DAA Assignments - Algorithms Questions

Unit 1: Introduction to Algorithms and Data Structures

Lab Session 1 : Fundamentals and Basic Sorting

Question 1: Write a program to determine whether the given number is Prime or not. Analyze its time complexity using Big O notation.
Question 2: Implement Bubble Sort for a given array of integers. Demonstrate the step-by-step sorting process.
Question 3: Implement Selection Sort for a given array of integers. Analyze its time complexity compared to Bubble Sort.
Question 4: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. Consider both iterative and recursive approaches.

Lab Session 2 : Advanced Sorting and Algorithmic Analysis

Question 1: Implement Insertion Sort for a given array of integers. Discuss scenarios where Insertion Sort might be preferable over Bubble or Selection Sort.
Question 2: Explain and demonstrate Big O, Omega, and Theta notations with concrete examples for different functions (e.g., f(n)=n2, f(n)=nlogn, f(n)=n).
Question 3: Given an array, implement a function to find the kth smallest element. Analyze its complexity.
Question 4: There are N children standing in a line with some rating value. You want to distribute a minimum number of candies to these children such that: Each child must have at least one candy. The children with higher ratings will have more candies than their neighbours. You need to write a program to calculate the minimum candies you must give. (Hint: Consider multiple passes or dynamic programming-like thought processes).

Unit 2: Divide and Conquer

Lab Session 1: Divide and Conquer Fundamentals

Question 1: Implement Binary Search to efficiently find an element in a sorted array. Compare its performance with a linear search for large datasets.
Question 2: Solve the Max-Min problem for a given array of numbers using the divide-and-conquer approach.
Question 3: Implement Quick Sort for a given array of integers. Trace the execution for a small example.
Question 4: There is a new barn with N stalls and C cows. The stalls are located on a straight

line at positions x[0],x[1]...x[N−1] (0≤xi≤1,000,000,000). We want to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance? (Hint: Use binary search on the answer, which involves a "check" function that places cows greedily).

Lab Session 2: Advanced Divide and Conquer

Question 1: Implement Merge Sort for a given array of integers. Analyze its time complexity and space complexity.
Question 2: Implement Strassen's algorithm for 2×2 matrix multiplication. Compare its number of multiplications with the standard matrix multiplication.
Question 3: Given an array of integers, find the maximum subarray sum using the divide and conquer approach.
Question 4: Given a set of points in a 2D plane, find the closest pair of points using the divide-and-conquer approach.

Unit 3: Greedy Algorithms

Lab Session 1 : Minimum Spanning Trees

Question 1: Implement Kruskal's algorithm to find the Minimum Spanning Tree (MST) of a given graph represented by an adjacency list/matrix.
Question 2: Implement Prim's algorithm to find the Minimum Spanning Tree (MST) of a given graph, starting from an arbitrary vertex.
Question 3: Compare Kruskal's and Prim's algorithms in terms of their time complexity and suitability for different graph representations.
Question 4: Given an undirected graph with N vertices and M is the number of columns in the grid consisting of '0's (Water) and '1's (Island). Find the number of islands. (Hint: This can be a greedy exploration problem using DFS/BFS to count connected components).

Lab Session 2 : Shortest Paths and Greedy Applications

Question 1: Implement Dijkstra's algorithm to find the shortest paths from a source vertex to all other vertices in a given weighted, directed graph with non-negative edge weights.
Question 2: Solve the Fractional Knapsack problem using a greedy approach. Demonstrate with an example how items are chosen.
Question 3: Implement Huffman Coding to compress a given text string. Calculate the compression ratio.
Question 4: Given a set of activities with start and finish times, select the maximum number of non-overlapping activities. Implement a greedy solution.

Unit 4: Dynamic Programming

Lab Session 1 : Core Dynamic Programming

Question 1: Solve the 0/1 Knapsack problem for given items (weights and values) and a knapsack capacity using dynamic programming.
Question 2: Solve the Matrix Chain Multiplication problem using dynamic programming to find the optimal order of matrix multiplications. Print the minimum number of scalar multiplications.
Question 3: Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits from num. (Hint: This can be approached with a greedy strategy or dynamic programming ideas depending on how you frame subproblems).
Question 4: Find the Longest Common Subsequence (LCS) of two given strings using dynamic programming. Show the DP table construction.

Lab Session 2 (90 minutes): All-Pairs Shortest Paths and Applications

Question 1: Implement Warshall's algorithm to find all-pair shortest paths in a given graph represented by an adjacency matrix.
Question 2: Implement Floyd's algorithm to find all-pair shortest paths in a given graph, allowing for negative edge weights (but no negative cycles).
Question 3: Implement a dynamic programming solution to the Rod Cutting problem, finding the maximum obtainable value by cutting a rod of length n into smaller pieces.
Question 4: There is a robot on an m×n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m-1][n-1]). The robot can only move either down or right at any point in time. Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

Unit 5: Graph Algorithms (Traversal and Shortest Paths)

Lab Session 1 : Graph Traversal

Question 1: Implement Depth First Search (DFS) for a given graph (adjacency list representation). Print the traversal order and identify visited nodes.
Question 2: Implement Breadth First Search (BFS) for a given graph. Print the traversal order and the shortest path (in terms of number of edges) from a source node to all reachable nodes.
Question 3: Given a graph with V vertices and E edges, check whether it contains any cycle or not. (Hint: Use DFS and keep track of visited and recursion stack).
Question 4: Given an undirected graph with N vertices and E edges. Represent N servers numbered from 0 to N-1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. A

critical connection is a connection that, if removed, will make some servers unable to reach some other servers. Return all critical connections in the network in any order. (Hint: Identify bridges in the graph).

Lab Session 2 : Advanced Graph Algorithms

Question 1: Implement Topological Sort for a given Directed Acyclic Graph (DAG) using either DFS-based or Kahn's algorithm (BFS-based).
Question 2: Implement Bellman-Ford algorithm to find the shortest paths from a source vertex to all other vertices in a given graph, handling negative edge weights. Detect if a negative cycle exists.
Question 3: Given a grid of size N×M where each cell in the grid can have values 0, 1, or 2. Implement a solution to determine the minimum time it takes for all fresh oranges (1) to rot. (Hint: Use BFS, treating '2' (rotten oranges) as source nodes and expand layer by layer).
Question 4: Given two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert str1 into str2. Operations: Insert, Remove, Replace. All of the above operations are of equal cost. (Hint: This is the Edit Distance problem, typically solved with dynamic programming, but can be framed here as a complex graph problem or for applying algorithmic thinking from this unit).

Unit 6: Backtracking and Branch & Bound

Lab Session 1: Backtracking Techniques

Question 1: Solve the N-Queens problem for a given N using backtracking. Display one valid configuration for N=4 and N=8.
Question 2: Implement a backtracking solution for the Sudoku Solver problem.
Question 3: Find all possible subsets of a given set of distinct integers using backtracking.
Question 4: Given an array of integers, find all unique combinations that sum up to a target value. Each number in the array may only be used once in the combination. Implement this using backtracking.

Lab Session 2: Optimization with Backtracking and Branch & Bound

Question 1: Solve the 0/1 Knapsack problem using a backtracking approach. Analyze the search space compared to the dynamic programming approach.
Question 2: Implement a backtracking solution for the Traveling Salesman Problem (TSP) for a small number of cities (e.g., 4-5 cities) to find the shortest possible route that visits each city exactly once and returns to the origin city.
Question 3: Briefly explain the core idea of the Branch and Bound method. Illustrate with a simple example (e.g., Job Assignment Problem).
Question 4: Minimum Path Sum says that given a grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along the path. Implement this problem. (Hint: This is typically solved with Dynamic Programming, but can be framed to illustrate how a Branch and Bound approach could explore paths).

Unit 7: String Matching and Complexity Classes

Lab Session 1: String Matching Algorithms

Question 1: Implement the Naive String Matching algorithm to find all occurrences of a pattern in a given text.
Question 2: Implement the Rabin-Karp algorithm for string matching using hashing. Handle potential collisions.
Question 3: Implement the Knuth-Morris-Pratt (KMP) algorithm for string matching. Construct the Longest Proper Prefix which is also a Suffix (LPS) array for a given pattern and use it to search in text.
Question 4: We have to determine what is the minimum time required to rot all oranges. A rotten orange at index [i,j] can rot other fresh oranges at indexes [i–1,j], [i+1,j], [i,j–1], [i,j+1] (up, down, left and right) in unit time. (This is a repeat from Unit 5 Lab 2, but it fits conceptually here as well under algorithms and could be revisited with a focus on efficient search patterns).

Lab Session 2 : Finite Automata and Complexity Theory

Question 1: Design and implement a string matching algorithm using the concept of finite automata. For a given pattern, construct its corresponding finite automaton and simulate its behavior on a given text.
Question 2: Explain the classes P and NP with examples of problems belonging to each class. Discuss the significance of the P vs NP problem in theoretical computer science.
Question 3: Discuss the Hamiltonian Cycle problem. Explain why it is an NP-complete problem and its implications for finding efficient solutions.
Question 4: What is the Travelling Salesman Problem (TSP)? Discuss why it is a well-known NP-hard problem. Briefly outline approaches (like approximation algorithms or heuristics) used to solve large instances of TSP.