🚀 DSA LeetCode Cheatsheet - Complete Interview Reference

⏱️ BIG-O COMPLEXITY

Operation	Time	Space
Array Access	O(1)	O(1)
Hash Lookup	O(1)	O(n)
Binary Search	O(log n)	O(1)
Linear Search	O(n)	O(1)
Quick Sort	O(n log n)	O(log n)
Merge Sort	O(n log n)	O(n)
DFS/BFS	O(V+E)	O(V)

🟢 Excellent 🔵 Good 🟠 Fair 🔴 Poor

🎯 DATA STRUCTURES BY PRIORITY

MUST KNOW

- **Arrays:** Random access O(1), insertion/deletion O(n)
- **Strings:** Immutable, substring operations
- **Hash Maps:** Key-value pairs, O(1) average lookup
- **Linked Lists:** Dynamic size, O(1) insertion at head
- **Stacks:** LIFO - push, pop, peek operations
- **Queues:** FIFO - enqueue, dequeue operations
- **Two Pointers:** Fast/slow, left/right techniques

IMPORTANT

- **Binary Trees:** BST properties, traversals
- **Heaps:** Priority queue, min/max heap operations
- **Sets:** Unique elements, union/intersection
- **Prefix Arrays:** Cumulative sums, range queries
- **Matrix:** 2D array operations, rotation
- **Deque:** Double-ended queue operations

ADVANCED

- **Tries:** Prefix trees for string operations
- **Segment Trees:** Range queries and updates
- **Union-Find:** Disjoint set operations
- **Fenwick Trees:** Binary indexed trees
- **Red-Black Trees:** Self-balancing BST
- **B-Trees:** Multi-way search trees

⚡ CORE ALGORITHMS

Sorting & Searching

- **Binary Search:** O(log n), requires sorted array
- **Merge Sort:** O(n log n), stable, divide & conquer
- **Quick Sort:** O(n log n) avg, in-place partitioning
- **Heap Sort:** O(n log n), in-place, not stable
- **Counting Sort:** O(n+k), for integer keys
- **Radix Sort:** O(nk), non-comparison based

Tree/Graph Traversal

- **DFS:** Stack/recursion, explores depth first
- **BFS:** Queue-based, level-by-level exploration
- **Inorder:** Left-Root-Right (gives sorted BST)
- **Preorder:** Root-Left-Right (tree copying)
- **Postorder:** Left-Right-Root (tree deletion)
- **Topological:** DAG ordering, dependency resolution

🧠 PROBLEM-SOLVING PATTERNS

Sliding Window	*Subarray/substring problems*
Two Pointers	*Sorted arrays, palindromes*
Fast & Slow	*Cycle detection, middle element*
Merge Intervals	*Overlapping ranges*
Cyclic Sort	*Missing/duplicate numbers*
Tree BFS	*Level-order traversal*
Tree DFS	*Path sum, diameter problems*
Backtracking	*Generate all combinations*
Binary Search	*Search space reduction*
Top K Elements	*Heap-based solutions*
Modified Binary Search	*Rotated/modified arrays*
Subsets	*Bit manipulation, recursion*

🔧 KEY TECHNIQUES

Dynamic Programming

- **Memoization:** Top-down, recursive with cache
- **Tabulation:** Bottom-up, iterative approach
- **1D DP:** Fibonacci, house robber, climb stairs
- **2D DP:** Grid paths, edit distance, LCS
- **Knapsack:** 0/1 and unbounded variants

Bit Manipulation

- **XOR:** Find unique number, swap without temp
- **AND (&):** Check if bit is set
- **OR (|):** Set a specific bit
- **Left Shift (<<):** Multiply by 2^n
- **Right Shift (>>):** Divide by 2^n
- **Complement (~):** Flip all bits

Graph Algorithms

- **Dijkstra:** Single-source shortest path
- **Bellman-Ford:** Handles negative weights
- **Floyd-Warshall:** All-pairs shortest path
- **Kruskal/Prim:** Minimum spanning tree
- **Union-Find:** Cycle detection in graphs
- **A* Search:** Heuristic pathfinding

📚 8-WEEK STUDY PATH

Week 1
Foundation
Big-O, Arrays, Strings, Hash Maps

Week 2
Linear Structures
Linked Lists, Stacks, Queues

Week 3
Trees
Binary Trees, DFS, BFS, Recursion

Week 4
Sorting
Merge, Quick, Heap Sort, Binary Search

Week 5
Dynamic Programming
Memoization, Tabulation, Classic Problems

Week 6
Graphs
DFS, BFS, Shortest Path, MST

Week 7
Advanced Topics
Heaps, Tries, Union-Find, Backtracking

Week 8
Practice
Company Problems, Mock Interviews

💡 INTERVIEW SUCCESS TIPS

Before Coding

- Clarify requirements and constraints
- Ask about input size and edge cases
- Discuss your approach before coding
- Estimate time and space complexity

While Coding

- Think out loud and explain your logic
- Start with brute force, then optimize
- Use meaningful variable names
- Test with examples as you code

Common Pitfalls

- Not handling null/empty inputs
- Off-by-one errors in loops
- Integer overflow in calculations
- Wrong loop termination conditions

Time Management (45min)

- 5min: Problem understanding
- 5min: Approach discussion
- 25min: Implementation
- 10min: Testing & debugging