FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

DEPARTMENT OF COMMUNICATION TECHNOLOGY AND NETWORK

SEMESTER 2 2024/2025

COURSE : **Bachelor Of Computer Science (Computer Network) With Honours**

COURSE CODE : **CSC4202**

PROJECT : **Optimizing Cargo Loading Using the 0/1 Knapsack Algorithm**

LECTURER : **Dr. Nur Arzilawati Binti Md Yunus**

NAME	**MATRIC NUMBER**
1. TAN KAH JUN	217075
2. CHUA HUI QI	215091
3. LOK YONG XUE	215524
4. TAN JIA QING	217067

Github Link: https://github.com/kahjun-0504/CSC4202-Project

1.0 Introduction

1.1 Scenario and Problem

DHL, a global leader in logistics and supply chain solutions, has transformed the cargo transport industry by combining advanced technology with decades of operational expertise. Operating in over 220 countries, DHL handles a wide spectrum of logistics services from express parcel delivery to large-scale freight movement across land, sea, and air. Its reputation for speed and reliability has made it a key logistics partner for e-commerce platforms, manufacturers, and multinational corporations.

Despite this global reach, DHL faces an increasingly critical operational issue: inefficient truck loading practices. With growing demand for faster and more cost-effective deliveries, the pressure on dispatch operations has intensified. Internal reviews have shown that trucks are often underloaded or inefficiently packed due to manual planning or rule-of-thumb methods. In many cases, dispatchers rely on experience rather than structured decision-making tools, which leads to inconsistencies in cargo prioritization and delivery performance.

For example, shipments containing high-priority or time-sensitive items may be delayed or underutilized if they are grouped with lower-priority goods in an inefficient manner. Such scenarios result in avoidable costs such as more delivery runs, increased fuel consumption, extended driver hours, and delayed shipments.

When scaled across hundreds of deliveries per day, these inefficiencies can significantly affect both operational costs and customer satisfaction. Delays, missed priorities, and partial shipments not only hurt DHL’s performance metrics but also disrupt the operations of businesses depending on timely cargo arrivals.

1.2 Importance of Optimal Cargo Loading

Finding an optimal solution for this cargo loading challenge is crucial for several reasons:

1. **Operational Efficiency and Cost Reduction:**

   DHL's implementation of an algorithmic cargo loading system addresses the core inefficiency of underutilized transport capacity. By maximizing load efficiency through the 0/1 Knapsack Problem approach, the company can ensure each shipment delivers the highest possible value per trip. This directly translates to reduced operational costs through fewer trips, optimized fuel usage, and better asset utilization.

2. **Customer Satisfaction and Service Quality:**

   The optimization algorithm enhances customer satisfaction by reducing delays, preventing partial shipments, and improving delivery accuracy. When cargo is loaded optimally, customers receive their goods faster and more reliably, leading to increased customer retention and positive reputation in the competitive logistics market.

3. **Environmental Impact and Sustainability:**

   By reducing the number of trips required and optimizing fuel usage, DHL's solution significantly reduces their carbon footprint. This environmental responsibility aligns with growing corporate sustainability goals and regulatory requirements.

4. **Competitive Advantage and Market Leadership:**

   The adoption of algorithmic intelligence for cargo optimization positions DHL as a forward-thinking logistics leader, setting new industry standards. This technological advancement provides a significant competitive edge in the rapidly evolving logistics sector, attracting more business partners and establishing the company as an innovation leader.

5. **Revenue Optimization:**

   The algorithm's focus on maximizing cargo value per trip directly impacts revenue generation. By ensuring that high-value items are prioritized and space is utilized optimally, DHL can increase profitability while maintaining competitive pricing for their logistics services.

1.3 Algorithm Suitability Review

To address DHL's cargo loading inefficiencies, several algorithmic approaches are assessed based solely on their effectiveness in maximizing cargo value under a weight constraint. This constraint reflects the real-world operational limit where each truck can carry only up to a specific total weight, regardless of cargo volume or shape.

1. **Sorting**

- Strengths:
    - Simple and fast to implement.
    - Enables quick ranking of items by value-to-weight ratio.
    - Useful as a preprocessing step to prioritize higher-efficiency cargo.

- Weaknesses:
    - Cannot guarantee optimal selection under strict weight limits.
    - Ignores combinations of items that may yield a better overall value.
    - Oversimplifies complex loading decisions by treating them as linear.

2. **Divide and Conquer (DAC)**

- Strengths:
    - Breaks the problem into smaller subproblems (e.g., by route or shipment category).
    - Allows for partial optimization in segmented contexts.
    - Scales well for distributed cargo management systems.

- Weaknesses:
    - May not find globally optimal cargo sets unless carefully merged.
    - Not inherently designed to optimize cumulative cargo value.
    - Assumes independent subproblems, which may not align with a global weight limit.

3. **Dynamic Programming (DP)**

- Strengths:
    - Provides an optimal solution for weight-constrained cargo selection using the 0/1 Knapsack Problem.
    - Evaluates all viable combinations of cargo within the weight limit.
    - Well-suited for discrete cargo decisions (i.e., include or exclude each item).
    - Handles discrete cargo items effectively, determining optimal inclusion/exclusion decisions.

- Weaknesses:
    - Computationally intensive for large item sets.
    - Requires careful modeling of item weights and values.
    - May need optimization for real-time decision-making.

4. Greedy Algorithms

- Strengths:
    - Very fast and easy to implement.
    - Prioritizes items with the best immediate value-to-weight ratio.
    - Useful for quick approximations or when time/resources are limited.

- Weaknesses:
    - Does not guarantee optimal cargo value.
    - Early selections may block higher-value combinations due to weight exhaustion.
    - Cannot revise earlier decisions, leading to suboptimal total loads.

5. Graph Algorithms

- Strengths:
    - Can model complex dependencies or constraints, if any exist (e.g., loading order).
    - Useful for visualizing cargo groupings or routing with weight limits.

- Weaknesses:
    - Overly complex for simple weight-based value maximization.
    - Requires additional layers to perform optimization.
    - Not naturally suited to solving knapsack-type problems.

Dynamic Programming (0/1 Knapsack): The Optimized Choice

For DHL's cargo loading optimization under a single weight constraint, Dynamic Programming using the 0/1 Knapsack Problem is the most effective approach. It allows:

- Maximization of total cargo value without exceeding the truck's weight capacity.
- Evaluation of all feasible item combinations for the best outcome.
- Systematic, reproducible decision-making compared to manual or heuristic methods.

While other algorithms may be useful in specific contexts (e.g., sorting for preprocessing, greedy for speed), Dynamic Programming offers the rigor and reliability needed to optimize cargo value across DHL's large-scale operations.

2.0 Scenario Model Development

2.1 Overview

The optimized cargo loading algorithm focuses on efficiently selecting which shipments to load onto a truck when there are more shipments than the truck can carry. This becomes especially important when trucks face strict weight limits, and not all shipments can be delivered in a single trip.

The decision on which shipments to load is based on a **priority score**, which combines revenue, urgency, and service level (e.g., premium vs. standard). This ensures high-value and time-sensitive shipments are prioritized.

The scenario can be divided into two main components:

- Determine the priority score of each shipment based on business value.
- Select the combination of shipments that maximizes the total priority score without exceeding the truck's weight limit.

2.2 Cargo Value Prioritization

The process of determining which cargo to prioritize becomes essential when the available shipments exceed the truck's maximum weight capacity. For a single shipment, loading is straightforward. However, when there are multiple shipments with different characteristics, a structured method is needed to decide which cargo should be selected for delivery.

The prioritization is influenced by several key factors, including the value of the shipment, its urgency, and the level of service required. These factors guide the decision on which shipments offer the greatest overall benefit when space is limited.

Shipment value serves as the initial layer in our cargo prioritization strategy. High-value cargo is given preference to maximize the efficiency of each trip. This helps ensure that the truck carries shipments that contribute more significantly to business goals.

Urgency adds another dimension to the decision-making process. Shipments that are time-sensitive or near their delivery deadline may be prioritized higher, even if their value is lower. This ensures timely delivery and helps maintain service reliability.

Service level introduces a further layer of consideration. Premium service shipments may receive higher priority over standard ones to meet customer expectations and contractual obligations.

While these factors help shape the prioritization process, complexity arises when multiple shipments score similarly across value, urgency, or service level. In such cases, deciding which shipment to include becomes more challenging, especially under strict weight constraints.

To manage these complexities, a prioritization strategy is developed that balances all the key factors. This approach allows for a fair and efficient selection of cargo that meets both operational and customer-oriented goals.

The method for calculating and comparing shipment priority will be further explored in Section 3.2, where a cargo value model will be introduced to formalize this selection process.

2.3 Constraints and Challenges

A key challenge in optimizing cargo loading lies in managing competing constraints, particularly weight and space, while maximizing shipment value. Although weight is often the primary constraint considered, overlooking cargo volume or dimensional fit can lead to suboptimal or even infeasible loading plans. For instance, high-value cargo may fall within the weight limit but exceed the spatial capacity of the delivery vehicle.

Another core difficulty lies in accurately quantifying shipment priority. The model incorporates multiple factors such as expected revenue, delivery urgency, and service level (e.g., premium vs. standard). However, determining the correct balance between these factors introduces subjective judgment. Misjudging the importance of one factor over another, such as overemphasizing urgency at the expense of overall revenue may reduce the total value delivered.

Moreover, shipments often vary widely in shape and packaging, which complicates space allocation. Even if the total volume appears acceptable, the geometry of individual items can make certain loading combinations impractical. This geometric constraint is especially significant for irregularly shaped cargo or items requiring upright placement, separation, or temperature control.

The algorithm also assumes accurate and consistent data on cargo attributes. In practice, missing or inaccurate data, such as an unrecorded delivery deadline or incorrect weight can undermine optimization efforts. In such cases, human oversight and exception handling may still be necessary.

Finally, the model focuses on what to load (i.e., cargo selection) rather than how to deliver (i.e., route planning). As a result, selected shipments may be optimal in value but suboptimal in terms of delivery sequence or routing efficiency. For example, a prioritized shipment might require a substantial detour, increasing delivery time or cost.

These constraints underscore the importance of a holistic approach to cargo optimization, one that balances quantitative modelling with operational realities and allows for flexibility when needed.

3.0 Algorithm Design

3.1 Overview

The optimized cargo loading algorithm consists of two main components:

- Determining the prioritization of cargo shipments based on their value
- Selecting the optimal combination of shipments to load within a fixed weight limit

The first component ranks shipments using a value-based prioritization approach. This value may reflect factors such as urgency, profitability, or delivery deadlines. The second component models the selection process as a **0/1 Knapsack Problem**, where each shipment has a defined weight and value, and the total weight of selected shipments must not exceed the vehicle's maximum load capacity.

We represent this scenario as a bounded optimization problem:

- Each **shipment** is an item with a weight and a calculated value score.
- The **truck** has a predefined maximum weight capacity.
- The **goal** is to select a combination of shipments that maximizes total value while staying within the weight limit.

The algorithm evaluates all possible combinations of available shipments using a dynamic programming approach. Each shipment is characterized by a value and weight. The goal is to select a subset of shipments that maximizes total value without exceeding the truck’s maximum weight capacity.

The steps include:

- Assign a value and weight to each shipment.
- Construct a dynamic programming table to evaluate the maximum value achievable for each possible weight limit up to the truck’s capacity.
- Backtrack through the table to determine the set of shipments to load.

Key modeling elements include:

- **Item (Shipment)**: Defined by its weight and value.
- **Constraint**: Maximum truck load weight (e.g., in kg or tonnes).
- **Optimization**: Maximize total shipment value without exceeding the weight limit.

This algorithm ensures high-value cargo is prioritized and fully utilizes available capacity for each shipment round. More complex considerations (e.g., shipment

deadlines or client tiers) can be integrated into the value scoring formula, which will be detailed in Section 3.2.

3.2 Cargo Value Modelling

For single shipment orders, the truck proceeds directly to the destination. However, in cases involving multiple shipment orders, a prioritization mechanism is essential to determine which shipments should be selected and delivered first, especially when the truck's weight capacity is limited.

To prioritize shipments effectively, each order is assigned a **priority score** that incorporates three key attributes: **revenue**, **urgency**, and **service level** (premium vs. standard). The score is calculated using the following formula:

$$priority\_score = \alpha \cdot revenue + \beta \cdot urgency + \gamma \cdot service\_level$$

Where:

- **revenue**: The amount earned by fulfilling the shipment.
- **urgency**: Calculated as $\frac{1}{time\_remaining\_in\_days}$, assigning higher scores to shipments with less time remaining.
- **service_level**: Set to **10** for **premium service**, and **0** for **standard service**, giving premium shipments significantly higher weight.
- **α, β, γ**: Weight parameters that can be adjusted based on business priorities, such as maximizing profit, ensuring timely delivery, or prioritizing premium customers. A **larger γ** ensures premium shipments are prioritized above all else when required.

Once the priority score is computed for each shipment, the values are used to guide the selection of shipments during the optimization process (e.g., using the 0/1 knapsack algorithm). The algorithm evaluates all possible combinations of shipments and selects the set that maximizes the total priority score without exceeding the truck's weight capacity, as defined by the 0/1 knapsack problem.

The following are the default weight parameter used to compute the priority score for each shipment order:

Parameter	Description	Value
α (alpha)	Importance of revenue	0.5
β (beta)	Importance of urgency (based on days remaining)	0.3
γ (gamma)	Importance of service level (Premium vs. Standard)	0.7

When the truck receives multiple shipment orders, and the total weight exceeds its capacity, a prioritization mechanism is required to determine which shipments should be selected for delivery. Each order is evaluated based on revenue, urgency, and service level (premium or standard). The goal is to maximize the overall value delivered within the truck's weight limit. Shipments with higher revenue, more urgent deadlines (i.e., fewer days remaining), or premium service levels are assigned higher priority scores. This ensures that time-sensitive and high-value deliveries are prioritized.

The shipment selection process follows these steps:

1. **If the truck receives a single shipment order**, proceed with delivering that shipment directly to the destination.

2. **If multiple shipment orders are received and their combined weight is within the truck's capacity**, load and deliver all shipments.

3. **If the total weight exceeds the truck's capacity**, proceed as follows:

    1. **Compute a priority score** for each shipment using the formula:

$$priority\_score = \alpha \cdot revenue + \beta \cdot urgency + \gamma \cdot service\_level$$

    - *Urgency* is calculated as $\frac{1}{time\_remaining\_in\_days}$, giving higher scores to more urgent deliveries.

    - *Service level* is set to 10 for premium shipments and 0 for standard shipments.

2. **Apply the 0/1 knapsack algorithm** to select the optimal combination of shipments:
    - Each shipment's **value** is its priority score.
    - Each shipment's **weight** is its actual weight (in kg).
    - The algorithm chooses the combination of shipments that **maximizes the total priority score** without exceeding the truck's weight capacity.
3. **Deliver the selected shipments**. The order of delivery is flexible, as the prioritization has already been handled during the selection phase.

This structured approach ensures that high-value, urgent, and premium shipments are always prioritized when resources are limited, enhancing both efficiency and customer satisfaction.

3.3 Example

To demonstrate the cargo loading optimization algorithm in action, consider the following real-world scenario faced by DHL:

A DHL distribution hub receives five shipment orders simultaneously. Each shipment has a different revenue, time remaining before the delivery deadline, and service level. The truck assigned to the deliveries has a maximum weight capacity of 30 kg. The challenge is to determine which shipments to load in order to maximize the total priority score without exceeding the weight limit.

Step 1: Shipment Data and Priority Score Calculation

Table 1: Shipment Information

Shipment ID	Weight (kg)	Revenue (RM)	Time Remaining (Days)	Service Level
A	10	200	1	Premium
B	8	180	3	Standard
C	5	120	2	Premium

D	12	220	4	Standard
E	7	150	1	Standard

Each shipment is evaluated using the priority score formula:

$$\text{priority\_score} = \alpha \cdot \text{revenue} + \beta \cdot \frac{1}{\text{time\_remaining\_in\_days}} + \gamma \cdot \text{service\_level}$$

Where:

- α = 0.5 (importance of revenue)
- β = 0.3 (importance of urgency)
- γ = 0.7 (importance of premium service)
- Service Level = 10 for Premium, 0 for Standard

Table 2: Calculated Priority Scores for each Shipment ID

Shipment ID	**Revenue**	**1 / Days**	**Service**	**Priority Score**
A	200	1.000	10	0.5×200 + 0.3×1 + 0.7×10 = 100 + 0.3 + 7 = 107.3 **≈ 107**
B	180	0.333	0	0.5×180 + 0.3×0.333 + 0 = 90 + 0.1 = 90.1 **≈ 90**
C	120	0.5	10	0.5×120 + 0.3×0.5 + 0.7×10 = 60 + 0.15 + 7 = 67.15 **≈ 67**

D	220	0.25	0	0.5×220 + 0.3×0.25 + 0 = 110 + 0.075 = 110.075 **≈ 110**
E	150	1.000	0	0.5×150 + 0.3×1 + 0 = 75 + 0.3 = 75.3 **≈ 75**

Step 2: Apply 0/1 Knapsack Algorithm

The goal is to select shipments that maximize total priority score without exceeding the 30 kg weight limit. The problem is modelled using the 0/1 Knapsack dynamic programming approach as shown in Table 3.

Table 3: Knapsack Dynamic Programming Table for Optimal Shipment Selection

			Capacity																											
		i	0	…	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
V	W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
107	10	0	0	0	0	0	0	0	0	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107	107
90	8	0	0	0	0	0	0	90	90	107	107	107	107	107	107	107	107	197	197	197	197	197	197	197	197	197	197	197	197	197
67	5	0	0	0	67	67	67	90	90	107	107	107	157	157	174	174	174	197	197	197	197	197	264	264	264	264	264	264	264	264
110	12	0	0	0	67	67	67	90	90	107	107	110	157	157	174	174	177	197	197	200	200	217	264	264	267	267	284	284	284	307
75	7	0	0	0	67	67	75	90	90	107	107	142	157	157	174	174	182	197	197	232	232	249	264	264	272	272	284	284	292	339

Step 3: Extracting the Optimal Shipment Set

From Table 2, the maximum total priority score achievable within the 30 kg weight constraint is 339.85. To identify the selected shipments, we backtrack through the table:

339 - 75(index 5's priority score) = 264

264 – 67 (index 3's priority score) = 197

197 - 90 (index 2's priority score) = 107

107 - 107 (index 1's priority score) = 0

From the above calculation we can confirm that the optimal choice will be Shipments A, B, C and E (index 1, 2, 3 and 5), giving a total weight of 30kg (within the weight limit) and yielding the maximum priority score of 339.

Step 4: Final Shipment Selection

Table 4: Final Shipment Selection Based on Optimization

Shipment ID	Weight (kg)	Priority Score
A	10	107
B	8	90
C	5	67
E	7	75
Total	***30***	***339***

The selected shipments in Table 4 will be loaded for the next delivery. This combination balances profitability, delivery urgency, and premium service considerations, making the most efficient use of the truck's capacity under current business constraints.

4.0 Algorithm Specification

4.1 Problem Definition

4.1.1 Background

DHL, a global logistics provider, faces challenges in optimally loading cargo onto trucks with limited weight capacities. Manual loading decisions often result in underutilization of truck space, higher operational costs, shipment delays, and reduced customer satisfaction.

4.1.2 Problem Statement

Given a set of shipments, each with attributes such as revenue, urgency, service level, and weight, the goal is to select a subset of shipments that maximizes total shipment value (priority score), while not exceeding the truck's maximum allowable weight.

4.2 Input and Output Specification

4.2.1 Input

- n: Total number of shipment orders.
- For each shipment $I \in \{1, 2, ..., n\}$:
    - revenue_i: Revenue earned if shipment is delivered.
    - time_remaining_i: Days remaining before delivery deadline.
    - service_level_i: 10 if premium, 0 if standard.
    - weight_i: Weight of the shipment in kg.
- W: Maximum weight capacity of the truck (kg).

4.2.2 Output

- A list of selected shipment IDs.
- The total priority score achieved.
- The total weight of selected shipments.

4.3 Mathematical Model

4.3.1 Priority Score Calculation

For each shipment, a priority score is calculated as:

$$\text{priority\_score}_i = \alpha \cdot \text{revenue}_i + \beta \cdot \frac{1}{time\_remaining\_in\_days_i} + \gamma \cdot \text{service\_level}_i$$

Where:

- α=0.5 (importance of revenue)
- β=0.3 (importance of urgency)
- γ=0.7 (importance of service level)

Important: The final result of the priority score will be rounded to the nearest whole number.

4.3.2 Optimization Objective

The goal of the algorithm is to select a combination of shipments that yields the maximum total priority score while ensuring that the total weight of the selected shipments does not exceed the truck's maximum capacity.

Formally, we aim to:

$$max \sum_{i=1}^{n} priority\_score_i \cdot x_i$$

Subject to:

$$\sum_{i=1}^{n} weight_i \cdot x_i \leq W$$

Where:

- $x_i \in \{0,1\}$ is a binary decision variable:
  - $x_i=1$: shipment i is selected.
  - $x_i=0$: shipment i is not selected.
- priority_score_i is the pre-computed score that reflects the combined importance of revenue, urgency, and service level.
- weight_i is the weight of shipment i.
- W is the truck's maximum weight capacity.

Explanation:

- The objective function sums up the priority scores of all selected shipments.
- The constraint ensures that the total weight of the selected shipments does not exceed the truck's maximum load capacity.
- The binary nature of x_i reflects the fact that each shipment can either be fully included or excluded (hence, 0/1 Knapsack).

This formulation allows the algorithm to make an optimal trade-off between revenue, urgency, and service level, automatically balancing business objectives based on the weighted priority score formula.

4.4 Algorithm Approach

4.4.1 Algorithm Type

The algorithm uses Dynamic Programming (DP) to solve the 0/1 Knapsack problem optimally. This approach systematically evaluates all possible combinations of shipments by building up solutions to smaller subproblems, ensuring that the final selection of shipments maximizes the total priority score without exceeding the truck’s weight capacity.

4.4.2 Why Dynamic Programming?

Dynamic Programming is chosen because it guarantees an optimal solution by exhaustively considering every possible combination of shipments under the given weight constraint. This method is particularly suitable for the problem at hand, as it effectively handles the trade-offs between multiple prioritization factors such as revenue, urgency, and service level. Additionally, Dynamic Programming remains computationally efficient and practical for moderate problem sizes, which aligns well with the typical scale of daily truck loading operations faced by DHL.

4.5 Algorithm Description

4.5.1 Key Idea

Construct a DP table dp[i][w] where:

- i represents the shipments index.
- w represents the truck's remaining weight capacity.

4.5.2 Recurrence Relation

$$dp[i][w] = \begin{cases} dp[i-1][w] & \text{if } weight_i > w \\ \max(dp[i-1][w], dp[i-1][w - weight_i] + priority\_score_i) & \text{otherwise} \end{cases}$$

4.5.3 Algorithm Steps

1. **Preprocessing:**
   Calculate priority scores for all shipments using the given formula.
2. **DP Table Construction:**
   Initialize dp[0][w] = 0 for all w.
   Fill the table using the recurrence relation.
3. **Backtracking:**
   Starting from dp[n][W], backtrack to identify which shipments were included in the optimal solution.

4.6 Correctness Justification

Briefly explain the correctness of Dynamic Programming by analysing the result.

4.7 Complexity Analysis

Analyze the worst-case, best-case, and average-case time complexity of the algorithm to evaluate its efficiency and scalability.

4.8 Assumptions and Limitations

- Truck capacity constraint is weight-based only (volume constraints are not modeled).
- Shipment data (revenue, urgency, service level, weight) is accurate and complete.
- Time remaining (time_remaining_i) must be greater than zero.
- Does not consider delivery routing or sequencing after cargo selection.

4.9 Implementation of Algorithm

- Programming Language: Java
- Shipment data is input as a list/array of weights, revenues, urgency, and service level.
- DP table is initialized and filled according to the recurrence relation.
- Backtracking function retrieves the optimal shipment set.

4.10 Program Testing

- Validate correctness using test cases including:
    - Example from Section 3.3.
    - Scenarios where:
        - All shipments fit.
        - Only partial shipments fit.
        - No shipment fits.
- Validate optimal priority score and selected shipment set for each scenario.

4.11 Documentation

Document the algorithm specification, including problem definition, input-output requirements, algorithm design, analysis, implementation details, and testing results. Maintain an organized online portfolio or documentation to ensure clarity and accessibility.

5.0 Algorithm Implementation

5.1 Coding

Refer to the appendix section.

5.2 Pseudocode

```
Function ComputePriorityScore(revenue, timeRemaining, serviceLevel):
    score = 0.5 * revenue + 0.3 * (1 / timeRemaining) + 0.7 * serviceLevel
    return rounded score

Function ReadCSV(filepath):
    Initialize empty list of shipments
    For each line in the CSV file:
        Parse id, revenue, timeRemaining, serviceLevel, weight
        Compute priorityScore using ComputePriorityScore()
        Add shipment to the list
    Return list of shipments

Function KnapsackOptimization(shipments, maxWeight):
    Initialize 2D array dp[n+1][W+1] for dynamic programming
    Initialize 2D array keep[n+1][W+1] to track selections

    For i from 1 to n:
        For w from 0 to W:
            If shipment[i-1].weight <= w:
                include = shipment[i-1].priorityScore + dp[i-1][w - weight]
                exclude = dp[i-1][w]
                If include > exclude:
                    dp[i][w] = include
```

keep[i][w] = true

Else:

dp[i][w] = exclude

Else:

dp[i][w] = dp[i-1][w]

Backtrack using keep[][] to get list of selected shipments

Return selected shipment IDs, total weight, and total score

5.3 Algorithm Descriptions

This project uses the **0/1 Knapsack Dynamic Programming Algorithm** to select a subset of shipments that maximizes a calculated **priority score** while ensuring the **total weight does not exceed the truck's capacity** (500 kg).

Key Components:

- **Priority Score Calculation:**

$$\text{priority\_score}_i = \alpha \cdot \text{revenue}_i + \beta \cdot \frac{1}{time\_remaining\_in\_days_i} + \gamma \cdot \text{service\_level}_i$$

  This custom formula prioritizes:

  - Higher revenue
  - More urgency (shorter time remaining)
  - Premium service level

- **Dynamic Programming (DP) Table:**

  - dp[i][w] holds the **maximum score** using the first i shipments with weight limit w.
  - keep[i][w] tracks whether shipment i is included in optimal solution for backtracking.

- **Backtracking:**

  - Starts from dp[n][W], traces which shipments were included.
  - Shipment IDs are collected and reversed to get the original order.

Advantages of this algorithm:

- Ensures **optimal selection** of shipments.
- Efficient for moderate input size.
- Easy to extend for additional constraints.

5.4 Demonstration

This is a simple demonstration of the example under section 5.0 Algorithm Design. Only the shipments that have high enough priority scores to influence the optimization decision are selected, while others are omitted from the result for brevity.

The CSV file named sample_shipments.csv contains 5 fields:

ShipmentID, Revenue, TimeRemaining, ServiceLevel, Weight.

Below is a sample of the first 5 lines in the CSV:

	A	B	C	D	E
1	1	202	2.85	10	18
2	2	370	1.13	0	65
3	3	206	4.86	0	14
4	4	171	3.93	10	8
5	5	288	4.71	0	16

For example, line 1 indicates:

- Shipment ID = 1
- Revenue = 202
- Time remaining = 2.85 hours
- Service level = 10
- Weight = 18 kg

The Java program reads this file, calculates priority scores, and runs the Knapsack algorithm. Only shipments contributing to the optimal score (while staying within the total 500 kg limit) are selected.

Please refer to Appendix for full sample CSV content and structure.

5.5 Results

```
<terminated> V4 [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (13 Jun 2025, 7:08:09 pm – 7:08:19 p
Enter path to shipment CSV file: C:\Users\User\Downloads\sample_shipments.csv

Final Shipment Selection Based on Optimization:
ShipmentID   Weight (kg)  Priority Score
001          18           108
003          14           103
004          8            93
005          16           144
006          18           67
007          11           101
009          27           157
012          5            126
013          12           122
014          5            125
015          15           186
016          65           197
019          16           137
020          10           68
021          11           177
022          9            175
024          5            86
025          37           136
026          18           151
027          6            185
029          16           137
033          8            122
034          41           124
035          15           60
036          25           141
038          5            94
040          10           178
045          10           67
046          10           153
047          15           97
049          5            58
050          13           84
-------------------------------------------------
Total        499          3959
```

<terminated> V4 [Java Application] C:\Program Files\Jav

```
--- All Shipment Priority Scores ---
ShipmentID   Priority Score
001           108
002           185
003           103
004           93
005           144
006           67
007           101
008           118
009           157
010           94
011           107
012           126
013           122
014           125
015           186
016           197
017           146
018           188
019           137
020           68
021           177
022           175
023           74
024           86
025           136
026           151
027           185
028           145
029           137
030           76
031           84
032           179
033           122
034           124
035           60
036           141
037           187
038           94
039           64
040           178
```

```
041             189
042             83
043             96
044             182
045             67
046             153
047             97
048             82
049             58
050             84
```

This Java program optimizes DHL shipment selection using a 0/1 Knapsack approach, where each shipment is assigned a *priority score* based on its shipment id, revenue, time remaining, and service level. The program reads shipment data from a CSV file, calculates integer-rounded priority scores, and selects the best combination of shipments whose total weight does not exceed 500 kg while maximizing the total priority score. The final output displays selected shipments with their IDs, weights, and scores, followed by the total weight and total priority score (3959). It also lists all shipments and their individual scores for reference.

6.0 Analysis of Algorithm

6.1 Correctness of Algorithm

Tracing the output of each possible input is impractical for large inputs. Therefore, it is important to demonstrate the correctness of the algorithm using a formal method. For this algorithm, we use the loop invariant technique to prove the correctness of our dynamic programming approach to solving the 0/1 Knapsack problem.

6.1.1 Problem Overview

In the 0/1 Knapsack problem, we aim to choose a subset of items (in this case, shipments) with given weights and values (priority scores), to maximize the total value without exceeding a weight capacity limit (W = 500).

We use a bottom-up dynamic programming (DP) approach to fill a 2D array dp[i][w], where:

- i is the number of shipments considered so far,
- w is the current weight capacity,
- dp[i][w] holds the maximum priority score achievable with the first i shipments and capacity w.

6.1.2 Loop Invariant for Dynamic Programming Table

Invariant Condition:

At the start of each iteration of the inner loop, dp[i][w] correctly stores the maximum total priority score that can be obtained using the first i shipments and with a knapsack capacity of exactly w.

Initialization:

Before the loops begin, the entire dp table is initialized to 0. This correctly represents the base case:

- If we use 0 items (i = 0), the best total score is 0 for all capacities.
- If the knapsack has 0 capacity (w = 0), no items can be included, and thus total score is also 0.

This base case satisfies the loop invariant.

Maintenance:

In each iteration of the nested loop:

for (int i = 1; i <= n; i++) {

for (int w = 0; w <= W; w++) {

We check whether the current item (i-1) can be included (weight <= w):

- Include case: if the item fits, we consider the score if it is included:

  include = priorityScore[i-1] + dp[i-1][w - weight[i-1]].

- Exclude case: if not included, we use dp[i-1][w].

The larger of these two is stored in dp[i][w], which maintains the invariant that dp[i][w] holds the best total score achievable for the first i items and weight w.

Termination:

When the loops complete, dp[n][W] contains the maximum score achievable using all n shipments within the weight limit W.

Additionally, the keep matrix stores the decisions made, which are then backtracked to reconstruct the optimal selection of shipments.

Thus, the final output derived from dp and keep reflects the correct and optimal subset of shipments based on the knapsack constraints.

6.1.3 Conclusion

Using the concept of loop invariants, we have shown that:

- The algorithm correctly initializes base cases,
- Maintains correctness at each step of the iteration,
- And provides the correct solution upon termination.

Hence, the 0/1 Knapsack implementation is correct and consistently computes the optimal selection of shipments based on their computed priority scores.

6.2 Time Complexity Analysis

This program uses dynamic programming to optimize the selection of shipments based on a calculated priority score. The overall time complexity includes several parts, which are analyzed below:

1. Priority Score Calculation

```
for (int i = 0; i < n; i++) {

    // Compute priority score

}
```

This loop runs once for each of the `n` shipments. Each calculation involves basic arithmetic operations, which take constant time. Therefore, the time complexity for this part is **O(n)**.

2. Dynamic Programming Table Construction

```
for (int i = 1; i <= n; i++) {

    for (int w = 0; w <= W; w++) {

        // Fill DP table using recurrence

    }

}
```

This nested loop iterates over all shipments (`n`) and all possible weight values up to the truck's capacity (`W`). Each cell in the table is filled using a constant-time operation. Thus, the time complexity for filling the table is **O(n × W)**.

3. Backtracking to Find Selected Shipments

```
for (int i = n; i > 0; i--) {

    if (dp[i][w] != dp[i - 1][w]) {

        // Track selected shipments

    }

}
```

This loop runs up to `n` times, once for each shipment, to trace back which shipments were selected. Therefore, the time complexity is **O(n)**.

4. Output of Selected Shipments

```
for (Shipment s : selected) {

    // Print details

}
```

This loop prints all selected shipments. In the worst case, all `n` shipments could be selected. So, the time complexity is **O(n)**.

When we combining all parts:

- Priority score calculation: O(n)
- DP table filling: O(n × W)
- Backtracking: O(n)
- Output: O(n)

Thus, the total time complexity is:

$$O(n \cdot W + n) = O(n \cdot W)$$

Since **O(n × W)** dominates **O(n)**, the overall time complexity is **O(n × W)**.

Best Case Scenario

The time complexity is **O(nW)**. This is because the dynamic programming method always fills the entire table, without skipping any part of the computation. For example, even if all shipments are too heavy and none can be added to the truck, the program still processes each cell in the DP table. The algorithm does not include any shortcuts or early exits, so it performs the same number of steps no matter how simple the input might be. Therefore, the best case still requires filling out the entire table.

Average Case Scenario

In the average case, where some shipments are selected and others are not, the algorithm still processes all **n × W** table entries. The actual results in the table may vary based on shipment weights and scores, but the number of operations stays the same. Since each shipment and weight value combination is considered, the time complexity remains **O(nW)** on average. The approach does not adjust the number of steps based on the input data, so the average-case performance is the same as in the best and worst cases.

Worst Case Scenario

In the worst-case scenario, the program processes every shipment and every possible truck weight from 0 to W. It fills a dynamic programming (DP) table of size (n + 1) × (W + 1), where **n** is the number of shipments and **W** is the truck's maximum weight capacity. For each entry in this table, the program checks whether including a shipment will improve the total priority score. This means it performs a constant-time operation for each cell in the table. As a result, the total number of operations in the worst case is proportional to n × W, giving a time complexity of **O(nW)**. This case happens regardless of how the shipment data is arranged, and it ensures that no possible option is missed.

One possible improvement to the current algorithm is to use a **Greedy approach** instead of dynamic programming. By selecting shipments based on their **priority score per unit weight**, the algorithm can sort the shipments and pick the most valuable ones first until the weight limit is reached. This reduces the time complexity from **O(n × W)** to **O(n log n)** due to sorting, followed by a single pass through the sorted list.

However, we did not implement this method in our program, as the Greedy algorithm does **not guarantee an optimal solution** for the 0/1 Knapsack problem. Our current use of dynamic programming, although slower, ensures we get the **best possible combination of shipments**. Given that our dataset is relatively small and manageable, the trade-off for optimality is acceptable.

7.0 Appendix

The following is the full content of the sample_shipments.csv file used in the demonstration. This file is in **Comma-Separated Values (CSV)** format, where each line represents one shipment record and each value is separated by a comma. The fields are ordered as follows:

Shipment ID, Revenue, **Time Remaining**, **Service Level**, **Weight**

	A	B	C	D	E
1	1	202	2.85	10	18
2	2	370	1.13	0	65
3	3	206	4.86	0	14
4	4	171	3.93	10	8
5	5	288	4.71	0	16
6	6	120	4.5	10	18
7	7	202	3.09	0	11
8	8	221	4.63	10	56
9	9	314	0.67	0	27
10	10	187	1.18	0	35
11	11	199	0.46	10	89
12	12	251	1.8	0	5
13	13	230	2.1	10	12
14	14	249	1.54	0	5
15	15	357	4.19	10	15
16	16	393	1.94	0	65
17	17	291	1.58	0	76
18	18	376	2.83	0	78
19	19	260	0.92	10	16
20	20	121	4.06	10	10
21	21	352	0.6	0	11
22	22	335	4.94	10	9
23	23	148	3.92	0	35
24	24	158	1.19	10	5
25	25	269	0.28	0	37

	A	B	C	D	E
26	26	287	4.12	10	18
27	27	370	3.61	0	6
28	28	289	3.71	0	49
29	29	274	3.91	0	16
30	30	150	0.6	0	54
31	31	154	1.95	10	80
32	32	343	0.8	10	61
33	33	230	4.35	10	8
34	34	234	3.21	10	41
35	35	120	1.82	0	15
36	36	266	0.55	10	25
37	37	373	1.73	0	95
38	38	188	1.79	0	5
39	39	113	3.72	10	19
40	40	341	3.28	10	10
41	41	364	4.46	10	82
42	42	152	2.49	10	39
43	43	191	0.82	0	38
44	44	363	3.64	0	76
45	45	134	3.86	0	10
46	46	305	2.92	0	10
47	47	180	3.91	10	15
48	48	149	2.6	10	49
49	49	101	2.73	10	5
50	50	153	2.28	10	13

This data was used as input for the optimization program to calculate priority scores and select shipments accordingly. The full dataset contains multiple rows, each representing a unique shipment entry.

Program Code:

```
 import java.util.*;
import java.io.*;

public class V4 {
    static class Shipment {
        int id;
        double revenue;
        double timeRemaining;
        int serviceLevel;
        int weight;
        double priorityScore;

        Shipment(int id, double revenue, double timeRemaining, int serviceLevel, int weight) {
            this.id = id;
            this.revenue = revenue;
            this.timeRemaining = timeRemaining;
            this.serviceLevel = serviceLevel;
            this.weight = weight;
            this.priorityScore = Math.round(0.5 * revenue + 0.3 * (1.0 / timeRemaining) + 0.7 *
serviceLevel);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter path to shipment CSV file: ");
        String filePath = scanner.nextLine();

        List<Shipment> shipmentList = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.trim().isEmpty()) continue; // skip empty lines
                String[] parts = line.split(",");

                int id = Integer.parseInt(parts[0]); // First column is ID
                double revenue = Double.parseDouble(parts[1]);
                double timeRemaining = Double.parseDouble(parts[2]);
                int serviceLevel = Integer.parseInt(parts[3]);
                int weight = Integer.parseInt(parts[4]);

                shipmentList.add(new Shipment(id, revenue, timeRemaining, serviceLevel,
weight));
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
            return;
        }

        int n = shipmentList.size();
        int W = 500;
        Shipment[] shipments = shipmentList.toArray(new Shipment[0]);
```

```
        double[][] dp = new double[n + 1][W + 1];
        boolean[][] keep = new boolean[n + 1][W + 1];

        // Fill DP table
        for (int i = 1; i <= n; i++) {
            for (int w = 0; w <= W; w++) {
                if (shipments[i - 1].weight <= w) {
                    double include = shipments[i - 1].priorityScore + dp[i - 1][w - shipments[i -
1].weight];
                    double exclude = dp[i - 1][w];
                    if (include > exclude) {
                        dp[i][w] = include;
                        keep[i][w] = true;
                    } else {
                        dp[i][w] = exclude;
                    }
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        // Backtrack to find selected shipments
        int w = W;
        List<Integer> selectedShipments = new ArrayList<>();
        int totalWeight = 0;
        for (int i = n; i > 0; i--) {
            if (keep[i][w]) {
                selectedShipments.add(shipments[i - 1].id);
                totalWeight += shipments[i - 1].weight;
                w -= shipments[i - 1].weight;
            }
        }

        // Output results
        Collections.reverse(selectedShipments);

        System.out.println("\nFinal Shipment Selection Based on Optimization:");
        System.out.printf("%-12s %-12s %-15s%n", "ShipmentID", "Weight (kg)", "Priority
Score");
        double totalScore = 0.0;

        for (int id : selectedShipments) {
            Shipment s = Arrays.stream(shipments).filter(sh -> sh.id == id).findFirst().get();
            System.out.printf("%03d          %-12d %-15d%n", s.id, s.weight, (int) s.priorityScore);
            totalScore += s.priorityScore;
        }

        System.out.println("-----------------------------------------------");
        System.out.printf("%-12s %-12d %-15.0f%n", "Total", totalWeight, totalScore);

        // Optional: Show each shipment's score
        System.out.println("\n--- All Shipment Priority Scores ---");
        System.out.printf("%-12s %-15s%n", "ShipmentID", "Priority Score");
```

```
        for (Shipment s : shipments) {
            System.out.printf("%03d          %-15d%n", s.id, (int) s.priorityScore);
        }
    }
}
```

Initial Project Plan

<table>
<tr><td>Group Name</td><td>Group 2</td></tr>
<tr><td>Members</td><td>
<table>
<tr><th>Name</th><th>Email</th><th>Phone number</th></tr>
<tr><td>Tan Kah Jun</td><td>217075@student.upm.edu.my</td><td>018-9569198</td></tr>
<tr><td>Chua Hui Qi</td><td>215091@student.upm.edu.my</td><td>011-23222350</td></tr>
<tr><td>Lok Yong Xue</td><td>215524@student.upm.edu.my</td><td>010-5463553</td></tr>
<tr><td>Tan Jia Qing</td><td>217067@student.upm.edu.my</td><td>011-55055288</td></tr>
</table>
</td></tr>
<tr><td>Problem scenario description</td><td>The scenario involves a logistics company responsible for delivering cargo products to customers using truck. The truck has a maximum capacity of 500 kilograms. Each piece of cargo has a specific weight. The decision on which shipments to load is based on a priority score, which combines revenue, urgency, and service level (e.g., premium vs. standard). The objective is to maximize the total priority value of the cargo loaded onto the truck without exceeding the weight limit.</td></tr>
<tr><td>Why it is important</td><td>Optimal cargo loading is important to maximize delivery value, reduce costs, and improve service by fully utilize transport capacity, leading to fewer trips, lower fuel usage, and better asset efficiency.
It also ensures faster, more accurate deliveries, boosting customer satisfaction and loyalty. Fewer trips mean lower emissions, supporting environmental goals. This smart use of technology strengthens DHL’s position as a market leader while increasing revenue by prioritizing high-value shipments.</td></tr>
<tr><td>Problem specification</td><td>Maximum Weight Capacity: 500 kg
Cargo Items: Various cargo with specific weights, revenue, urgency and service level.
Objective: Maximize the total priority value without exceeding the weight limit</td></tr>
<tr><td>Potential solutions</td><td>1. Sorting: Sort items based on value-to-weight ratio to select items.
2. Divide and Conquer (DAC): Break the problem into smaller sub-problems and combine their solutions.
3. Dynamic Programming (DP): Use a DP approach to find the optimal combination of items that maximize the priority value without exceeding the weight capacity.
4. Greedy Algorithm: Select items based on the highest priority value-to-weight ratio until the weight limit is reached.
5. Graph Algorithms: Model the problem as a graph and use algorithms to find the optimal path.</td></tr>
<tr><td>Sketch (framework, flow, interface)</td><td>Framework: Java-based application
Flow:
Read cargo items from csv file with weights, revenue, urgency, and service level
Calculate the priority score of each item based on the revenue, urgency and service level
Apply the selected algorithm to determine the optimal cargo load
Output the selected items and their total priority value
Interface: Simple command-line interface for input and output</td></tr>
</table>

Project Proposal Refinement

<table>
<tr><td>Group Name</td><td>Group 2</td></tr>
<tr><td>Members</td><td>
<table>
<tr><th>Name</th><th>Role</th></tr>
<tr><td>Tan Kah Jun</td><td>Scenario model development and documentation</td></tr>
<tr><td>Chua Hui Qi</td><td>Algorithm implementation and coding</td></tr>
<tr><td>Lok Yong Xue</td><td>Algorithm analysis</td></tr>
<tr><td>Tan Jia Qing</td><td>Algorithm design and specification</td></tr>
</table>
</td></tr>
<tr><td>Problem statement</td><td>Given a set of shipments, each with attributes such as revenue, urgency, service level, and weight, the goal is to select a subset of shipments that maximizes total shipment value (priority score), while not exceeding the truck's maximum allowable weight.</td></tr>
<tr><td>Objectives</td><td>Design an algorithm to maximize the priority value of the loaded cargo.
Implement the algorithm in Java.
Analyse the correctness and time complexity of the algorithm.
Develop an online portfolio to illustrate the project and its results.</td></tr>
<tr><td>Expected output</td><td>1. An optimized list of cargo items
2. Total priority value of the loaded cargo
3. An online portfolio detailing the problem, solution, and results</td></tr>
<tr><td>Problem scenario description</td><td>The scenario involves a logistics company responsible for delivering cargo using a cargo truck. The truck has a maximum weight capacity of 500 kilograms. Each piece of cargo has a specific weight and priority value. The decision on which shipments to load is based on a priority score, which combines revenue, urgency, and service level (e.g., premium vs. standard). The objective is to maximize the total priority value of the cargo loaded onto the truck without exceeding the weight limit.</td></tr>
<tr><td>Why it is important</td><td>
1. Operational Efficiency and Cost Reduction
DHL uses an algorithm to load cargo more efficiently, ensuring each trip carries the most valuable items. This reduces the number of trips, cuts fuel costs, and makes better use of vehicles.

2. Customer Satisfaction and Service Quality
Optimized loading means faster, more accurate deliveries. Customers get their items on time, which builds trust and improves DHL’s reputation.

3. Environmental Impact and Sustainability
Fewer trips and better fuel use mean lower emissions. This supports DHL’s green goals and meets environmental regulations.

4. Competitive Advantage and Market Leadership
Using smart algorithms gives DHL a tech edge over competitors, helping it lead the logistics industry and attract more clients.

5. Revenue Optimization
</td></tr>
</table>

	By filling trucks with the most valuable items, DHL earns more per trip and increases profits without raising prices.
Problem specification	**Maximum Weight Capacity:** 500 kg **Cargo Items:** Various cargo with specific weights and each with attributes such as revenue, urgency and service level. **Objective:** Maximize the total priority value without exceeding the weight limit
Potential solutions	1**. Sorting:** Sort items based on value-to-weight ratio to select items. 2. **Divide and Conquer (DAC):** Break the problem into smaller sub-problems and combine their solutions. 3. **Dynamic Programming (DP):** Use a DP approach to find the optimal combination of items that maximize the priority value without exceeding the weight capacity. 4. **Greedy Algorithm:** Select items based on the highest priority value-to-weight ratio until the weight limit is reached. 5**. Graph Algorithms:** Model the problem as a graph and use algorithms to find the optimal path.
Sketch (framework, flow, interface)	**Framework** The application will be implemented using Java. **Flow:** **1. Input Cargo Items** • User Input: The user provides the file path to a CSV containing shipment data. • Data Structure: Each shipment is read from the file and stored as a Shipment object with the following attributes: *id, revenue, timeRemaining, serviceLevel, weight* A priority score is calculated for each item using a weighted formula. **2. Algorithm Application** • Algorithm Used: A Dynamic Programming (DP) approach is applied to solve the 0/1 Knapsack Problem. It selects a combination of shipments that maximizes the total priority score without exceeding a 500 kg weight limit. • Processing Steps: o A DP table (dp) is filled to store the maximum score possible at each sub-capacity. o A keep matrix tracks which shipments are included in the optimal solution. o The algorithm backtracks to identify which shipments make up the optimal selection. **3. Output** • Selected Items: The program outputs a list of selected shipment IDs, along with their weights and priority scores. • Total Priority Value: The total priority score and total weight of selected shipments are displayed. • Feedback:

<table>
<tr><td></td><td>A full list of all shipment IDs and their individual priority scores is shown for transparency and verification.

Interface:
Simple Command-Line Interface (CLI)

1. Prompt for Input
a. The application prompts the user to enter the file path of a CSV file containing shipment data.
b. The CSV file must contain the following details for each cargo item (in order): ID, Revenue, Time Remaining, Service Level, Weight
c. The application reads and parses this data, automatically calculating the priority score for each item.

2. Algorithm Selection
a. The algorithm is predefined as Dynamic Programming (DP) to ensure optimal cargo selection under a 500 kg weight limit.
b. No manual algorithm selection is required from the user, simplifying the process for accurate and efficient results.

3. Display Output
a. The program displays a list of selected shipment IDs, including their individual weights and priority scores.
b. It also shows the total priority score and total weight of the selected shipments.
c. For transparency, the application displays a complete list of all shipment priority scores at the end.</td></tr>
<tr><td>Methodology</td><td>
<table>
<tr><td>Milestone</td><td>Time</td></tr>
<tr><td>Scenario refinement</td><td>wk10</td></tr>
<tr><td>Scenario model development and algorithm selection</td><td>wk11</td></tr>
<tr><td>Code implementation</td><td>wk12</td></tr>
<tr><td>Correctness and time complexity analysis</td><td>wk13</td></tr>
<tr><td>Online portfolio and presentation</td><td>wk14</td></tr>
</table>
</td></tr>
</table>

Project Progress (Week 10 – Week 14)

<table>
<tr><td>Milestone 1</td><td>Scenario refinement</td></tr>
<tr><td>Date (week)</td><td>5 June 2024 (Week 11)</td></tr>
<tr><td>Description/ sketch</td><td>Refine the problem scenario and initial problem statement.</td></tr>
<tr><td>Role</td><td>
<table>
<tr><td>Member 1</td><td>Member 2</td><td>Member 3</td><td>Member 4</td></tr>
<tr><td>Tan Kah Jun</td><td>Chua Hui Qi</td><td>Lok Yong Xue</td><td>Tan Jia Qing</td></tr>
<tr><td>Lead the scenario refinement.</td><td>Prepare initial problem statements.</td><td>Prepare initial problem statements.</td><td>Prepare problem sketches and objectives.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Milestone 2</td><td>Scenario model development and algorithm selection</td></tr>
<tr><td>Date (Wk)</td><td>8 June 2024 (Week 12)</td></tr>
<tr><td>Description/ sketch</td><td>Model the problem scenario and select suitable algorithm for the problem.</td></tr>
<tr><td>Role</td><td>
<table>
<tr><td>Member 1</td><td>Member 2</td><td>Member 3</td><td>Member 4</td></tr>
<tr><td>Tan Kah Jun</td><td>Chua Hui Qi</td><td>Lok Yong Xue</td><td>Tan Jia Qing</td></tr>
<tr><td>Model the problem scenario.</td><td>Research and select suitable algorithms.</td><td>Research and select suitable algorithms.</td><td>Research and select suitable algorithms.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Milestone 3</td><td>Code implementation</td></tr>
<tr><td>Date (Wk)</td><td>10 June 2024 (Week 12)</td></tr>
<tr><td>Description/ sketch</td><td>Select the algorithm and begin initial coding.</td></tr>
<tr><td>Role</td><td>
<table>
<tr><td>Member 1</td><td>Member 2</td><td>Member 3</td><td>Member 4</td></tr>
<tr><td>Tan Kah Jun</td><td>Chua Hui Qi</td><td>Lok Yong Xue</td><td>Tan Jia Qing</td></tr>
<tr><td>Assist with coding and document the methodology.</td><td>Begin coding the algorithm.</td><td>Assist with coding and document the methodology.</td><td>Assist with coding and document the methodology.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Milestone 4</td><td>Correctness and time complexity analysis</td></tr>
<tr><td>Date (Wk)</td><td>15 June 2024 (Week 13)</td></tr>
<tr><td>Description/ sketch</td><td>Analyze the correctness and time complexity of the algorithm.</td></tr>
<tr><td>Role</td><td>
<table>
<tr><td>Member 1</td><td>Member 2</td><td>Member 3</td><td>Member 4</td></tr>
<tr><td>Tan Kah Jun</td><td>Chua Hui Qi</td><td>Lok Yong Xue</td><td>Tan Jia Qing</td></tr>
<tr><td>Analysis correctness of the algorithm.</td><td>Compile and document analysis results.</td><td>Analyse time complexity.</td><td>Compile and document analysis results.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Milestone 5</td><td>Online portfolio and presentation</td></tr>
<tr><td>Date (Wk)</td><td>23 June 2024 (Week 14)</td></tr>
<tr><td>Description/ sketch</td><td>Prepare the online portfolio and presentation materials.</td></tr>
<tr><td>Role</td><td>
<table>
<tr><td>Member 1</td><td>Member 2</td><td>Member 3</td><td>Member 4</td></tr>
<tr><td>Tan Kah Jun</td><td>Chua Hui Qi</td><td>Lok Yong Xue</td><td>Tan Jia Qing</td></tr>
<tr><td>Create online portfolio.</td><td>Develop the presentation materials.</td><td>Develop the presentation materials.</td><td>Review and finalize the portfolio and presentation.</td></tr>
</table>
</td></tr>
</table>