1. **Identify the algorithm that will be used to create a program to deliver the packages and meets all requirements specified in the scenario.**

The algorithm used is mainly a greedy algorithm since the option chosen is the optimal one at specific points of time. This is evident since each truck has a list of packages and it drives to the package that has the least distance to their current location. This algorithm is only mostly greedy since manually sorting some packages is done to make sure it fits the special delivery requirements. After some packages are manually sorted, however, the truck still automatically sorts them and goes to the package that is closest.

B.  **Write a core algorithm overview, using the sample given, in which you do the following:**

1.  **Comment using pseudocode to show the logic of the algorithm applied to this software solution.**

Custom data structures are created to store the packages named which is the Node class. This node class includes the package id, package location, and package distance the location. The hash table is used to pull the information of each packages and stored in the node objects. Before the trucks start delivering, an array of these package objects is created with the distance attribute set as the distance from the hub’s location to the package’s address. Furthermore, after every delivery, the distances of each node is recalculated to the distance from truck’s current location instead of the hub’s location to make sure the truck always goes to the closest package.

O(1)

arrayOfPackages = []

O(n) on average and O(n^2) at worst since it has to take each package’s information and then fill the hash table

hashMapOfAllPackages = {…}

O(1); creation of an object

PackageObjects = Node(“8”, “300 State St”, distanceFromHub)

Time complexity of O(n) as it iterates through the number of packages and O(1) for appending it to an array.

for eachNumber in range(n)

arrayOfPackages.append(str(hashMapOfAllPackages.get(eachNumber)))

O(n^2) since it is nested. O(n) for iterating through the array of packages and O(n) for finding the distance on the csv which is done by iterating through the columns and rows.

for eachPackage in arrayOfPackages

eachPackage.distance = distance(fromHub, toLocation)

O(n^2logn) since it is iterating through the array of packages, which is O(n), and sorting the entire array, which is O(nlogn). Since the two is nested, the complexity comes to O(n^2logn)

Before every delivery:

Sort(arrayOfPackages.distances)

Go to first item’s location on the list of arrayOfPackages

2.  **Apply programming models to the scenario.**

The csv file is read, and the information loaded into the hash table. The program then uses the hash table in order to create an array of packages with only the pertinent information. The truck class then uses the hash table and the array of packages to find the package’s delivery information. All of this is done locally on a MacBook environment using visual studio code as the ide and use of the terminal to repeatedly run trials of the program.

4.  **Discuss the ability of your solution to adapt to a changing market and to scalability.**

The program’s complexity is O(n^3) since it iterates through n items and sorts them after every delivery. No matter the increase in the number of packages, this complexity will always stay the same since n is based on packages that is on the csv file. The complexity may seem to be inefficient, however, there are algorithms that have exponential or factorial time complexities which will deeply hurt the scalability of the software but the one provided here is in cubic time and is a great improvement over them.

5.  **Discuss the efficiency and maintainability of the software.**

The program’s efficiency is O(n^3) since there are n items to be added to an array, sorted with nlogn efficiency, and repeated every time a package is delivered. The program is easy to maintain since all that is needed is for the information to be added to the csv file and it will be automatically added to the packages list. The only thing that is difficult to maintain is the special delivery requirements which might need to be manually sorted but most of the work is done with the provided algorithm used.

6. **Discuss the self-adjusting data structures chosen and their strengths and weaknesses based on the scenario.**

The data structure is self-adjusting since it recalculates the closest package every time it drops off a package so that the package with the lowest distance is always at the front of the array. I do this by calling a recalculate function on the array after every delivery.

D.  **Identify a data structure that can be used with your chosen algorithm to store the package data.**

1.  Explain how your data structure includes the relationship between the data points you are storing.

A hash table is used to hold all the packages information. This includes package id, address, zip, city, state, delivery status, delivery deadline, package pick up time, and delivery time. A hash table is really important since it allows you to store a lot of information in a key/value pair and lets you look it up with an average search complexity of O(1). The hash table is then used to fill up the package object with the necessary information such as location before being added to the array and being sorted.

**I.  Justify your choice of algorithm by doing the following:**

1.  Describe at least **two** strengths of the algorithm you chose.

A greedy algorithm tries to find the optimal solution at a specific time. This helps with delivering packages since it will deliver the package that it is closest to. Another strength is that this algorithm does not have a large time complexity since it is only finding the optimal solution at a given interval instead of looking for the optimal solution in all of the possibilities.

**2.**Verify that the algorithm you chose meets all the criteria and requirements given in the scenario.

It is shown in a printout within the program that all the packages are delivered in a timely manner even those with special delivery requirements. Furthermore, the algorithm helps to decrease the miles traveled as much as possible since it always delivers the package that is closest.

**3.**Identify **two** other algorithms that could be used and would have met the criteria and requirements given in the scenario.

An algorithm that could be used and would meet the criteria and requirements is a divide and conquer algorithm, where one solution could be that each package is divided by their zip code and delivered accordingly.

Another algorithm that would fit but may not be practical is a bruteforce algorithm, where the algorithm finds the absolute optimal route, however, this would be in factorial time complexity. A bruteforce algorithm would have to go through every possible route and then find the one with the lowest total distance.

1. Describe how each algorithm identified in part I3 is different from the algorithm you chose to use in the solution.

A divide and conquer is not used since the core of the greedy algorithm is not dividing the items into smaller group. A greedy algorithm is instead trying to find the optimal solution for just that one instance of time.

Also, it is not a bruteforce algorithm since it is not trying to find the lowest distance traveled of every possible route which would take a massive amount of time. For this reason, the greedy algorithm used is magnitudes faster than a bruteforce algorithm.

**J.  Describe what you would do differently if you did this project again.**

Combining the custom created data structure, Node, and the hash table would organize the code a lot better and make it easier to handle the project.

A different data structure could have been used altogether such as a weighted graph which could have helped the time complexity of the entire project.

**K.  Justify your choice of data structure by doing the following:**

1.  Verify that the data structure you chose meets all the criteria and requirements given in the scenario.

a.  Describe the efficiency of the data structure chosen.

Hash tables have an average search, add, delete time complexities of O(1) and, in the worst case, has a time complexity O(n).

1. Explain the expected overhead when linking to the next data item.

The time complexity of a hash table is O(1) on average and O(n) in the worst case.

The memory complexity of a hash table is O(n), where n is the number of packages.

The hash table is used to pull information to create the package node objects before being appended to an array. By doing this, the algorithm saves overhead time since it is iterating through the array instead of through the hash table when delivering packages. However, it is possible that they have similar overhead if the hash table has zero collisions, but an array would still be faster since it doesn’t have to create a hash value which takes up time.

Memory overhead for a hash table is a lot larger than if an array is used since the hash table might have extra buckets that are unused. Furthermore, a hash table has to consider that creating a hash from the key takes up space. While minimal, can add up as the number of key-value pair increases.

Hashing keys and having additional unused buckets are factors when using hash tables and adds time and memory overheads. Because of these factors, even if the hash table has zero collisions, iterating through it will have lower bandwidth than iterating through an array with the same data. However, since a hash function has O(1) time and space complexity, the overhead is negligible as the amount of data increases.

1. Describe the implications of when more package data is added to the system or other changes in scale occur.

If more packages are added to the hash table and the size attribute is not changed to reflect the increased size of the data, then there could be more hash collisions when using the hash table to add, remove, or search and this could make the time complexity worse. One way to fix this could be dynamically changing the hash table size based on the number of packages. Another solution would be to handle hash table collisions using methods such as chaining, linear probing, or quadratic probing. Moreover, using different hash functions instead of modulo hash could minimize hash collisions such as mid-square hash or multiplicative string hash.

Manual sorting of the packages could also be required for any additional packages that require special delivery requirements. It is possible to automate this process by creating functions that deal with specific requirements and sorting them accordingly.

Another possible changes in scale that might occur is more trucks being available to be used. If this were the case, then instead of using the process of manually instantiating the truck objects, the process can be automated for the packages by evenly dividing the packages between the available number of trucks.

Another possibility is that more cities are included. If this were the case, then the program could account for this by dividing all packages into its respective trucks and cities based on the city information located in the hash table.

2.  Identify **two** other data structures that can meet the same criteria and requirements given in the scenario.

An adjacency matrix or an adjacency list can be used for the packages and to look up the package information.

a.  Describe how each data structure identified in part K2 is different from the data structure you chose to use in the solution.

The hash table holds the packages and includes a key-value pair, where the key is the package id and the value are the package information. A key is needed for this data structure to add, remove, or search a value. The average time complexity of an add, remove, and search is O(1) and, in the worst case, is O(n). However, an adjacency list or matrix would work since the routes can be graphed as vertexes and edges. Looking up whether two vertexes have an edge has a time complexity of O(n) where n would be the number of packages/vertexes. If it is an adjacency matrix, the lookup time complexity between two vertex would be O(1). The advantages of an adjacency list is in its space complexity which is O(n + m), where n is the vertexes and m and is the edges, while the space complexity of a matrix is O(n^2), where n is the vertex.