**Brandon Baggatta**

**CS950 PA**

**A**. The algorithm used in this program to deliver the packages is a variation of the "Nearest Neighbor" algorithm.

**B. Overview of the program:**

1.Algorithm’s logic in pseudocode:

FOR each truck

WHILE packages to deliver remain

SET next\_distance to a large value

SET next\_package to None

FOR each package in pending\_packages

CALCULATE current\_distance from truck to package

IF current\_distance < next\_distance

SET next\_distance to current\_distance

SET next\_package to this package

UPDATE truck status with new package, location, and distance

UPDATE package delivery status

**2**.The programming environment used to create the Python application was Python 3.8, inside of Visual Studio Code for writing the code and the integrated terminal for executing the scripts. No additional libraries beyond Python's built-in libraries were used.

**3**.The space complexity of the program is primarily determined by the number of packages, which we'll call n. The major space-consuming part is the creation of the hash table for storing package data, which takes O(n) space.

The time complexity of the program is primarily driven by the delivery algorithm. In the worst-case scenario, every package could be at the maximum distance from each other, forcing the truck to traverse all of them, making the complexity O(n^2).

**4**.The solution can adapt to a growing number of packages as it uses data structures (like lists and hash tables) that can expand dynamically. However, as the number of packages increases, the time taken by the nearest neighbor algorithm may increase significantly due to its quadratic time complexity.

**5**.The software is efficient because it makes good use of Python's built-in data structures and it avoids unnecessary computation by using a hash table for package lookups. It's easy to maintain because it has a modular design with well-defined classes and functions, each with a single responsibility.

**6**.The primary data structure used is a hash table, which has several strengths and weaknesses:

Strengths:

* Fast access: Hash tables provide constant time complexity for access, insertion, and removal of data, making it very efficient.
* Scalability: Hash tables can be resized to adapt to the volume of data, making them good for a scalable solution.

Weaknesses:

* Collisions: In cases where different keys produce the same hash, there's a need to handle collisions, which can impact performance.
* Space inefficiency: In order to maintain the efficiency of a hash table, there needs to be a balance between the number of entries (load factor) and the size of the table. If the size is much larger than the number of entries, it can lead to wasted space.
* Lack of order: Hash tables do not maintain any order of elements, making it unsuitable for operations that require sorted data.

As for the self-adjusting nature of the algorithm, the delivery process automatically adjusts to the closest package destination, which is a strength in reducing unnecessary travel. However, it's worth noting that this can lead to suboptimal solutions in some cases (the Nearest Neighbor problem), as it may not result in the shortest possible route for all deliveries.

**D.**The program utilizes a hash table data structure to store the package data. The hash table stores key-value pairs, where the key is the package ID and the value is the package object itself. In this program, the package ID serves as the primary identifier or key that uniquely identifies each package. The corresponding value is an instance of the Package class, which contains all the relevant information about the package: delivery address, city, state, zip code, deadline, weight, and delivery status.  
The hash table data structure is ideal for this kind of relationship because it allows for quick and efficient access to the package information when given a package ID. It essentially creates a mapping from package IDs to package information.  
This design is particularly effective when we want to update or retrieve the status of a specific package. We can use the package ID to hash directly into the table and fetch the relevant Package object in constant time. This efficiency becomes increasingly significant as the number of packages grows, making the hash table an excellent choice for this use case.

**I. Justify the core algorithm:**

Two strengths of the Nearest Neighbor algorithm are:

* + Simplicity: It's an easy-to-understand and implement algorithm.
  + Efficiency: It offers a quick and practical solution for small to moderate data sets.

The Nearest Neighbor algorithm meets all the requirements mentioned in the scenario. It schedules the delivery of the packages by always choosing the nearest destination, thereby minimizing travel time and ensuring packages are delivered as quickly as possible.

* Two other algorithms that could be used are:
  + Dijkstra's Algorithm: This algorithm can determine the shortest path from one node (the starting point) to all other nodes in a graph. In this scenario, each delivery location could be a node, and the distances between them could be the edges.
  + Ant Colony Optimization (ACO): This is a probabilistic technique that could be useful in determining good paths through graphs, such as our delivery points.
* a. Dijkstra's Algorithm differs from the Nearest Neighbor algorithm as it finds the shortest path to all nodes from the source, not just the nearest one. It could potentially offer better overall routes for the delivery trucks. Ant Colony Optimization, on the other hand, simulates the behavior of ants and uses randomness and a feedback mechanism to find an optimal path. This could potentially find even better solutions than the deterministic Dijkstra and Nearest Neighbor approaches.

**J.**

If I were to do this project again, I would consider creating a more efficient way to load the packages onto the truck. Additionally, I would consider implementing a more interactive user interface, potentially with a graphical display of the trucks' routes.

**K**.

**1**. The hash table data structure meets all requirements in the scenario. It stores package information and allows for quick look-ups, insertions, and deletions.

**a**. The time needed to complete the look-up function in a hash table remains constant, regardless of the number of packages. This is because hash tables provide O(1) average time complexity for search operations.

**b**. The space usage of a hash table grows linearly with the number of packages, as each package will require its own entry in the table.  
c. Changes to the number of trucks or the number of cities would not directly affect the lookup time or the space usage of the hash table, as it is dependent on the number of packages. However, more trucks or cities might indirectly increase the number of packages, and thus the size of the hash table.

**2.** Two other data structures that could meet the requirements are:

* + Balanced search trees (e.g., AVL or Red-Black trees): These data structures maintain a sorted order of elements, allowing for efficient search, insert, and delete operations.
  + Trie: If package IDs were strings, a trie could be used to store them for efficient retrieval.
* a. A balanced search tree differs from a hashtable in that it keeps elements in a sorted order and guarantees logarithmic time complexity for search, insert, and delete operations. However, it generally uses more space than a hash table. A trie, on the other hand, is a tree-like data structure that provides efficient retrieval of keys (in this case, package IDs). It can be faster than a hash table for certain types of keys but uses more space due to the storage of links between nodes.

L. Sources:

Included are the links I used to learn about the algorithms discussed in this write up.

* [Nearest Neighbor Algorithm](https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm)
* [Dijkstra's Algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
* [Ant Colony Optimization](https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms)