Week 1: Basics of Programming and Arrays	
What is DSA? Why is it important?	
Basic time and space complexity analysis.	
Find the maximum element in an array.	
Find the minimum element in an array.	
Find the sum of all elements in an array.	
Find the average of elements in an array.	
Find the kth largest element in an array.	
Find the kth smallest element in an array.	
Find the majority element (more than n/2 occurrences).	
Check if an array contains duplicate elements.	
Rotate an array to the right by k steps.	
Rotate an array to the left by k steps.	
Merge two sorted arrays.	
Move all zeroes to the end of the array while maintaining the order of non-zero elements.	
Find the intersection of two arrays.	
Find the union of two arrays.	
Find the missing number in an array containing n-1 elements out of 1 to n.	
Find the duplicate number in an array containing n+1 elements from 1 to n.	
Rearrange an array such that all negative numbers come before positive numbers.	
Count the frequency of each element in an array.	
Week 2: Strings	
Reverse a string.	
Check if a string is a palindrome.	
Count the number of vowels in a string.	
Check if two strings are anagrams.	
Find the longest substring without repeating characters.	
Find the longest common prefix of two strings.	
Find all permutations of a string.	
Convert a string to an integer (atoi implementation).	
Perform basic string compression (e.g., "aabcccccaaa" -> "a2b1c5a3").	
Implement a basic string search (naive approach).	
Implement string search using the Knuth-Morris-Pratt (KMP) algorithm.	
Find the longest palindromic substring.	
Remove all adjacent duplicate characters from a string.	
Replace all spaces in a string with '%20'.	
Count and replace all occurrences of a given character in a string.	
Implement a function to check if two strings are rotations of each other.	
Find the length of the longest substring with at most two distinct characters.	
Check if a string is a valid number (integer/float).	
Find the shortest substring containing all characters of another string.	
Implement the strstr function.	
Week 3: Linked Lists	
Implement a singly linked list.	

Find the length of a linked list.	
Insert a node at the beginning of the linked list.	
Insert a node at the end of the linked list.	
Insert a node after a given node in the linked list.	
Delete a node from the linked list.	
Delete the first occurrence of a value from the linked list.	
Reverse a linked list.	
Find the middle of a linked list.	
Detect a cycle in a linked list.	
Remove duplicates from a linked list.	
Find the nth node from the end of a linked list.	
Merge two sorted linked lists.	
Find the intersection point of two linked lists.	
Split a linked list into two halves.	
Check if a linked list is a palindrome.	
Rotate a linked list by k nodes.	
Implement a doubly linked list.	
Reverse a doubly linked list.	
Remove all nodes with a given value from a linked list.	
Week 4: Stacks and Queues	
Implement a stack using arrays.	
Implement a stack using a linked list.	
Implement a stack with a maximum value.	
Evaluate a postfix expression.	
Check for balanced parentheses in an expression.	
Sort a stack using another stack.	
Implement a queue using arrays.	
Implement a queue using a linked list.	
Implement a circular queue.	
Implement a queue with two stacks.	
Design and implement a priority queue using a heap.	
Find the first non-repeating character in a stream of characters.	
Implement a double-ended queue (deque) using arrays.	
Implement a deque using a doubly linked list.	
Implement a stack with push, pop, min, and top operations in O(1) time complexity.	
Week 5: Trees	
Implement a binary tree.	
Find the height of a binary tree.	
Traverse a binary tree in-order.	
Traverse a binary tree pre-order.	
Traverse a binary tree post-order.	
Level order traversal of a binary tree.	
Check if a binary tree is balanced.	
Find the lowest common ancestor of two nodes in a binary tree.	

Convert a binary tree to a doubly linked list.	
Check if a binary tree is a binary search tree (BST).	
Find the diameter of a binary tree.	
Find the distance between two nodes in a binary tree.	
Find the maximum width of a binary tree.	
Find all nodes at distance k from a given node in a binary tree.	
Find the maximum path sum in a binary tree.	
Serialize and deserialize a binary tree.	
Flatten a binary tree to a linked list.	
Find the level of a given node in a binary tree.	
Find all leaf nodes in a binary tree.	
Print all root-to-leaf paths in a binary tree.	
Week 6: Binary Search Trees (BST)	
Implement a BST.	
Insert a node in a BST.	
Delete a node from a BST.	
Find the minimum value in a BST.	
Find the maximum value in a BST.	
Find the kth smallest element in a BST.	
Find the kth largest element in a BST.	
Convert a BST to a sorted doubly linked list.	
Check if two BSTs are identical.	
Find the range of values in a BST.	
Find the successor of a given node in a BST.	
Find the predecessor of a given node in a BST.	
Validate if a given sequence is a BST postorder traversal.	
Print all nodes within a given range in a BST.	
Find the lowest common ancestor of two nodes in a BST.	
Check if a BST is height-balanced.	
Count the number of nodes in a BST.	
Convert a BST to a greater tree (i.e., every node’s value is replaced by the sum of all greater values).	
Find all nodes at a distance k from a given node in a BST.	
Check if a BST is a valid preorder traversal.	
Week 7: Heaps and Hashing	
Implement a min-heap.	
Implement a max-heap.	
Heap sort algorithm.	
Find the k largest elements in an array using a heap.	
Find the k smallest elements in an array using a heap.	
Merge k sorted arrays using heaps.	
Implement a priority queue using a heap.	
Find the median of a list of numbers (using heaps).	
Implement a hash table with chaining for collision handling.	
Implement a hash table with open addressing for collision handling.	

Find the first non-repeating character in a string using a hash table.	
Check for pairs with a given sum in an array using hashing.	
Find common elements in three sorted arrays using hashing.	
Group anagrams from a list of strings using hashing.	
Find all subsets of a set using hashing.	
Check if a subarray with zero sum exists using hashing.	
Count the frequency of elements in an array using hashing.	
Find all pairs of integers in an array that sum up to a specific target using hashing.	
Find the longest substring with exactly k distinct characters.	
Implement a LRU cache using hashing and doubly linked list.	
Week 8: Graphs	
Implement a graph using adjacency matrix.	
Implement a graph using adjacency list.	
Perform Depth-First Search (DFS) on a graph.	
Perform Breadth-First Search (BFS) on a graph.	
Check if a graph is connected.	
Find the shortest path in an unweighted graph using BFS.	
Find the shortest path in a weighted graph using Dijkstra's algorithm.	
Find shortest paths in a weighted graph using Bellman-Ford algorithm.	
Detect a cycle in an undirected graph.	
Detect a cycle in a directed graph.	
Find all paths from source to destination in a directed graph.	
Find the Minimum Spanning Tree (MST) using Kruskal's algorithm.	
Find the Minimum Spanning Tree (MST) using Prim's algorithm.	
Find strongly connected components in a directed graph using Kosaraju's algorithm.	
Find strongly connected components in a directed graph using Tarjan's algorithm.	
Find all bridges in a graph.	
Find all articulation points in a graph.	
Topological sort of a directed graph.	
Detect negative weight cycles using Bellman-Ford algorithm.	
Implement a graph traversal that outputs the shortest path from source to all other nodes.	
Week 9: Dynamic Programming	
Solve the 0/1 Knapsack problem.	
Find the longest common subsequence between two sequences.	
Find the longest increasing subsequence in an array.	
Solve the coin change problem (minimum number of coins).	
Find the edit distance between two strings.	
Solve the matrix chain multiplication problem.	
Find the number of ways to climb stairs (with 1 or 2 steps at a time).	
Find the minimum path sum in a grid (dynamic programming approach).	
Solve the partition problem (divide a set into two subsets with equal sum).	
Solve the 0/1 Knapsack problem using a dynamic programming table.	
Find the maximum length of a subarray with a sum equal to a given value.	
Solve the rod cutting problem (maximize revenue by cutting a rod into pieces).	

Find the minimum number of operations required to convert one string into another (Levenshtein distance).	
Find the maximum profit from stock prices with at most one transaction.	
Find the maximum profit from stock prices with unlimited transactions.	
Solve the problem of finding the largest square of 1s in a binary matrix.	
Solve the problem of finding the longest palindromic subsequence in a string.	
Solve the problem of counting the number of unique paths in a grid.	
Find the maximum sum of a submatrix.	
Solve the problem of finding the minimum number of coins needed for a given amount.	
Find the length of the longest path in a matrix where you can move in all 8 directions.	
Solve the problem of longest common substring between two strings.	
Find the maximum sum of non-adjacent elements in an array.	
Solve the problem of finding the number of ways to partition a set into subsets with equal sum.	
Week 10: Backtracking	
Solve the N-Queens problem.	
Find all subsets of a set.	
Solve the Sudoku problem (fill in the grid so that each row, column, and subgrid contains all digits from 1 to 9).	
Find all permutations of a given string.	
Solve the subset sum problem (find a subset with a given sum).	
Solve the Rat in a Maze problem (find all possible paths from start to end in a maze).	
Solve the problem of generating all possible combinations of k elements from a set.	
Find all possible combinations that sum up to a target value.	
Solve the problem of finding all possible ways to place N queens on an N x N chessboard.	
Solve the problem of generating all valid IP addresses from a given string.	
Find all possible unique paths in a matrix from top-left to bottom-right.	
Solve the problem of placing a fixed number of knights on a chessboard such that no two knights can attack each other.	
Solve the problem of finding all subsets of a given set with duplicate elements.	
Solve the problem of finding all possible ways to construct a binary tree from a given preorder traversal.	
Solve the problem of finding all valid parentheses combinations for a given number of pairs.	
Solve the problem of finding all possible ways to color a graph such that no two adjacent nodes have the same color.	
Find all possible arrangements of a given list of integers with duplicates.	
Solve the problem of finding all possible ways to reach a target sum using a combination of numbers (combination sum).	
Find all unique combinations that add up to a specific target from a list of candidates.	
Solve the problem of finding all possible paths from the root to leaves in a binary tree.	
Final Review and Practice	
Review and re-solve all previously solved problems to reinforce concepts.	
Attempt mock interviews with timed problems to simulate real interview conditions.	
Focus on areas where you had difficulties and practice related problems.	
Explore additional problems on platforms like LeetCode, HackerRank, Codeforces, or GeeksforGeeks.	
Implement and practice problems related to advanced data structures such as Trie and Segment Trees if time permits.	