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 I use is mainly a greedy algorithm since I choose the option that is optimal at specific points of time. This is evident since each truck has a list of packages and they drive to the package that has the least distance to their current location. I say that this is the algorithm mostly used since I manually sort some packages to make sure it fits the special delivery requirements. However, after I manually sort it, however, the truck still automatically sorts it and goes to the closest package to it.

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.**

I created my own data structure to store the packages as, which I named Node, in the file. This node file includes the package id, package location, and package distance to whatever it is set. I use the hashmap to pull the information of each packages. Before the trucks start delivering, an array of these package objects are created with the distances set from the hub. But after every delivery, the distances of each node is recalculated to make sure the the truck always goes to the closest package.

O(1)

arrayOfPackages = []

O(n) to create and fill this hash

hashMapOfAllPackages = {…}

O(1) to create a single node object

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

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

Before every delivery:

Sort(arrayOfPackages.distances)

Go to first on the list of arrayOfPackages

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

In this, the csv file is loaded into the program and loaded into a hash table with the information from the csv file. The program then uses the hash table in order to create an array of packages with limited information. The truck class then takes in the hash table and the array of packages in order to continuously look up information about packages in the array as it makes its way through the route.

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

As noted, the program’s whole complexity is O(n^3) since iterates through n items and then sorts them after every delivery. No matter the increase in the amount of packages, this complexity will always stay the same since n is based on packages that is on the csv file. The complexity is also not terrible as there are algorithms that have exponential or factorial time complexities which will deeply hurt the scalability of the software, however, the one provided here is in cubic time.

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

The program’s efficiency is O(n^3) based on the fact that there are n items to be added to an array, and the array is then sorted with nlogn efficiency and then this is done every time the package is delivered. The program is easy to maintain since if you simply add information to the csv file, it will be automatically be added to the packages list. The only thing that is difficult is the special delivery requirements which might need some manual sort 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 lowest distance is always at the front of the array. I do this by calling a recalculate function 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.

I had a hash table to hold all the package information. This includes package id, address, zip, city, state, delivery status, delivery deadline time delivered, and time picked up. A hash table is really important since it allows you to store a lot of information in each key/value pair and lets you look it up with an average search complexity of O(1). This is important since I use the hash table to fill up the package object with the necessary information such as location before creating an array of it which is then sorted for my algorithm.

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

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

Greedy algorithm tries to find the optimal solution at a specific time. So this helps with delivering packages since I can delivery to the closest package which is optimal. Another strength of this is that, I don’t have to deal with large complexities since I am just looking at a specific time, and finding the optimal option right then and there.

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

I have shown that the packages are deliveries based on the specifications of the special delivery requirements and the algorithm helps to decrease the miles traveled as much as possible since it is finding the optimal package to deliver next every time it drops off a package.

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

A divide and conquer algorithm where maybe each packages is divided by their zip code and then make routes in order of zip code group.

A bruteforce algorithm where it finds the optimal routes possible, 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.

I don’t use a divide and conquer since the core of my algorithm is not dividing anything, the package is always trying to find the optimal solution for just that one package that it will deliver next.

Also, I am not going through every possibilities of routes which takes massive amounts of time and the algorithm I use is magnitudes faster than a bruteforce algorithm.

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

I think combining my custom created data structure and the hash table would organize the code and make it a lot easier to handle the project.

I could also have used a different data structure altogether such as a weighted graph which might 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 average search, add, delete complexities of O(1) and O(n) for their worst case.

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

I use an array to hold the packages for delivery and the efficiency of going to the next package is constant time since it is just removing the first package that was delivered and then looking at the first index of the array.

However, when I am creating the array and looking at the hash table for the information. This would be O(1) on average or O(n) in the worst case since I have to find the information of the package in the hash table which is O(1) on average and O(n) at worst, and then appending it to the array which is O(1)

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

If I am adding more package data to my hash table, if the size attribute is not change to reflect the size of the data then there might be more hash collisions when using the hash table to add, remove, or search which can make the time complexity worse.

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 could be used for the packages and can be used 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.

I chose a hash table to hold the packages and includes a key-value pair. A key is needed for this data structure to add, remove, or search a value. The average time complexity of a add, remove, and search is O(1) and O(n) at worst. However, an adjacency list or matrix would work as well 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 between two vertex would be O(1). The advantages of an adjacency list is in its space complexity where it 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.