## **A. Algorithm Identification**

The algorithm I will use is the Greedy Nearest Neighbor Algorithm. It begins at the hub and repeatedly selects the closest unvisited delivery location until all packages are delivered. This approach adjusts based on the truck's current location and remaining deliveries, which makes it a self-adjusting routing solution. It meets the requirement to deliver all packages while minimizing total distance traveled, staying under the 140-mile limit.

## **B. Data Structure Identification**

The self-adjusting data structure I used is a hash table, implemented using a Python dictionary. It dynamically allocates space and handles growth automatically, making it ideal for tracking real-time delivery data. I use it to store all package details with constant-time access and updates.

### **B1. Explanation of Hash Table Structure**

I organized the hash table by using the package ID as the key. Each key maps to a “Package” object that holds the delivery address, deadline, city, zip, weight, and delivery status. Python handles hash collisions internally using open addressing with probing. If two package IDs hash to the same location, the interpreter probes for the next open slot automatically, making the structure efficient and self-adjusting without requiring manual chaining or bucket management.

### **C1. Algorithm Pseudocode**

Start at the hub at 8:00 AM

While truck has undelivered packages:

For each remaining package:

Calculate distance from current location

Skip if package has future constraint (e.g., delayed address fix)

Select package with nearest valid address

Drive to address

Update current time based on travel distance

Mark package as delivered, store delivery time

If packages remain at hub:

Return to hub

Load next set of packages

### **C2. Programming Environment**

* **Language:** Python 3.10
* **IDE:** PyCharm
* **OS:** Windows 11
* **Computer:** Lenovo ThinkPad, Intel i5, 16 GB RAM

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Segment** | **Function** | **Time Complexity** | **Space Complexity** |
| Package Class | \_\_init\_\_ | O(1) | O(1) |
| Hash Table Access | Insert + Lookup | O(1) | O(n) |
| Distance Matrix | Load and Access | O(n²) | O(n²) |
| Routing Algorithm | Nearest Neighbor | O(n²) | O(n) |
| Address Correction | Manual Update | O(1) | O(1) |
| **Overall Program** | **Entire System (composite)** | **O(n²)** | **O(n²)** |

### **C4. Scalability and Adaptability**

The solution scales well for moderate increases in package volume due to the hash table’s dynamic resizing and the algorithm’s self-adjusting logic. However, since the nearest neighbor algorithm has O(n²) complexity, it may need to be replaced with a more efficient algorithm, such as 2-opt, if the package count increases significantly.

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

The program is efficient because it uses O(n²) time complexity, which is acceptable for the dataset size. It is also maintainable due to its modular design, clear class structures, and use of standard data structures. Adding features like a GUI or advanced routing logic can be done with minimal refactoring.

### **C6. Strengths and Weaknesses of Hash Table**

**Strengths:**

* O(1) access and updates for package data
* Automatically resizes and handles growth

**Weaknesses:**

* No inherent ordering as it requires sorting for deadlines or locations
* Poor key choice can cause clustering or collisions

### **C7. Key Justification**

I used “package ID” as the hash table key. It is unique, constant, and never changes, making it the most efficient and reliable choice. Using address, deadline, or delivery status would introduce ambiguity and slow down lookups.

### **D. Sources**

* Cormen, Thomas H., et al. *Introduction to Algorithms*. 3rd ed., MIT Press, 2009, <https://enos.itcollege.ee/~japoia/algorithms/GT/Introduction_to_algorithms-3rd%20Edition.pdf>.
* Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009.