*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 Nearest Neighbor Algorithm is used.

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

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

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

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

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

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

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

1. Pseudocode:

* + 1. not\_delivered = package\_hash\_table.lookup(packageID) for each packageID in truck.packages
    2. truck.packages.clear()
    3. while not\_delivered is not empty: 4. next\_package = None
    4. next\_address = 1000
    5. for each package in not\_delivered: 7. distance = distance\_in\_between(extract\_address(truck.address), extract\_address(package.address))
    6. if distance <= next\_address: 9. next\_address = distance
    7. next\_package = package
    8. truck.packages.append(next\_package.ID)
    9. truck.mileage += next\_address
    10. truck.address = next\_package.address
    11. truck.time += next\_address / 18
    12. next\_package.delivery\_time = truck.time
    13. next\_package.departure\_time = truck.depart\_time
    14. not\_delivered.remove(next\_package)

2. I utilized PyCharm Community Edition in the creation of this Python application.

3. Space-time complexity is identified in big O notation within the program’s comments.

4. My solution can scale decently from small to medium size amounts of packages. However, due to the time complexity of the sorting algorithm being quadratic time, it will begin to increasingly slow down as the load of packages increases.

5. By utilizing classes, I am able to modularize my program, reducing the need for repeat code as well as streamlining testing and pinpointing program errors. My program is also able to manually place packages upon trucks, which enables packages with specific requirements to be accommodated easily.

6. A hash table is a self-adjusting data structure that is excellent for storing and retrieving data quickly. One of its biggest strengths is the constant-time average-case performance for basic operations, such as inserting, deleting and looking up an element. Some of a hash table’s weaknesses include not being suitable for operations that require order (such as sorting), suffering from key collision and using large amounts of memory.

*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.*

1. A hash table can be utilized with the nearest neighbor algorithm by having the hash table store the data points alongside their corresponding keys, then using the hash table to look up the nearest neighbor of a given query point. A hash function maps each data point to a unique key.

*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.*

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

*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. Some of the major strengths of using the nearest neighbor algorithm include:

Simplicity: The nearest neighbor algorithm is a simple and easy-to-implement algorithm that can be easily understood and applied to a wide range of problems.

Efficiency: The nearest neighbor algorithm is an efficient algorithm that can be used to quickly find the closest point in a large dataset. The time complexity of the algorithm is typically O(n) in the worst case, and O(log n) if the dataset is organized in a data structure like a k-d tree.

Flexibility: The nearest neighbor algorithm can be used in different ways for different types of data and applications. For example, it can be used for classification, regression, density estimation, clustering, and more.

2. This algorithm is self-adjusting, delivers packages adhering to their requirements, as well as keeps the combined distance traveled between all three trucks below 140 miles.

3. Two other algorithms that would have met the requirements of the scenario are Dijkstra's shortest path algorithm and Bellman-Ford algorithm. As opposed to the nearest neighbor algorithm, Dijkstra’s shortest path is a single-source shortest path algorithm, which means it finds the shortest path from one specific starting node to all others in the graph. Bellman-Ford’s algorithm differs from the nearest neighbor algorithm by utilizing a weighted directed graph, which is used to update distances between points.

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

If I did this project again, I would implement a more sophisticated GUI that would allow the user to graphically see the path that the trucks took to deliver their packages. I would also implement a method to automatically sort packages onto trucks depending on their requirements.

*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.*

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

*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.*

*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.*

1. The hash table I used within the solution contains an insertion function, lookup function and is able to adjust to different amounts of packages. Because I am using a hash table, as the number of packages increases, the length time needed to complete the look-up function increases by n. Given a large number of packages, this can become unwieldy. As more packages are added the space required will also increase to an unsuitable amount. As long as new addresses are provided alongside distances to all other addresses, there will be no issue there. However, as the size of the packages that need to be delivered grows, the program will begin to suffer more and more in terms of lookup time and space usage.

2. Two other data structures that would have met requirements for the scenario would be balanced trees and graphs. A balanced tree differs from a hash table in that it stores data in a tree-like format. Due to this, balanced trees of a logarithmic time complexity for basic operations, such as insertions, deletions, and lookups. Graphs differ from hash tables in that they use vertices and edges, which connect the vertices.