DSA Practice Questions

1. Arrays

- **Find the missing number in an array**: Given an array containing `n-1` distinct numbers taken from `1` to `n`, find the missing number.
- **Subarray with given sum**: Find a subarray that adds up to a given sum in an unsorted array.
- **Maximum subarray sum (Kadane's Algorithm)**: Find the contiguous subarray with the maximum sum in a given one-dimensional array.
- **Rearrange array in alternating positive and negative items**: Without using extra space, rearrange the array to have alternating positive and negative elements.

2. Linked Lists

- **Reverse a linked list**: Write a function to reverse a linked list.
- **Detect a loop in a linked list**: Use Floyd's Cycle-Finding Algorithm to detect if a loop exists.
- **Merge two sorted linked lists**: Merge two sorted linked lists into a single sorted linked list.
- **Remove nth node from end**: Remove the nth node from the end of a linked list in one pass.

3. Stacks and Queues

- **Implement a stack using queues**: Implement a stack using two queues.
- **Balanced parentheses**: Given a string containing just the characters `(`, `)`, `{`, `}`, `[` and `]`, determine if the input string is valid.
- **Next greater element**: For each element in an array, find the next greater element on its right side in the array.
- **LRU Cache implementation**: Design and implement an LRU (Least Recently Used) cache.

4. Trees and Graphs

- **Binary Tree Inorder Traversal**: Implement inorder traversal of a binary tree (both recursive and iterative).
- **Lowest Common Ancestor in a Binary Tree**: Given a binary tree, find the lowest common ancestor of two nodes.
- **Graph Depth-First Search (DFS) and Breadth-First Search (BFS)**: Implement DFS and BFS for a graph.

- **Dijkstra's Shortest Path Algorithm**: Implement Dijkstra's algorithm to find the shortest path in a graph with non-negative weights.

5. Dynamic Programming

- **Longest Increasing Subsequence**: Find the length of the longest subsequence that is strictly increasing.
- **0/1 Knapsack Problem**: Solve the knapsack problem using dynamic programming.
- **Coin Change Problem**: Given a set of coins, find the minimum number of coins required to make a certain amount.
- **Edit Distance**: Compute the minimum number of operations required to convert one string into another.

6. Sorting and Searching

- **Binary Search**: Implement binary search in a sorted array.
- **Merge Sort**: Write a function to perform merge sort on an array.
- **Quick Sort**: Implement quick sort using a pivot.
- **Find the peak element**: Given an array, find an element that is greater than or equal to its neighbors.

7. Strings

- **Longest Palindromic Substring**: Find the longest palindromic substring in a given string.
- **Anagram Check**: Check if two strings are anagrams of each other.
- **String to Integer (atoi)**: Convert a string to an integer.
- **Substring with Concatenation of All Words**: Find all starting indices of substring(s) in a given string that is a concatenation of each word in the given list exactly once.

8. Greedy Algorithms

- **Activity Selection Problem**: Given `n` activities with their start and finish times, select the maximum number of activities that can be performed.
- **Huffman Coding**: Implement Huffman coding for data compression.
- **Fractional Knapsack Problem**: Given weights and values of `n` items, determine the maximum value of the knapsack with a weight capacity `W`.
- **Minimum Platforms**: Find the minimum number of platforms required at a railway station given arrival and departure times.

9. Backtracking

- **N-Queens Problem**: Solve the N-Queens problem using backtracking.
- **Sudoku Solver**: Implement a Sudoku solver.

- **Subset Sum Problem**: Determine if there is a subset of a given set with a sum equal to a given value.
- **Permutations of a String**: Generate all permutations of a given string.

10. Advanced Topics

- **Trie Data Structure**: Implement a Trie with insert, search, and delete operations.
- **Segment Tree**: Build and query a segment tree for range queries.
- **Kruskal's Algorithm**: Implement Kruskal's algorithm for finding the Minimum Spanning Tree (MST).
- **KMP String Matching Algorithm**: Implement the Knuth-Morris-Pratt (KMP) algorithm for string pattern matching.

Tips for Practicing

1. **Start simple**: Begin with easier problems to build confidence and understanding.
2. **Understand the problem**: Read the problem statement carefully and understand the constraints and expected input/output.
3. **Pseudocode**: Write pseudocode before implementing the solution to clarify your approach.
4. **Practice regularly**: Consistency is key. Regular practice will help solidify concepts.
5. **Analyze your solutions**: Review your code for efficiency and correctness, and try to improve it.