A.

Greedy Algorithm

B1.

* Start time and trucks, already manually added to trucks, are prompted.
* Every truck has the start point of the address for the HUB and compare distance to each address end.
* If address has lower distance than other, it is replaced as the next destination.
* The end address is then added as the start address and time to each package and mileage is added to each truck.

B2.

Python 3.8, JetBrains/pyCharmEdu2022.1, i7-8550U CPU @ 1.80GHz, x64-based processor

B3.

* Entire code complexity: O(N^3)
* distanceBetween complexity: O(1)
* minDistance complexity: O(N)
* truckDeliverPackage complexity: O(N)

B4.

The Hash Table could obtain more packages as the table is structured to chain keys to the bucket\_list. The algorithm can properly annotate the increased numbers of packages if the addresses are part of the addresses listed in the 'WGUPS Address Table.csv' file. Package can be either added to csv file or manually inserted, and the algorithm can adjust to the new packages if they are either added to a truck or a new truck is added to deliver a separate set of packages.

B5.

The entire program has a complexity of O(N^3) and the algorithm is separated into 3 portions. This makes it easy to read code and separated any changes need as well as add any packages to the Class TruckHoldingPackages if more packages need to be delivered.

B6.

Strengths - Data can be stored and combined with other keys, chaining hash table. Hash table size is large enough to limit the amount of chaining needed.

Weaknesses - If a key is chained, it becomes difficult to find another key within the bucket\_list. No additional buckets are added to increase the size of the table.

D.

Chaining Hash Table

D1.

The package ID is stored by the results from % (modulo) of the table length and deleted, searched, and inserted into a bucket that corresponds with the results. Any additional packages that will be chained with the other packages if there is more than one package in a bucket. The result of the modulo equation is the key, where to find the specific bucket, and the data points for the Package Class are stored as an item in the bucket and can be viewed by searching for the item’s key.

I1.

The algorithm accurately obtains the shortest path between addresses, at that time, and can obtain as many packages as possible if within the address table.

I2.

Total mileage is 99.4 miles, and all packages are delivered on time according to packages requirements. When using the interface, choose option 1 and choose a distant time, i.e., 18:00:00, and the packages will show all the delivery times.

I3.

Dijkstra's Shortest Path & Knapsack

I3a.

Dijkstra's Shortest and the Greedy Algorithm both look for the shortest path at that time, address. However, though both are good at optimizing, Dijkstra's Shortest path would have to be used with a graph for the addresses noted. Knapsack and Greedy Algorithm also look for the best possible option from the start, but Knapsack can be less optimal.

J.

Creating a graph would for the addresses and utilizing Dijkstra's Shortest Path with weighted and unweighted edges.

K1.

Total mileage is 99.4 miles, and all packages are delivered on time according to packages requirements. When using the interface, choose option 1 and choose a distant time, i.e., 18:00:00, and the packages will show all the delivery times. The hash table successfully pulls up all package data when executed.

K1a.

Adding more packages to the hash table will affect how the look up function since there would be more keys to search through in the buckets.

K1b.

Adding more packages will affect the structure since it would need store and track the location of the data.

K1c.

Adding additional addresses or trucks will not affect the structure of the table much since address are added to an array and are looked up specifically as a string and the truck will be initiated solely by what packages that truck holds.

K2.

Binary Search Tree and MaxHeap

K2a.

Binary Search Tree will organize the nodes from smallest to largest, however searching and retrieving the nodes requires to go through multiple nodes, at least the root node, to find the specific key needed. Both BST and the Hash Table use a key to insert, search, and delete data from the tree and hash table. Max Heap will have the largest node as the root, nodes organized largest to smallest, and inserting a node would require adding the node at the bottom and moving the node up if it is larger than the other nodes above. MaxHeap uses a value that is like the Hash Table’s key that is used to insert and delete data.

L.

No outside sources were used.