Tyler Meester

Student ID: 009476045

C950

**Western Governor's University Parcel Service**

**Optimal Package Routing Application**

**A.  Identify a named self-adjusting algorithm (e.g., “Nearest Neighbor algorithm,” “Greedy algorithm”) that you used to create your program to deliver the packages.**

The self-adjusting algorithm that I implemented in this program was a Nearest Neighbor algorithm. This means that at every iteration the package with the closest delivery address will be delivered next. It is self-adjusting because regardless of the size of the input, the function will function properly. This approach is beneficial as it optimizes the delivery route based on the real-time state of the system (the truck's current position and the remaining packages to be delivered), reducing the total distance traveled and the time required to deliver all packages.

**B.  Write an overview of your program, in which you do the following:**

**1.  Explain the algorithm’s logic using pseudocode.**

*initialize three trucks with respective packages and departure times*

*load the package data into the package hash map*

*function sortAndDeliver(truck):*

*create an undeliveredPackages list and fill it using the package IDs in the truck*

*while undeliveredPackages is not empty:*

*set closestDistance to infinity*

*set closestPackage to None*

*for each package in undeliveredPackages:*

*calculate the distance from the truck to the package (currentDistance)*

*if currentDistance < closestDistance:*

*update closestDistance to currentDistance*

*update closestPackage to current package*

*add closestPackage to the truck's delivery list*

*remove closestPackage from undeliveredPackages*

*update truck's mileage and time*

*update package's delivery time and departure time*

*sortAndDeliver for each truck in the necessary order*

**2.  Describe the programming environment you used to create the Python application.**

This Python application was developed using Python 3.9.6 in VS Code on a 2019 Apple MacBook Pro running Ventura 13.4.

**3.  Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.**

Time Complexity:

* Loading the CSV data and packages into the hash map has a time complexity of O(n), where n is the number of packages.
* The sortAndDeliver() function, where the nearest neighbor algorithm is located, has a worst-case time complexity of O(n²) because for each package we're looking at all other undelivered packages to find the nearest one.
* The lookup operation in the hash map has an average-case time complexity of O(1), which allows efficient access to package details.
* Overall, the program's time complexity is O(n²), dominated by the sortAndDeliver() function.

Space Complexity:

* The hash map storing package information contributes O(n) space complexity, where n is the number of packages.
* The remaining variables and data structures used in the program contribute constant space, O(1).
* The overall space complexity of the program is O(n).

**4.  Explain the capability of your solution to scale and adapt to a growing number of packages.**

The application can adapt to handle a larger number of packages with minor modifications. The hash map used to store packages scales well with increased data, maintaining an average lookup time of O(1). However, the primary limitation of the current solution is the O(n²) time complexity of the sortAndDeliver() function, which could slow down the program if the number of packages increases substantially. Furthermore, adding a lot of packages could require more manual input to decide which packages to put in which trucks at what time. Future iterations of this project might try to automate that process.

**5.  Discuss why the software is efficient and easy to maintain.**

The software's efficiency stems from several factors:

* Algorithmic Efficiency: The primary algorithm used, Nearest Neighbor, has a time complexity of O(n²), making it relatively efficient for solving the problem at hand.
* Data Structure: The use of a hash table for package storage ensures fast retrieval and insertion operations, usually in constant time O(1), enhancing the overall performance of the software.
* Code Organization: The program is divided into modular functions, each performing a specific task. This makes the code easier to understand, debug, and improve.

The maintainability of the software is also a strength:

* Modular Design: The use of object-oriented programming principles allows for easy modifications and additions to the code. Each class and function has a clear, defined role, making it easy to isolate and adjust sections of the code as necessary.
* Code Comments and Documentation: The codebase is well-documented, with comments explaining the purpose and functionality of each section. This will assist any developer in understanding the codebase quickly, making maintenance easier.

**6.  Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).**

Strengths

* Fast Access: Hash tables provide constant-time average complexity for search, insert, delete operations. This makes them extremely efficient for tasks that require frequent access and updates, such as in this application.
* Key-Value Pairs: Hash tables store data in key-value pairs, providing an intuitive way to store data that naturally fits into a two-item structure.

Weaknesses:

* Collisions: If two keys hash to the same index, a collision occurs. Handling these collisions requires additional computing, which can slow down operations.
* Space Inefficiency: Hash tables need to be oversized to reduce the likelihood of collisions, which can lead to inefficient use of memory.
* Poor Ordering: Hash tables don't maintain any specific order of entries. This can be a problem for tasks that require sorting or the retrieval of keys in a specific order.

**C.  Write an original program to deliver all the packages, meeting all requirements, using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and the “WGUPS Package File.”**

See attached program code.

**D.  Identify a self-adjusting data structure, such as a hash table, that can be used with the algorithm identified in part A to store the package data.**

**1.  Explain how your data structure accounts for the relationship between the data points you are storing.**

The Hash Table, which is the self-adjusting data structure utilized in this project, effectively manages the relationships between various data points. Each package is represented as a unique entry in the Hash Table, with the package ID acting as the key and the package details (such as delivery address, delivery deadline, weight, etc.) as the corresponding value(s). This key-value pairing is fundamental to understanding the relationship between data points. This information is further abstracted by the creation of a Package class, which effectively organizes and simplifies the data within the Hash Table.

**E.  Develop a hash table, without using any additional libraries or classes, that has an insertion function that takes the following components as input and inserts the components into the hash table:**

**•   package ID number**

**•   delivery address**

**•   delivery deadline**

**•   delivery city**

**•   delivery zip code**

**•   package weight**

**•   delivery status (e.g., delivered, en route)**

See attached program code.

**F.  Develop a look-up function that takes the following components as input and returns the corresponding data elements:**

**•   package ID number**

**•   delivery address**

**•   delivery deadline**

**•   delivery city**

**•   delivery zip code**

**•   package weight**

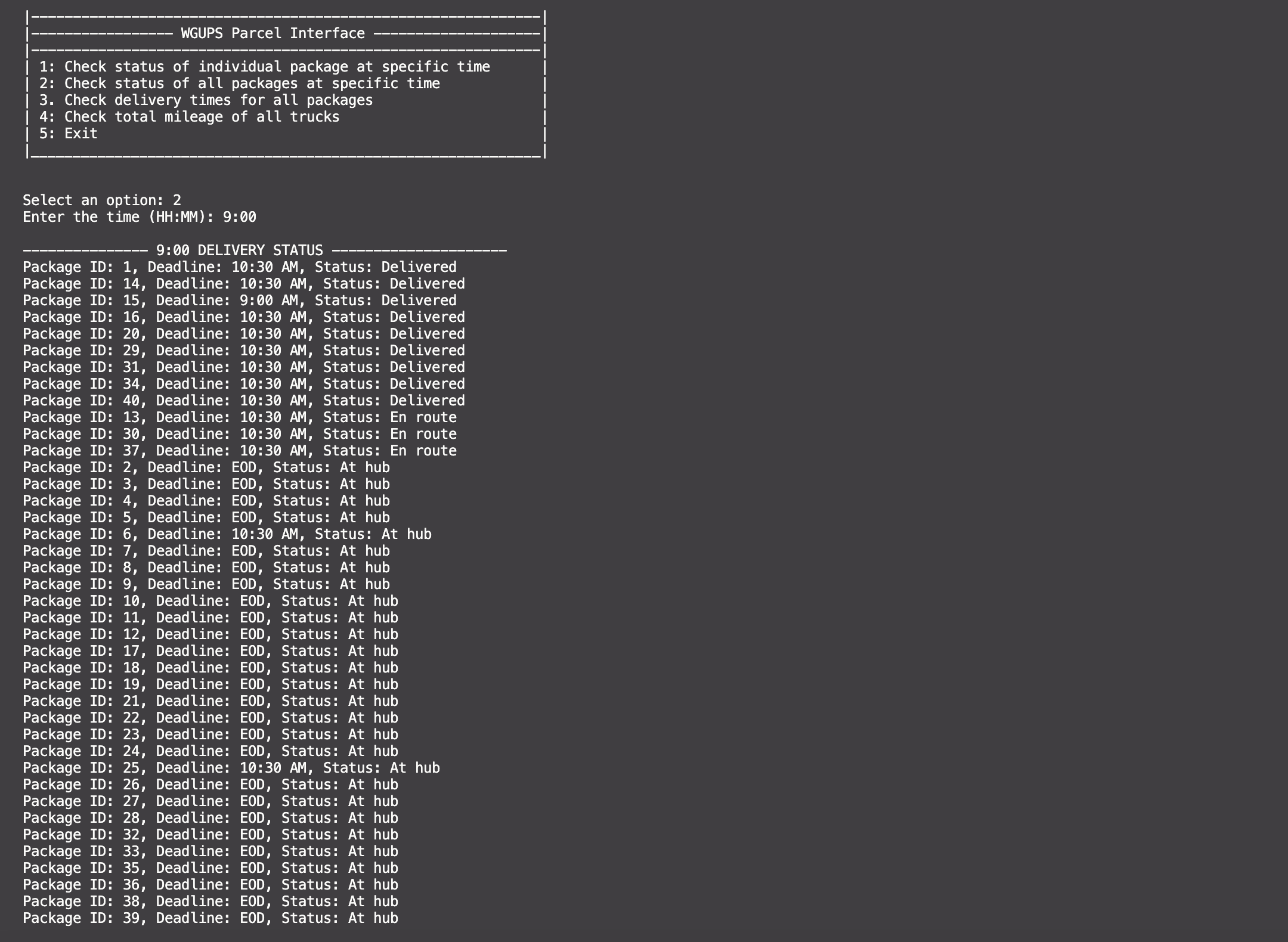
**•   delivery status (i.e., “at the hub,” “en route,” or “delivered”), including the delivery time**

See attached program code.

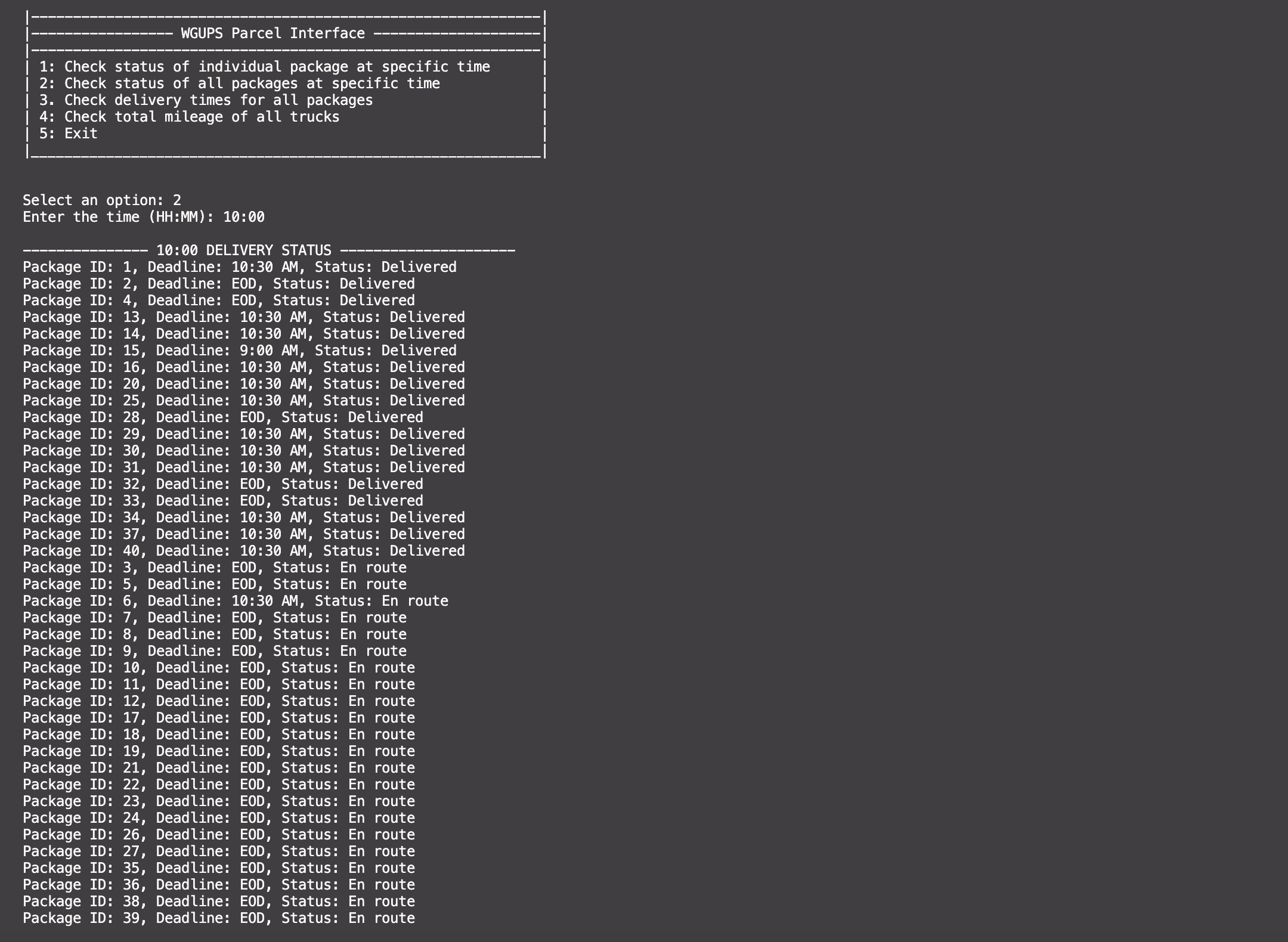
**G.  Provide an interface for the user to view the status and info (as listed in part F) of any package at any time, and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)**

See attached program code.

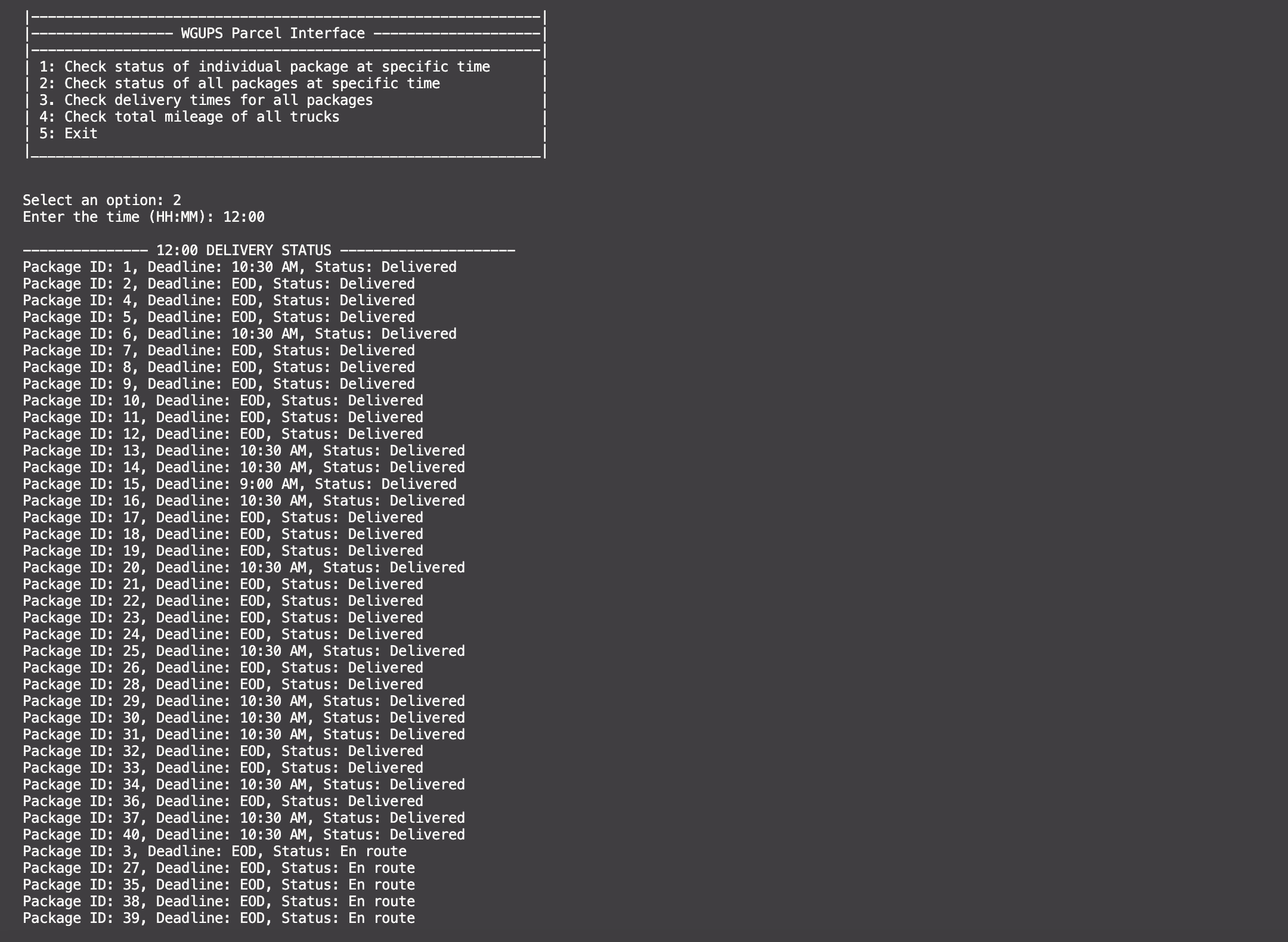
**1.  Provide screenshots to show the status of all packages at a time between 8:35 a.m. and 9:25 a.m.**

****

**2.  Provide screenshots to show the status of all packages at a time between 9:35 a.m. and 10:25 a.m.**

****

**3.  Provide screenshots to show the status of all packages at a time between 12:03 p.m. and 1:12 p.m.**

****

**H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by all trucks.**

**Text

Description automatically generated**

**I.  Justify the core algorithm you identified in part A and used in the solution by doing the following:**

**1.  Describe at least two strengths of the algorithm used in the solution.**

* Efficiency: One of the strengths of this algorithm is its efficiency. It works by always selecting the closest undelivered package for the next delivery, which results in a relatively efficient route. This approach reduces the overall travel distance and hence the time required for all packages to be delivered.
* Simplicity: The Nearest Neighbor Algorithm is straightforward to implement. It doesn't require complex data structures or extensive computation, and it's easy to understand. The simplicity of the implementation aids in debugging and verification of the algorithm's correctness.

**2.  Verify that the algorithm used in the solution meets all requirements in the scenario.**

The total combined miles traveled by all trucks can be verified by selecting option 4 in the user interface ("Check total mileage of all trucks"). The output is the mileage traveled by each truck and the total mileage traveled by all trucks, which is confirmed to be less than 140 miles.

All packages were delivered on time, and the delivery times can be verified by choosing option 3 ("Check delivery times for all packages"). The delivery time for each package is displayed, which can be cross-verified with the deadline times from the package file to confirm on-time delivery.

The user interface also allows you to check the status of a particular package or all packages at a specific time by choosing option 1 ("Check status of individual package at specific time") or option 2 ("Check status of all packages at specific time"). You can input any time and see the status of each package at that time, verifying that packages were delivered according to their delivery requirements.

**3.  Identify two other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.**

**a.  Describe how each algorithm identified in part I3 is different from the algorithm used in the solution.**

1. Brute Force Algorithm: The brute force algorithm would involve calculating the distance for every possible route and then selecting the route with the shortest total distance. The main difference from the nearest-neighbor algorithm is that the brute force algorithm does not start by selecting the closest undelivered package. Instead, it generates all possible routes and then picks the one with the shortest distance. This algorithm is guaranteed to find the shortest path, unlike the nearest-neighbor algorithm. However, it is computationally expensive, especially for large numbers of packages, as it involves calculating the distance for each permutation of the delivery route.
2. 2-opt Algorithm: This algorithm refines initial solutions by iteratively rearranging the tour path for a shorter total distance. It's different from the nearest-neighbor algorithm, which selects the next node purely based on proximity. The 2-opt algorithm instead rearranges paths for global optimization, providing a better balance between solution quality and computational effort.

**J.  Describe what you would do differently, other than the two algorithms identified in I3, if you did this project again.**

If I were to do this project again I would want to figure out a way to optimize the process of selecting which packages go into each truck. There was a degree of manual placement of packages in delivery trucks to ensure that all of the delivery constraints were met. In a real world application this would require too much effort on a regular basis, and would likely need to be automated.

**K.  Justify the data structure you identified in part D by doing the following:**

**1.  Verify that the data structure used in the solution meets all requirements in the scenario.**

**a.  Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.**

As the number of packages to be delivered increases, the time needed to complete the look-up function in a hash map generally stays constant. This is because hash maps use hashing functions to map keys to specific buckets in an array, providing nearly constant time search, insert, and delete operations.

**b.  Explain how the data structure space usage is affected by changes in the number of packages to be delivered.**

The space usage of a hash map increases linearly with the number of packages to be delivered. Each additional package (and its associated data) would require one new key-value pair in the map.

**c.  Describe how changes to the number of trucks or the number of cities would affect the look-up time and the space usage of the data structure.**

Changes to the number of trucks or cities would not significantly affect the lookup time of the hash map. This is because the lookup time of a hash map is, in an ideal situation, O(1) constant time, which means it's independent of the number of elements in the hash map. The lookup time is determined by the hash function, which takes the key and calculates the index of an array where the value can be found.

**2.  Identify two other data structures that could meet the same requirements in the scenario.**

**a.  Describe how each data structure identified in part K2 is different from the data structure used in the solution.**

Binary Search Tree (BST): Unlike the Hash Map that uses a hashing function to store and retrieve items, the BST stores items in a sorted manner within a tree-like structure. Each node has a key, and each left child is less than the parent node, while the right child is more. This allows for ordered operations like "minimum", "maximum", or "next largest" which cannot be performed as efficiently in a hash map. BST could provide a balance between query and update performance and handle larger datasets than could fit in memory.

Array: Arrays could store the package data, with the index as the package ID and the value as the package details. Arrays provide constant time access for reading values if the index is known. However, arrays are not dynamic, so they might not be the best choice if the number of packages can vary greatly. Also, they do not efficiently handle operations such as insertion or deletion in the middle.