**DATA STRUCTURES AND ALGORITHMS II — C950**

## TASK 1: WGUPS ROUTING PROGRAM PLANNING **A. Algorithm Identification**

I will use the Greedy Algorithm, specifically a Nearest Neighbor approach, to build an efficient package delivery route that stays under 140 miles while delivering all 40 packages on time.

**B. Data Structure Identification**

To support my algorithm, I will use a self-resizing hash table (custom-built in Python) that stores all package objects with their ID as the key.

**B1. Explanation of Data Structure**

The hash table stores package objects where each package ID is mapped to the complete package info (address, deadline, delivery status, etc.). This structure allows for constant-time access (O(1)) and updates to delivery status, which keeps all data components tightly linked and accessible.

**C1. Algorithm Logic (Pseudocode)**

For each truck:  
 SET the current location to HUB  
 WHILE the truck has packages

FIND the closest delivery location from the current location

ADD that location to the truck route

UPDATE mileage and delivery time

MARK all packages at that location as delivered

SET the current location to that location

## **C2. Development Environment**

I will use **PyCharm 2025.1 (Build #PY-251.23774.444)** on a **MacBook Pro (M4 Pro, 48GB RAM)** running **macOS 15.4.1**.

The environment runs **Python 3.12+** on an **OpenJDK 64-bit Server VM (JetBrains s.r.o)** with Metal Rendering enabled.

This setup ensures smooth execution and testing of my WGUPS routing application with efficient memory and performance handling.

## **C3. Space-Time Complexity Using Big-O Notation**

* **Routing algorithm (Nearest Neighbor):**
  + Time: O(n^2) (for each package, compare with remaining ones)
  + Space: O(n) (storing routes and distances)
* **Hash table for packages:**
  + Time: O(1) access and updates
  + Space: O(n) for n packages
* **User interface (lookup by ID, status):**
  + Time: O(1) per query
  + Space: O(n) (references package objects)
* **Whole program:**
  + Time: O(n^2)
  + Space: O(n)

## **C4. Scalability and Adaptability**

The current program is designed to scale efficiently as the number of packages increases. The hash table allows for constant-time (O(1)) lookups and insertions of package data, even as the number of packages grows into the hundreds or thousands.

The nearest neighbor algorithm also remains adaptable. However, since it is O(n²) in complexity, it could be replaced with a more scalable algorithm like Dijkstra’s if performance drops at enormous scales. Suppose new package requirements appear (like multi-city routes or hundreds of deliveries). In that case, the system can be expanded by updating the truck loading logic, using more trucks, and upgrading the routing algorithm.

## **C5. Software Efficiency and Maintainability**

The software is efficient because it separates concerns clearly: the hash table manages package data storage, the truck class manages delivery operations, and the routing logic is handled in a separate function.

This modular design allows parts to be updated or optimized independently without rewriting the entire system. For example, switching the nearest neighbor algorithm to Dijkstra’s would only require updating one function. Maintaining the project is easy because the code is simple, well-commented, and organized into logical sections following best practices for readability and reuse.

## **C6. Self-Adjusting Data Structure (Hash Table)**

**Strengths:**

The hash table offers constant-time (O(1)) lookup and insert operations under normal conditions, which makes accessing and updating package information very fast. It allows the delivery system to scale without degrading lookup performance as the number of packages grows.

**Weaknesses:**

If too many keys map to the same bucket (a collision), performance can drop to O(n) for that bucket. Additionally, the current simple chaining hash table does not automatically resize itself, so if the number of packages grows significantly beyond the original capacity, it would require manually resizing and rehashing to maintain performance.

## **C7. Key Justification**

For efficient delivery management, I chose delivery deadline as the key during the initial sorting and manual truck loading phase, because:

* It helps prioritize time-sensitive deliveries (e.g., EOD vs. 10:30 AM packages).
* It ensures all packages are grouped and routed logically around time constraints.
* It allows easy filtering for same-deadline deliveries to reduce mileage and speed up delivery.

However, once packages are in the hash table, the package ID is used as the hash key because:

* It is unique and constant.
* It supports O(1) lookup for tracking, status updates, and UI filtering.
* It’s ideal for real-time queries like “Where is package 13?” or “When was it delivered?”

This combo (deadline for loading, package ID for tracking) balances priority routing with efficient delivery management.