13-Week Data Structures and Algorithms (DSA) Lesson Plan

Duration: 6 hours per week (4 hours lecture + 2 hours practice)
Objective: Build a strong foundation in DSA concepts and prepare students for technical interviews and competitive programming.

#	Lecture Plan (4 hrs)	Practice Plan (2 hrs)	Practice Problems
1	**Introduction to DSA (1 Hour)** ■ Importance and real-life applications of DSA in software development ■ Overview of complexity analysis: ○ Time complexity and its significance ○ Space complexity and optimization strategies **Arrays (1.5 Hours)** ■ Basics of arrays: Definition, advantages, and limitations. ■ Array operations: ○ Insertion and deletion ○ Searching (Linear Search, Binary Search) ■ Common problems: ○ Finding duplicates in an array ○ Subarray problems (e.g., maximum sum subarray) **Strings (1.5 Hours)** ■ Introduction to strings and their representation in memory ■ Common operations: Concatenation, substring search, and reversal ■ Example problems: ○ Palindrome check ○ Detecting anagrams	**Array Problems** ■ Reverse an array ■ Find the maximum and minimum element in an array ■ Search for an element using Linear and Binary Search **String Problems** ■ Check if a string is a palindrome ■ Detect if two strings are anagrams ■ Count occurrences of a substring within a string	**LeetCode** • Reverse Array: LeetCode - Reverse Array • Find the Maximum and Minimum Element in Array: LeetCode - Find Maximum **Codeforces** • Watermelon: Codeforces - Watermelon • Array Partition: Codeforces - Array Partition
2	**Recursion Basics (1.5 Hours)** • Understanding recursion: Definition and structure of a recursive function • Applications of recursion: Factorial, Fibonacci sequence, Tower of Hanoi • Recursion vs. Iteration: When to use which approach **Sorting Techniques (2.5 Hours)** • Introduction to sorting: Importance in data organization	**Recursive Problems** ■ Solve problems like finding factorial, Fibonacci sequence, and sum of digits **Sorting Problems** ■ Implement Bubble Sort and Selection Sort	**LeetCode** • Factorial of a Number: LeetCode - Factorial • Generate Parentheses: LeetCode - Generate Parentheses **Codeforces**

	• Bubble Sort and Selection Sort: ○ Step-by-step implementation ○ Time and space complexity analysis • Real-world use cases of sorting	▪ Solve problems that require sorting (e.g., sorting student marks)	• Recursive Practice: Codeforces - Recursive Practice • Palindrome Partitioning: Codeforces - Palindrome Partitioning
3	**Advanced Sorting Techniques (2 Hours)** • Merge Sort: ○ Divide-and-conquer approach ○ Step-by-step implementation ○ Time and space complexity analysis • Quick Sort: ○ Pivot selection techniques ○ Partitioning logic ○ Best, average, and worst-case analysis **Introduction to Hashing (2 Hours)** • Basics of hashing: Definition and use cases • Hash functions and collision handling techniques (Chaining, Open Addressing) • Applications of hashing: Dictionary, frequency count, etc.	**Sorting Problems** • Implement Merge Sort and Quick Sort • Solve problems requiring sorted data for optimization **Hashing Problems** • Implement a hash table using chaining • Solve frequency count problems (e.g., count occurrences of elements in an array)	**LeetCode** • Merge Sort Implementation: LeetCode - Merge Sort • Quick Sort Implementation: LeetCode - Quick Sort • Two Sum (Hashing): LeetCode - Two Sum **Codeforces** • Sorting for Efficient Search: Codeforces - Sorting • Hashing Challenge: Codeforces - Hashing
4	**Basics of Linked Lists (2 Hours)** • Introduction to linked lists: ○ Difference between arrays and linked lists • Types of linked lists: Singly, Doubly, and Circular • Operations: ○ Insertion, Deletion, Traversal **Applications of Linked Lists (2 Hours)** • Implementing stacks and queues using linked lists • Solving problems like reversing a linked list and detecting loops	**Linked List Problems** • Implement a singly linked list with insertion and deletion • Reverse a linked list • Detect and remove a loop in a linked list	**LeetCode** • Reverse a Linked List: LeetCode - Reverse Linked List • Detect Cycle in Linked List: LeetCode - Linked List Cycle **Codeforces** • Linked List Problem: Codeforces - Linked List • Remove Duplicates in Linked List: Codeforces - Remove Duplicates
5	**Stacks (2 Hours)** • Introduction to stacks: LIFO principle • Operations: Push, Pop, Peek • Applications: Expression evaluation, balancing parentheses	**Stack Problems** • Implement a stack using arrays and linked lists	**LeetCode** • Valid Parentheses: LeetCode - Valid Parentheses

	Queues (2 Hours) • Introduction to queues: FIFO principle • Types of queues: Normal, Circular, Priority • Applications: Scheduling, buffer management	• Solve problems like balancing parentheses and evaluating postfix expressions **Queue Problems** • Implement a circular queue. • Solve problems like simulating a job queue	• Implement Stack using Queues: LeetCode - Implement Stack **Codeforces** • Stack Data Structure: Codeforces - Stack • Queue Implementation: Codeforces - Queue
6	**Basics of Trees (2 Hours)** • Definition and terminology (node, edge, height, depth, etc.) • Binary trees: Types and properties • Applications of trees in search and storage **Tree Traversals (2 Hours)** • Depth-First Search (DFS): Inorder, Preorder, Postorder • Breadth-First Search (BFS): Level-order traversal	**Tree Problems** • Implement DFS (Inorder, Preorder, Postorder) and BFS traversals • Solve problems like finding the height of a binary tree	**LeetCode** • Binary Tree Inorder Traversal: LeetCode - Inorder Traversal • Binary Tree Level Order Traversal: LeetCode - Level Order Traversal **Codeforces** • Binary Tree Depth: Codeforces - Binary Tree Depth • Tree Traversals Challenge: Codeforces - Tree Traversals
7	**Introduction to Binary Search Trees (2 Hours)** • Definition and properties of BST • Insertion and deletion in a BST • Searching in a BST **Applications of BST (2 Hours)** • Use cases like maintaining sorted data and efficient searching • Solving problems like finding the lowest common ancestor (LCA) and range queries	**BST Problems** • Implement insertion, deletion, and search operations in a BST • Solve problems like finding the minimum and maximum elements in a BST	**LeetCode** • Validate Binary Search Tree: LeetCode - Validate BST • Lowest Common Ancestor of BST: LeetCode - LCA in BST **Codeforces** • Binary Search Tree Implementation: Codeforces - BST • BST Search and Delete: Codeforces - Search and Delete
8	**Basics of Heaps (2 Hours)** • Min-heaps and max-heaps: Properties and structure • Insertion and deletion in heaps • Heapify process and building a heap	**Heap Problems** • Implement a min-heap and max-heap	**LeetCode** • Merge k Sorted Lists: LeetCode - Merge k Sorted Lists

	Priority Queues (2 Hours) • Implementing priority queues using heaps • Applications: Scheduling tasks, finding the k largest/smallest elements	• Solve problems like merging k sorted arrays using heaps	• Kth Largest Element in an Array: LeetCode - Kth Largest **Codeforces** • Priority Queue Problem: Codeforces - Priority Queue • Heaps Implementation: Codeforces - Heap Problem
9	**Basics of Graphs (2 Hours)** • Graph terminology (vertices, edges, adjacency list/matrix). • Types of graphs: Directed, undirected, weighted, unweighted. **2. Graph Traversals (2 Hours)** • Depth-First Search (DFS). • Breadth-First Search (BFS). • Applications: Finding connected components, detecting cycles	**Graph Problems** • Implement DFS and BFS. • Solve problems like detecting a cycle in a directed/undirected graph	**LeetCode** • Clone Graph: LeetCode - Clone Graph • Number of Connected Components in an Undirected Graph: LeetCode - Connected Components **Codeforces** • BFS Traversal: Codeforces - BFS • Graph Connectivity: Codeforces - Graph Connectivity
10	**Shortest Path Basics (2 Hours)** • Dijkstra’s Algorithm: Single-source shortest path. • Bellman-Ford Algorithm: Handling negative weights. **Applications of Shortest Path Algorithms (2 Hours)** • Network routing. • Solving problems like finding the shortest path in a weighted graph.	**Shortest Path Problems** • Implement Dijkstra’s and Bellman-Ford algorithms. • Solve real-world scenarios like finding the shortest route in a city map	**LeetCode** • Dijkstra's Algorithm: LeetCode - Dijkstra's Algorithm • Bellman-Ford Algorithm: LeetCode - Bellman-Ford **Codeforces** • Dijkstra’s Algorithm in Graphs: Codeforces - Dijkstra • Shortest Path Challenge: Codeforces - Shortest Path
11	**Basics of DP (2 Hours)** • Definition and need for DP. • Principles of overlapping subproblems and optimal substructure. • Memoization vs. Tabulation. **Classic DP Problems (2 Hours)**	**DP Problems** • Solve problems like LCS and 0/1 Knapsack using DP. • Practice converting recursive solutions to DP-based solutions	**LeetCode** • Fibonacci Number: LeetCode - Fibonacci • 0/1 Knapsack Problem: LeetCode - 0/1 Knapsack

12	• Fibonacci sequence. • 0/1 Knapsack Problem. • Longest Common Subsequence (LCS)		• Longest Common Subsequence: LeetCode - LCS **Codeforces** • Dynamic Programming Introduction: Codeforces - DP Introduction • Knapsack DP Challenge: Codeforces - Knapsack
12	**Minimum Spanning Trees (MST) (2 Hours)** • Kruskal’s Algorithm. • Prim’s Algorithm. • Applications of MST in network design. **2. Advanced Topics (2 Hours)** • Floyd-Warshall Algorithm for all-pairs shortest paths. • Topological sorting and its applications	**Graph Problems** • Implement MST using Kruskal’s and Prim’s algorithms. • Solve problems like finding the critical edges in a network	**LeetCode** • Minimum Spanning Tree (Prim's Algorithm): LeetCode - MST • Floyd-Warshall Algorithm: LeetCode - All-Pairs Shortest Path **Codeforces** • Minimum Spanning Tree: Codeforces - MST Problem • Topological Sorting: Codeforces - Topological Sort

13 DP++