**DATA STRUCTURES AND ALGORITHMS II — C950**

TASK 1: WGUPS ROUTING PROGRAM implementation

**Task D - Intuitive Interface  
  
Task D1 - Screenshots to show the status of all packages loaded onto each truck at a time between 8:35 a.m. and 9:25 a.m.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Task D2 - Screenshot to show the status of all packages loaded onto each truck between 9:35 a.m. and 10:25 a.m.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Task D3 - Screenshot to show the status of all packages loaded onto each truck between 12:03 p.m. and 1:12 p.m.**

A screenshot of a computer

AI-generated content may be incorrect.

**Task E - Screenshot of Successful Completion of the Code That Includes the Total Mileage Traveled by All Trucks**

A screenshot of a computer program

AI-generated content may be incorrect.

**Task F - Justifying the Algorithm**

**F1. Two strengths of the algorithm used**

* **Simple and Fast**: The nearest neighbor algorithm always picks the closest address, so it's easy to implement and runs fast, even with many packages.
* **Works Well in Real Life**: It creates a short delivery route that stays under the 140-mile limit and delivers everything on time.

**F2. How does it meet all the requirements**

1. **Checkpoint Statuses (Screenshots D1–D3)**
   * **09:00 AM**: All packages with 9:00 AM deadlines (e.g. #15) are either **En route** or **Delivered**, and none remain at the hub past their deadline.
   * **10:30 AM**: All 10:30 AM deadline packages (e.g. #1, #6, #14, #16, #20, #25, #30, #34, #40) have status **Delivered at …** on or before 10:30 AM.
   * **12:30 PM**: Every package is now marked **Delivered at …**, showing 100% completion.
2. **Mileage Constraint**
   * Across all three runs the combined mileage is **91.3 miles**, well under the 140-mile cap.
3. **Dynamic Address Correction**
   * Package 9’s wrong address is deliberately used until **10:20 AM**, then automatically corrected in both the routing pass and the status checker, matching the specification.
4. **Performance & Priority**
   * Each run completes instantaneously for 40 packages.
   * Early-deadline packages naturally get picked first by proximity, so deadlines are met without explicit sorting.

**F3. Two other algorithms that would work**

* **Dijkstra’s Algorithm**
  + Finds the shortest path between two points.
  + Best for complex networks with variable edge weights.
* **A (A-Star) Search Algorithm**
  + Like Dijkstra, it adds heuristics for faster decisions.
  + Suitable for optimizing routes with distance and time in mind.

**F3a. How are they different?**

* The Nearest Neighbor algorithm selects the closest unvisited location at each step without considering the future path. It can result in a longer overall route because it doesn’t plan.
* Dijkstra’s Algorithm calculates the shortest overall path from the start to each destination by considering all possible paths at once. It ensures the shortest global path but is slower and uses more memory because it examines every possibility.
* A\* Search Algorithm improves on Dijkstra by adding a heuristic (like straight-line distance to the goal) to make more intelligent guesses about which path is better. A\* often finds faster paths than Dijkstra, especially when the heuristic is well-chosen. Like Dijkstra, it’s more complex and resource-intensive than Nearest Neighbor.

**Task G - What I’d Do Differently**

If I did this again, I’d:

* Add automatic truck loading by deadline priority.
* Build a more detailed UI with a main menu and CSV export.
* Try Dijkstra or A\* to compare routing results and gain more experience.

**Task H - Verifying the Data Structure**

**H. How my hash table meets requirements**

* Stores all required package fields (ID, address, deadline, city, zip, weight, status).
* Lookups and updates are fast using the package ID as the key.
* Used consistently for all package tracking.

**H1. Two other data structures that could work**

* **Binary Search Tree (BST)**
  + Stores packages in sorted order.
  + Suitable for range queries, but slower lookup than a hash table.
* **Linked List**
  + Very simple to implement.
  + Slow search times (O(n)) but easy to insert/delete.

**H1a. Differences from my hash table**

* A Hash Table provides constant-time (O(1)) access to packages by ID, which is ideal when we need to frequently look up, update, or check the status of a package quickly. It does not maintain any specific order of the elements.
* A Binary Search Tree (BST) organizes packages in a sorted structure, allowing for efficient range queries or finding minimum/maximum values. However, basic lookups are slower (O(log n)) than hash tables and even slower (O(n)) if the tree is unbalanced.
* A Linked List stores elements linearly. It is efficient and straightforward for inserting and deleting elements, but very slow for searching (O(n)) because it requires scanning each node individually. It does not support fast lookup by key like a hash table does.