## **Contents**

* *Title*
* *Introduction*
* *Implementation details*
* *Algorithm descriptions*
* *Performance Comparison*
* *Challenges and Solutions*
* *Conclusion*
* *References*

**Comparing Divide and Conquer and Greedy Algorithms in Java**

**Introduction**

This project aims at using object - oriented programming (OOP) concepts in the implementation of a java application on divide and conquer and greedy algorithms. These algorithms will be compared and tested. The program will have a friendly user interface for a good user experience.

**Implementation Details**

* Our application will be implemented using object- oriented programming concepts like classes, objects and methods.
* The various algorithms will be represented by their class names.
* Subclasses of algorithms will be created using inheritance.
* Objects will be instantiated from these classes.
* The objects will be encapsulated to maintain the integrity of the various algorithms.
* Every class and objects will be described by their methods.
* Similar methods will be called on different objects using polymorphism.

**Algorithm Descriptions**

This section provides a brief description of the various algorithms under divide- and- conquer and greedy algorithms (Quick sort, Merge sort, Closest-Problem, Strassen’s Matrix Multiplication, Quick Hull, Prim’s Minimum Spanning Tree (MST), Travelling Salesman Problem (TSP), Kruskal’s MST, Djikstra’s Shortest Path and Huffman codes).

***Divide and Conquer Algorithms***

Divide and conquer algorithms solves problems by dividing them into independent subproblems, which are solved in a specific order to get the final solution. They work according to this general plan;

* Divide a problem into several sub problems of the same type
* Solve them recursively
* Combine the solutions to the subproblems to get an overall solution to the original problem.

Some of these algorithms will be described subsequently as we progress.

* **Merge Sort**

This is a divide- and-conquer algorithm for sorting unsortedarrays or lists. It works by recursively by dividing the array into two halves, sorting each halves and then merging the sorted arrays to produce a fully sorted array. It has a time complexity of O(nlogn) for all cases.

* **Quick Sort**

This also a divide and conquer algorithm for sorting arrays. But, unlike the merge sort; this sorting algorithm works by selecting a pivot element from the array. It then divides the other elements into two sub arrays in decreasing or increasing order with reference to the pivot. The sub arrays are then sorted recursively to generate the solution to the problem. It has an average worst case and best case of 0(nlogn) and O(n2 ) respectively. A good choice of pivot will enhance the efficiency of the algorithm.

* **Closest Pair Problem**

This is a geometric algorithm that finds two points in a set that are closest to each other. Here, a set of points are divided into two halves. They are sorted by their x-coordinates and stored in an array say, M. The minimum distance in the divided sets of points are recursively found. They are then combined; where one point is from the first half and the other point from the second half. An array containing points in the dividing line(d) is created. The distance for each point m in the newly created array is checked. Then the closest pair of points found in any of the divided points or across the dividing line becomes the closest pair found.

* **Strassen’s Matrix Multiplication**

This is an efficient algorithm for multiplying matrices. Given any n\*n matrix A and B; strassen’s matrix multiplication splits each matrix into four submatrices. Seven products of the submatrices are computed using some formulas. The results of the four matrices are computed using another formula. They are then combined to form the final n\*n matrix.

* **Quick Hull**

This is also a computational geometric algorithm. It is used for finding the convex hull of a set of points in a plane. This algorithm is also similar to the Quick sort algorithm which was described earlier. This algorithm works by identifying the points with the minimum and maximum x-coordinates in the convex hull. A line segment is used to divide the set of points into two subsets; points above and below the line segment. The points that form the convex hull on both sides of the line segments are found recursively. The are then combined as the final solution.

***Greedy Algorithms***

* **Prim’s Minimum Spanning**

This is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. It works by choosing an arbitrary vertex to start from and initializes a priority queue. A minimum edge is removed from the priority queue. The vertex inclusion is checked and all edges emanating from the included verted is added to the priority queue. This process continues till all vertices are included in the minimum spanning tree.

* **Travelling Salesman Problem**

This is an algorithm for solving combinatorial problems. It is used for finding the shortest route to a city. It works by visiting the nearest unvisited city until all cities are visited.

* **Kruskal’s MST**

This is used for finding the minimum spanning tree. It works by sorting the edges in a graph in ascending order. Then it adds the smallest edge to the MST. Once all vertices are connected, the MST is returned.

* **Djikstra’s Shortest Path**

This algorithm is used to find the shortest path from a particular node to other nodes in a graph.

* **Huffman Codes**

This is a greedy algorithm used for lossless data compression**.**