---

🟢 **Beginner Level (Core Foundations)**

🧠 **Theory:**

- What is DSA and why it's needed
- Time complexity (Big O notation)
- Space complexity
- Best, worst, and average case
- How to analyze algorithms
- Constant, linear, quadratic, and logarithmic complexities
- Asymptotic notations (O, Ω, Θ)

💻 **Practical:**

- Measuring time complexity of simple loops
- Convert nested loops into efficient forms
- Write functions with O(1), O(n), and $O(n^2)$ complexity examples
- Use recursion to calculate factorial, sum of n numbers

---

🟡 **Intermediate Level (Linear Data Structures)**

🧠 **Theory:**

- Arrays and their memory representation
- Strings and immutability
- Linked Lists (Singly, Doubly, Circular)
- Stacks and Queues
- Hashing (Hash tables and maps)
- Sliding window concept
- Two-pointer technique

💻 **Practical (JavaScript based):**

- Reverse an array in-place
- Move all zeros to end (O(n))
- Rotate array by k positions
- Find missing number from 1..n

- Implement Stack using array
- Implement Queue using two stacks
- Check for balanced parentheses using stack
- Find first non-repeating character in string
- Implement Linked List (insert, delete, print)
- Detect loop in linked list (Floyd's algorithm)
- Find middle of linked list
- Merge two sorted linked lists
- Remove duplicates from sorted array
- Sliding window problems (max sum subarray, anagrams, longest substring)

---

🔴 **Advanced Level (Non-linear Structures & Sorting/Searching)**

🧠 **Theory:**

- Trees and Binary Trees
- Binary Search Tree (BST)
- Binary Heaps (Min & Max Heap)
- Graphs (Directed, Undirected, Weighted, Unweighted)
- Tries (Prefix Tree)
- HashMap & Set internals
- Sorting algorithms
- Searching algorithms
- Divide and Conquer principle

💻 **Practical (JavaScript):**

🔹 **Sorting:**

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Heap Sort
- Counting Sort

- Radix Sort
- Time & space complexity comparison of sorts

🔹 **Searching:**

- Linear Search
- Binary Search (recursive & iterative)
- Search in rotated sorted array
- Find first/last occurrence using binary search
- Find peak element

🔹 **Trees:**

- Create binary tree from array
- Inorder / Preorder / Postorder traversal
- Level order traversal (BFS)
- Height / Depth of tree
- Count leaf nodes
- Check if binary tree is balanced
- Lowest Common Ancestor (LCA)
- BST insert, delete, search
- Validate BST

🔹 **Heaps:**

- Build heap (heapify)
- Extract min/max
- Heap Sort using array
- Kth largest/smallest element using heap

🔹 **Graphs:**

- Represent graph using adjacency list
- BFS & DFS traversal
- Detect cycle in directed/undirected graph
- Topological sort
- Dijkstra's shortest path algorithm
- Prim's and Kruskal's MST

🔹 **Tries:**

- Insert words into trie
- Search word/prefix
- Autocomplete feature using trie

---

🔵 **Expert Level (Dynamic Programming, Backtracking & Advanced Patterns)**

🧠 **Theory:**

- Greedy algorithms
- Backtracking
- Recursion tree analysis
- Dynamic programming (Top-down, Bottom-up)
- Memoization and Tabulation
- Bit Manipulation
- Graph algorithms (Advanced)
- Advanced trees (AVL, Segment, Fenwick Tree)

💻 **Practical (JavaScript):**

🔹 **Dynamic Programming:**

- Fibonacci (recursive + memoized + bottom-up)
- Climbing stairs problem
- 0/1 Knapsack
- Subset sum
- Longest Common Subsequence (LCS)
- Longest Increasing Subsequence (LIS)
- Minimum Path Sum
- Coin Change (min coins)
- Edit Distance
- Matrix Chain Multiplication

🔹 **Backtracking:**

- N-Queens problem
- Sudoku solver
- Rat in a maze
- Generate all subsets (power set)

- Generate permutations of array
- Combination sum

🔹 **Greedy:**

- Activity selection problem
- Fractional knapsack
- Huffman coding concept
- Minimum number of coins

🔹 **Bit Manipulation:**

- Count set bits
- Find missing number using XOR
- Power of two check
- Single number in array (XOR method)
- Subsets using bit masking

🔹 **Advanced Graphs:**

- Bellman-Ford algorithm
- Floyd–Warshall algorithm
- Kosaraju's SCC algorithm
- Tarjan's algorithm
- Disjoint Set Union (Union-Find)
- Detect bridges and articulation points

🔹 **Advanced Trees:**

- Segment tree (range sum query)
- Fenwick tree (binary indexed tree)
- Trie-based prefix search optimization
- LRU Cache implementation

---

⚫ **Expert+ (System Level & Real Interview Prep)**

- Analyze space vs time tradeoffs
- Optimization techniques for recursive problems
- Cache-based DP optimization

- DSA patterns (Sliding Window, Two Pointers, Binary Search on Answer, Backtracking, Bitmask DP)
- Advanced heap usage (median of data stream, top K elements)
- Graph traversal on grid problems
- Union-Find applications (connected components, network problems)
- Tree + DP hybrid problems (path sum, diameter, etc.)
- Optimize complex algorithms using memoization + pruning
- Competitive programming patterns
- Real-world DSA implementation in JS (Maps, Sets, WeakMaps, WeakSets, Queues)
- Practice LeetCode patterns by category

---