FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
DEPARTMENT OF COMMUNICATION TECHNOLOGY AND NETWORK

PROJECT CNS 4202 SEMESTER II 2024/2025

COURSE NAME : DESIGN AND ANALYSIS OF ALGORITHM

COURSE CODE : CNS 4202

GROUP: 11

PROJECT TITLE : OPTIMIZING PARCEL DELIVERY ROUTES FOR EFFICIENT COURIER OPERATIONS

LECTURER: DR. NUR ARZILAWATI BINTI MD YUNUS

NAME	MATRIC ID
MUHAMMAD SYAFI BIN ABDUL NASIR	214321
ARIFF AMEER AIZAD BIN NORDIN	214037
MUHAMMAD FAZRUL HAFIZI BIN ANUAR	213271
MUHAMMAD ARIF ZAHIRUDDIN BIN YASIR	214264

Table of Content

1 Introduction

In the current era of e-commerce technology and logistics, parcel delivery has become a key service in meeting customer expectations. One of the biggest issues faced by courier companies is how to plan delivery routes that save time, reduce fuel usage, and improve delivery efficiency. Poorly planned routes lead to longer travel times, higher costs, and delayed deliveries.

This project focuses on solving that problem by applying algorithmic techniques to improve delivery route planning. The goal is to determine the **shortest route** for a courier to visit all required destinations **once** and return to the starting point—a well-known issue called the **Travelling Salesman Problem (TSP)**.

To tackle this, we explore two different algorithms: the graph **approach**, which is quick and simple but may not always give the best result, and **dynamic programming**, which takes longer to run but guarantees the most efficient route. Both methods will be analyzed in terms of their performance, accuracy, and practicality in real-life delivery situations.

2 Objective

The objective of this project is to design and implement algorithmic solutions to optimize parcel delivery routes by minimizing travel distance and improving efficiency. By modeling the problem as a variation of the Travelling Salesman Problem (TSP), the project applies and compares two algorithmic approaches—Graph and Dynamic Programming—to evaluate their effectiveness in generating efficient delivery paths. The project also aims to analyze the performance of both algorithms in terms of accuracy and complexity, develop a working Java program to demonstrate the solution, and present the findings through an online portfolio.

3 Overview of Problem Scenario

In today's rapidly evolving logistics industry, courier companies face increasing challenges in managing efficient parcel delivery operations. Many delivery routes are still planned manually, which often leads to longer delivery times, increased fuel consumption, and limited ability to respond to dynamic conditions such as new delivery orders or traffic disruptions. This lack of route optimization results in delays, customer dissatisfaction, and inefficient resource usage.

The problem becomes more critical as demand for faster and more reliable delivery services grows. Without an intelligent system in place, couriers may not take the shortest or fastest route, and route planning cannot easily adapt to changes throughout the day. The current system also struggles to handle delivery updates in real time, making it unsuitable for modern logistics demands.

To address these issues, this project proposes an automated and optimized route planning system that leverages algorithmic solutions—specifically, **dynamic programming** and **graph algorithms—**to determine the most efficient path that allows a courier to visit all required delivery locations exactly once and return to the starting hub. By minimizing total travel distance and time, this system aims to improve operational efficiency, reduce delivery costs, and enhance customer satisfaction.

3.1 Story Description

A courier company operating in the busy urban landscape of Kuala Lumpur faces increasing pressure to deliver parcels quickly and efficiently. Each day, delivery riders are assigned multiple destinations, often spread across different parts of the city. However, the current method of planning routes is largely manual, relying on individual driver experience or static route maps. This leads to inconsistencies, longer delivery times, and inefficient fuel usage. The situation becomes even more complex when real-time events such as traffic congestion, last-minute order changes, or delivery cancellations are introduced. Without an intelligent and adaptive system, the company struggles to maintain punctuality and optimize delivery performance. The challenge lies in designing a system that can generate the most efficient delivery route — one that visits all required locations exactly once, starts and ends at the courier hub, and adapts quickly to changes throughout the day.

3.2 Importance of Optimal Solution

Finding an optimal solution is crucial in the courier delivery context, as it directly impacts operational cost, delivery speed, and customer satisfaction. An optimal route not only reduces the total distance traveled but also cuts fuel expenses and lowers the environmental impact of daily delivery operations. In urban areas like Kuala Lumpur, even small improvements in routing efficiency can translate into significant time savings and better service quality. By applying advanced algorithmic techniques such as dynamic programming and graph algorithms, the system can automatically compute the most efficient paths, removing the reliance on manual planning. This ensures that couriers spend less time on the road, reduce the likelihood of late deliveries, and meet customer expectations with greater consistency. Furthermore, having an automated and optimized routing system enables the company to scale operations more effectively as order volumes grow.

3.3 Expected Result

The expected output of this project is a working system that automatically generates optimized delivery routes based on input delivery points. The system should produce a delivery path that starts and ends at the courier hub, visits each destination only once, and minimizes the overall travel distance and time. It should support real-time adaptation to changes, such as updated delivery points or cancellations, by re-optimizing the route efficiently. Using Graph algorithms, the system will compute the shortest paths between individual points, while the Dynamic Programming approach will be used to evaluate all possible delivery sequences and select the most optimal one. The final output will include the delivery order, total distance, and a visual or textual representation of the route, along with a performance comparison between the two algorithms used.

4 Overview of Algorithm Design Techniques

In solving optimization problems like parcel delivery route planning, selecting a suitable algorithm design technique is essential to balance efficiency and solution quality. Two common techniques for this type of problem are the Greedy algorithm and Dynamic Programming (DP).

The Greedy algorithm builds a solution step by step by selecting the locally optimal choice at each stage with the hope of finding a global optimum. In the context of the Travelling Salesman Problem (TSP), the Greedy (or Nearest Neighbor) approach is fast and easy to implement, but it does not always guarantee the shortest route. According to Choudhary and Kumar (2021), greedy strategies are efficient for large-scale route planning but may result in suboptimal solutions when the problem complexity increases.

On the other hand, Dynamic Programming is a more comprehensive approach that solves problems by breaking them down into overlapping subproblems and storing solutions to avoid redundant computation. For TSP, the Held-Karp algorithm a DP-based method—finds the optimal route but has exponential time complexity, making it more suitable for smaller datasets. As noted by Yadav et al. (2019), DP provides optimal solutions for combinatorial optimization but is limited in scalability due to memory and time constraints.

Comparatively, the Greedy algorithm is well-suited for real-time decision-making in logistics, whereas Dynamic Programming is ideal for exact planning when the number of delivery points is relatively small. Pisinger and Røpke (2007) emphasize that hybrid methods combining Greedy heuristics with dynamic optimization can achieve a balance between performance and precision in logistics route planning.

4.1 Greedy Algorithm

The Greedy algorithm solves problems by choosing the locally optimal option at each step in the hope that these choices lead to a globally optimal solution. In the context of parcel delivery, the Greedy method can be implemented using the **Nearest Neighbor approach**, which always visits the closest unvisited delivery location. This method is simple and efficient, especially for large datasets where quick approximations are acceptable. However, it does not guarantee the optimal route. According to Choudhary and Kumar (2021), Greedy algorithms perform well for real-time systems due to their low computational cost but often result in suboptimal solutions in complex routing problems like TSP.

4.2 Dynamic Programming

Dynamic Programming is ideal for solving optimization problems that involve overlapping subproblems and optimal substructure, such as the **Held-Karp algorithm** for TSP. In this project, DP provides an exact solution by evaluating all possible routes efficiently using memoization, which makes it suitable for smaller instances of the delivery problem. Yadav et al. (2019) emphasize that DP is effective in producing optimal solutions but becomes impractical for larger datasets due to its exponential time and space complexity. It's best used when accuracy is more critical than speed.

4.3 Divide and Conquer

Divide and Conquer works by breaking a problem into smaller subproblems, solving them independently, and combining the results. While this strategy is effective for many algorithmic problems such as merge sort and quicksort, it is **not naturally suited for route optimization** like TSP. According to Skiena (2008), D&C is powerful for computational problems but lacks the ability to maintain global constraints in problems like delivery path planning, where all subpaths must contribute to a single coherent route.

4.4 Graph Algorithm

Graph algorithms, such as **Dijkstra's algorithm**, are commonly used for finding the shortest path between two points. In the parcel delivery context, they can be useful to compute the shortest distance between individual pairs of delivery points, which can then be used as input for higher-level TSP solvers. However, standard graph algorithms alone do not solve the entire delivery routing problem where multiple nodes must be visited exactly once. According to Sedgewick and Wayne (2011), graph algorithms form a crucial foundation for shortest-path computations but require integration with TSP-specific logic for full-route optimization.

4.5 Algorithm Comparison

Algorithm	Strength	Weaknesses	Time Complexity
Greedy	Fast and simple to implement, Low memory usage, quick approximations	May lead to suboptimal solutions, makes local choices without future consideration	O(n log n)
Dynamic Programming	Guarantees optimal solution, efficient with small input (≤15 cities)	High time and space complexity, not scalable for large n, and Slower than greedy	$O(n^2)$ to $O(n^3)$
Graph Algorithm	Excellent for shortest path between two nodes, useful for building distance matrix	Cannot solve multi-node route problems alone and not suited for TSP directly	varies
Divide and Conquer	Great for sorting and structured data processing, efficient for divisible problems	Not suitable for problems requiring global path awareness and cannot solve TSP directly	O(n log n)
Sorting Algorithm	Faster search operations	Extra computational overhead	O(n log n)

Figure 1: Algorithm Comparison Table

5 Algorithm Design

5.1 Algorithm Paradigm Explanation

This project applies two algorithmic paradigms, which are Dynamic Programming and Greedy Algorithm (Nearest Neighbor) to optimize parcel delivery routes. The algorithms aim to find the shortest possible route that visits all delivery points exactly one, adapting intelligently to varying delivery maps.

5.1.1 Data Type Used

To implement both algorithms efficiently, three primary data types were used, which are Boolean to track whether each node has been visited or not. Next data type is Array, which is used to form the 5x5 Matrix Routes that are used to test both of the algorithms. This matrix graph will represent the distance matrix between all nodes (cities). Lastly, data type List is used to store the sequence of nodes in a resulting path or route.

5.1.2 Objective Function

The objective function of both algorithms is to minimize delivery time, by taking the optimal routes that has the least distance travelled and idle time. To achieve this, the next objective is to automate route planning. Instead of manual planning of delivery route, this project can use algorithms that can determine the optimal route to be taken by courier. Due to many locations and ever-changing landscape, the last objective of this project is to have it adapt to changing location. This is done by designing an algorithm that can accept different matrix routes and determine the best path to be taken in each matrix routes, ensuring flexibility of algorithm, no matter where it is applied.

5.1.3 Constraints

To ensure route optimization, this project sets several constraints that are to be fulfilled during the optimization process. First is to have the courier to travel to ALL nodes, where each nodes must be visited exactly once. Secondly, the courier must return to the starting point to complete the cycle, as per the Travelling Salesman Problem constraint. Lastly, total length of path must be of the shortest possible distance to ensure best option.

5.1.4 Recurrence and Optimization Function

Recurrence occurs in Dynamic Programming, where this algorithm uses bitmasking and memoization to store and reuse solutions of subproblems. In this project, that means going through the matrix routes and seeing all possible routes available, and that the shortest path will be selected and recreated. In the algorithm, the recurrence relation is:

$dp[mask][i]=min(dp[mask \oplus (1 \ll i)][j]+cost[j][i])$
Where:

Mask indicates the set of visited nodes,
i is the current node
j iterates through all possible previous nodes

This function minimizes the path cost incrementally by evaluating all subsets, that is, the possible paths. This results in a time complexity of $O(n^2 \cdot 2^n)$

5.2 Pseudocode

5.2.1 Pseudocode Using Greedy Algorithm

```
function NearestNeighbor(matrix, start):
    current ← start
    repeat until all nodes are visited:
        find nearest unvisited node to current
        mark it visited
        add to path
        current ← nearest
    return path
```

5.2.2 Pseudocode using Dynamic Programming

```
function TSP_DP(dist):
    for each subset mask:
        for each current node u in mask:
            for each unvisited node v:
                update dp[nextMask][v] with min cost
                update parent for path reconstruction

    find endNode with minimum cost from fullMask

    path = reconstruct path from parent table
    return minCost
```

6 Algorithm Analysis

6.1 Algorithm Correctness

To verify algorithm correctness, we must assess whether each algorithm consistently generates valid delivery routes that comply with constraints like rider capacity and delivery time windows.

6.1.1 Greedy Algorithm Correctness

The correctness of the Greedy algorithm in the context of parcel delivery route optimization is based on its ability to consistently produce **valid and complete routes**. At each step, the algorithm selects the **nearest unvisited delivery location**, building a path that eventually includes all destinations and returns to the starting point. While it does **not guarantee the optimal solution**, it satisfies **partial correctness** by ensuring that all delivery points are visited exactly once without duplication. The algorithm also satisfies **termination**, as it completes once all nodes are visited. Therefore, the Greedy algorithm is considered correct in terms of generating feasible routes, even if the total travel distance may not be the shortest possible.

6.1.2 Dynamic Programming Correctness

The correctness of the Dynamic Programming (DP) algorithm in parcel delivery optimization lies in its ability to generate the **optimal delivery route** by exhaustively evaluating all possible paths using memoization. Specifically, the Held-Karp algorithm ensures that **each delivery location is visited exactly once**, the route starts and ends at the courier hub, and the **total travel distance is minimized**. It satisfies **partial correctness** by always returning the optimal solution for the given input, and it guarantees **termination** after all subproblems have been processed. As a result, the DP algorithm is considered fully correct for small to medium-sized delivery datasets, producing accurate and constraint-compliant routes.

Conclusion

In conclusion, both Greedy and Dynamic Programming algorithms demonstrate correctness by producing valid delivery routes that meet the required constraints. The Greedy algorithm provides feasible solutions quickly but may sacrifice optimality, while the Dynamic Programming approach guarantees the shortest possible route at the cost of higher computation time. Together, they offer a balance between speed and accuracy, and their correctness is ensured through valid outputs and reliable termination for all input cases.

6.2 Complexity Analysis

6.2.1 Greedy Algorithm

The Greedy algorithm, implemented as the Nearest Neighbor, selects the closest unvisited city at each step until all cities are visited and returns to the starting point.

Time Complexity : O(n^2)
Where n is the number of delivery point

The time complexity of the Greedy algorithm for solving the Traveling Salesman Problem (TSP) is $O(n^2)$ in the best, average, and worst cases. This is because, at each step, the algorithm must compare all unvisited cities to select the nearest one, regardless of how the delivery points are arranged. While the runtime remains the same across all cases, the quality of the route may vary significantly — with the best case producing a near-optimal path and the worst case resulting in a highly inefficient one.

6.2.2 Dynamic Programming

The Dynamic Programming, implemented as the Traveling Salesman Problem computes the shortest path for every subset of cities and every possible current city.

Time Complexity : O(n^2 x 2^n)
Where n is the number of delivery point

The best, average, and worst-case time complexity of the Dynamic Programming (Held-Karp) algorithm for TSP is consistently $O(n^2 \times 2^n)$. This is because the algorithm must evaluate every subset of cities and every possible path regardless of how the input is arranged. Unlike Greedy algorithms, the input structure does not affect the number of operations—all possible combinations must be checked to guarantee the optimal solution. Therefore, the execution time remains the same in all cases, but the actual route quality is always optimal.

Conclusion

In conclusion, the chosen time complexities reflect the trade-off between speed and accuracy in solving the parcel delivery problem. The Greedy algorithm, with its $O(n^2)$ complexity, offers fast execution suitable for real-time use but may produce suboptimal routes. In contrast, the Dynamic Programming approach, with $O(n^2 \times 2^n)$ complexity, guarantees the optimal route but is only practical for smaller datasets due to its higher computational cost.

7 Testing Result

7.1 Purpose of Testing

Purpose of testing is to determine the best algorithm between Dynamic Programming and Greedy Algorithm in parcel delivery route, that is, the Travelling Salesman Program. By having 5 different route matrices, this project can compare which of the two algorithms can have the best route optimization.

7.2 Test Environment

- Machine Specifications

Components	Specification
Laptop	Intel core i7-8750H, 2.20GHz
Eclipse app	Java language

7.3 Test-case

Graph 1:

```
{ 0, 10, 15, 20, 30, 35 },
{ 10, 0, 18, 25, 40, 45 },
{ 15, 18, 0, 22, 25, 50 },
{ 20, 25, 22, 0, 30, 28 },
{ 30, 40, 25, 30, 0, 12 },
{ 35, 45, 50, 28, 12, 0 }
```

Graph 2:

```
{ 0, 12, 18, 23, 40, 35 },
{ 12, 0, 20, 30, 25, 22 },
{ 18, 20, 0, 14, 28, 32 },
{ 23, 30, 14, 0, 16, 24 },
{ 40, 25, 28, 16, 0, 10 },
{ 35, 22, 32, 24, 10, 0 }
```

Graph 3:

```
{ 0, 10, 50, 45, 60, 55 },
{ 10, 0, 20, 30, 35, 40 },
{ 50, 20, 0, 15, 25, 30 },
{ 45, 30, 15, 0, 20, 25 },
{ 60, 35, 25, 20, 0, 10 },
{ 55, 40, 30, 25, 10, 0 }
```

Graph 4:

```
{ 0, 14, 35, 28, 42, 36 },
{ 14, 0, 18, 26, 33, 31 },
{ 35, 18, 0, 12, 15, 20 },
{ 28, 26, 12, 0, 10, 17 },
{ 42, 33, 15, 10, 0, 8 },
{ 36, 31, 20, 17, 8, 0 }
```

Graph 5:

```
{ 0, 13, 17, 25, 30, 29 },
{ 13, 0, 14, 22, 27, 24 },
{ 17, 14, 0, 10, 20, 19 },
{ 25, 22, 10, 0, 15, 14 },
{ 30, 27, 20, 15, 0, 11 },
{ 29, 24, 19, 14, 11, 0 }
```

7.4 Final Ouput

Graph 1:

```
Greedy Route: 0 -> 1 -> 2 -> 3 -> 5 -> 4 -> 0
Greedy Route Cost: 120
DP Route: 0 -> 0 -> 3 -> 5 -> 4 -> 2 -> 1 -> 0
DP Route Cost: 113
```

Graph 2:

```
Greedy Route: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0
Greedy Route Cost: 107
DP Route: 0 -> 0 -> 2 -> 3 -> 4 -> 5 -> 1 -> 0
DP Route Cost: 92
```

Graph 3:

```
Greedy Route: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0
Greedy Route Cost: 130
DP Route: 0 -> 0 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0
DP Route Cost: 130
```

Graph 4:

```
Greedy Route: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0
Greedy Route Cost: 98
DP Route: 0 -> 0 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0
DP Route Cost: 98
```

Graph 5:

```
Greedy Route: 0 -> 1 -> 2 -> 3 -> 5 -> 4 -> 0
Greedy Route Cost: 92
DP Route: 0 -> 0 -> 2 -> 3 -> 4 -> 5 -> 1 -> 0
DP Route Cost: 90
```

7.5 Observation

As seen above, out of 5 routes, 3 of the test cases have DP outperforms Greedy algorithm. In the other two testcases, both had equal total distance travelled. In terms of time taken, DP should have theoretically taken more time than Greedy, but due to overhead in having small route.

Conclusion

By the observation and discussions, it appears that parcel delivery routes can be optimized automatically with both Greedy and Dynamic Programming, instead of manual heuristic thinking by the Courier. While Greedy always pick the best local option, it does not always have the optimal path in the long run, compared to Dynamic Programming, which already explores all possible paths, and choose the objectively best path.

References

1. Choudhary, M., & Kumar, D. (2021). *A comparative study of greedy and dynamic programming techniques for route optimization problems*. International Journal of Computer Applications, 183(10), 1–6. https://doi.org/10.5120/ijca2021921141

2. Yadav, R., Singh, A., & Sharma, N. (2019). *Application of dynamic programming in optimization problems*. International Journal of Innovative Technology and Exploring Engineering, 8(12), 1451–1455. https://doi.org/10.35940/ijitee.L3940.1081219

3. Pisinger, D., & Røpke, S. (2007). *A general heuristic for vehicle routing problems*. Computers & Operations Research, 34(8), 2403–2435. https://doi.org/10.1016/j.cor.2005.09.012

4. Skiena, S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer. https://doi.org/10.1007/978-1-84800-070-4

5. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley. https://algs4.cs.princeton.edu/home/

Appendix A

Distance Matrix

```
public class DistanceMatrix {
  public static int[][] getMatrix() {
     return new int[][] {
          { 0, 13, 17, 25, 30, 29 },
        { 13, 0, 14, 22, 27, 24 },
        { 17, 14, 0, 10, 20, 19 },
        { 25, 22, 10, 0, 15, 14 },
        { 30, 27, 20, 15, 0, 11 },
        { 29, 24, 19, 14, 11, 0 }
     };
  }
}
```

Main Method

```
public class MainMethod {
  public static void main(String[] args) {
     int[][] dist = DistanceMatrix.getMatrix();
     int start = 0;
     // Measure time for Greedy
     long startTimeGreedy = System.nanoTime();
     int greedyCost = TspGreedy.tspGreedy(dist, start);
     long endTimeGreedy = System.nanoTime();
     double elapsedGreedy = (endTimeGreedy - startTimeGreedy) / 1_000_000.0;
     System.out.printf("Greedy Time Taken: %.3f ms\n", elapsedGreedy);
     System.out.println();
     // Measure time for DP
     long startTimeDP = System.nanoTime();
     int dpCost = TspDP.tspDP(dist);
     long endTimeDP = System.nanoTime();
     double elapsedDP = (endTimeDP - startTimeDP) / 1_000_000.0;
     System.out.printf("DP Time Taken: %.3f ms\n", elapsedDP);
  }
}
```

Dynamic Programming

```
import java.util.*;
public class TspDP {
  public static int tspDP(int[][] dist) {
     int n = dist.length;
     int[][] dp = new int[1 << n][n];
     int[][] parent = new int[1 << n][n]; // to reconstruct path
     for (int[] row : dp)
        Arrays.fill(row, Integer.MAX_VALUE / 2);
     for (int[] row : parent)
        Arrays.fill(row, -1);
     dp[1][0] = 0;
     for (int mask = 1; mask < (1 << n); mask++) {
```

```
            for (int u = 0; u < n; u++) {
                if ((mask & (1 << u)) == 0) continue;
                for (int v = 0; v < n; v++) {
                    if ((mask & (1 << v)) != 0) continue;
                    int nextMask = mask | (1 << v);
                    int newCost = dp[mask][u] + dist[u][v];
                    if (newCost < dp[nextMask][v]) {
                        dp[nextMask][v] = newCost;
                        parent[nextMask][v] = u;
                    }
                }
            }
        }
        int minCost = Integer.MAX_VALUE;
        int last = -1;
        int fullMask = (1 << n) - 1;
        for (int i = 1; i < n; i++) {
            int cost = dp[fullMask][i] + dist[i][0];
            if (cost < minCost) {
                minCost = cost;
                last = i;
            }
        }
        // Reconstruct path
        List<Integer> path = new ArrayList<>();
        int mask = fullMask;
        while (last != -1) {
            path.add(last);
            int temp = parent[mask][last];
            mask ^= (1 << last);
            last = temp;
        }
        path.add(0); // start
        Collections.reverse(path);
        path.add(0); // return to hub
        // Print route
        System.out.print("DP Route: ");
        for (int i = 0; i < path.size(); i++) {
            System.out.print(path.get(i));
            if (i < path.size() - 1) System.out.print(" -> ");
        }
        System.out.println();
        System.out.println("DP Route Cost: " + minCost);
        return minCost;
    }
}
```

Greedy Algorithm

```java
import java.util.Arrays;
public class TspGreedy {
  public static int tspGreedy(int[][] dist, int start) {
     int n = dist.length;
     boolean[] visited = new boolean[n];
     int[] route = new int[n + 1];
     int totalCost = 0;
     int current = start;
     visited[current] = true;
     route[0] = current;
     for (int i = 1; i < n; i++) {
        int next = -1;
        int minDist = Integer.MAX_VALUE;
        for (int j = 0; j < n; j++) {
           if (!visited[j] && dist[current][j] < minDist) {
              minDist = dist[current][j];
              next = j;
           }
        }
        visited[next] = true;
        route[i] = next;
        totalCost += minDist;
        current = next;
     }
     // Return to start (optional — comment this out if not needed)
     totalCost += dist[current][start];
     route[n] = start;
     // Output route
     System.out.print("Greedy Route: ");
     for (int i = 0; i <= n; i++) {
        System.out.print(route[i] + (i < n ? " -> " : "\n"));
     }
     System.out.println("Greedy Route Cost: " + totalCost);
     return totalCost;
  }
}
```

Appendix B

<table>
<tr><td>Group Name</td><td>11</td></tr>
<tr><td>Members</td><td>
<table>
<tr><th>Name</th><th>Email</th><th>Phone number</th></tr>
<tr><td>Muhammad Syafi bin Abdul Nasir</td><td>214321@student.upm.edu.my</td><td>+601119546043</td></tr>
<tr><td>Ariff Ameer Aizad bin Nordin</td><td>214037@student.upm.edu.my</td><td>+60172004132</td></tr>
<tr><td>Muhammad Fazrul Hafizi bin Anuar</td><td>214271@student.upm.edu.my</td><td>+60132916220</td></tr>
<tr><td>Muhammad Arif Zahiruddin bin Yassir</td><td>214264@student.upm.edu.my</td><td>+60136008690</td></tr>
</table>
</td></tr>
<tr><td>Problem scenario description</td><td>Couriers take inefficient routes due to manual planning, causing delays, higher costs, and poor adaptability to changes. An automated system is needed to find the shortest delivery route using algorithms.</td></tr>
<tr><td>Why it is important</td><td>● leads to significant cost reductions for delivery companies.
● vastly improves customer satisfaction
● contribute to environmental sustainability</td></tr>
<tr><td>Problem specification</td><td>Given a set of delivery locations, determine the shortest possible route that starts and ends at the courier hub, visits each location exactly once, and adapts to changes in real time using algorithmic methods.</td></tr>
<tr><td>Potential solutions</td><td>Use Greedy for fast, approximate routing and Dynamic Programming for accurate, optimal delivery routes.</td></tr>
</table>

Sketch (framework, flow, interface)