traversal (Islands)

Concept:
Traverse a grid (2D array) as a graph. Each cell is a node, neighbors are edges.

Core Logic:

- Use DFS or BFS.
- Mark visited cells to avoid infinite loops.
- Check boundaries (row < 0, col < 0, row >= rows, col >= cols).

Top Practice Questions:

1. **Number of Islands** (Medium)
2. **Flood Fill** (Easy)
3. **Max Area of Island** (Medium)
4. **Rotting Oranges** (Medium)