Algorithms
Assignment

Guidelines:

- This assignment is a group of 3 from the same Lab.
- Please submit your solutions via Google Classroom
- You should submit a zip file (name this file with your ids A_ID1_ID2_ID3.zip), put all files (codes) for each problem (name code file p1 , p2 , .. etc)
- In all problems, take into account, measuring performance, in terms of time, and draw a graph that represents the analysis.
- Feel free to use any programming language, C++ is preferred.
- Feel free to use any AI tool, take into your consideration, grading will be based on your understanding, all your team, if the team couldn't explain will take no grade in the problem

Problems:

Problem 1:

Finding the Maximum Subarray Sum

You need to implement the Divide and Conquer approach to find the maximum subarray sum of a given array. Compare the performance of your Divide and Conquer solution with a brute force approach by measuring the execution time for both methods on arrays of varying sizes.

Given an array of integers, find the contiguous subarray with the largest sum.

Example:
Input: [-2, 1, -3, 4, -1, 2, 1, -5, 4]
Output: 6 (Subarray: [4, -1, 2, 1])

Problem 2:

Hash Map Implementation and Collision Handling

Implement a hashmap data structure with chaining as the collision resolution technique. Test the efficiency of your hashmap by inserting and retrieving a large number of key-value pairs. Compare the performance of your hashmap with a simple array-based approach in terms of insertion and retrieval times for various load factors

- **Bonus(Optional)** → Draw A GUI that represents the chaining hashing.

Example:
Insert (Key, Value): (101, "Alice"), (201, "Bob"), (112, "Carol")
Retrieve (Key): 201
Output: "Bob"

Problem 3:

Ordered Sequence Maintenance

Implement a skip list data structure to maintain an ordered sequence of elements. Use the skip list to efficiently support insertion, deletion, and search operations. Compare the time complexity of these operations with those of a balanced binary search tree (e.g., AVL tree) for various input sizes.

- **Bonus(Optional)** → Draw GUI that represents the skiplist

Example:
Insert: 5, 10, 20, 15
Search: 20
Output: Found

Problem 4:

Coin Change Problem

You are given an array of coin denominations and a target amount. Your task is to find the minimum number of coins needed to make up the target amount. Implement a dynamic programming solution to solve this problem efficiently. Compare the performance of your dynamic programming solution with a naive recursive approach by measuring the execution time for both methods on various target amounts and coin denominations.

Example 1:

Coin Denominations: [1, 5, 10, 25]
Target Amount: 30

To make up 30 cents, one optimal solution is to use three 10-cent coins, resulting in a total of 30 cents. The minimum number of coins needed is 3.

Example 2:

Coin Denominations: [1, 5, 10, 25]
Target Amount: 16

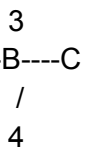
To make up 16 cents, an optimal solution is to use one 10-cent coin and one 5-cent coin, resulting in a total of 15 cents. The minimum number of coins needed is 2.

Problem 5:

Implement an algorithm to find the Minimum Spanning Tree of a given connected graph. You can use either Kruskal's algorithm or Prim's algorithm. Measure and compare the runtime of your chosen algorithm with a brute force approach that generates all possible spanning trees and selects the one with the minimum weight.

- **Bonus(Optional)** → Draw the graph that represents the map, and show the solutions in steps in the GUI

Example:
Graph:



Output (MST edges): [(A, B), (B, C)]

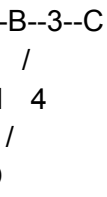
Problem 6:

Shortest Path Calculation

Implement Dijkstra's algorithm to find the shortest paths from a given source vertex to all other vertices in a weighted graph. Compare the efficiency of Dijkstra's algorithm with a naive approach that explores all possible paths from the source vertex to each destination vertex. Measure the time taken by both methods for various graph sizes and densities.

- **Bonus(Optional)** → Draw the graph that represents the map, and show the solutions in steps in the GUI

Example:
Graph:



Source: A
Shortest Paths: A to B: 2, A to C: 5, A to D: 1