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.
I used the nearest neighbor algorithm. The part that adjusts a lot is the part that decides the order of the packages to optimize delivery.

B. Write an overview of your program, in which you do the following:
1. Explain the algorithm’s logic using pseudocode.

```
keep track of the path
make sure each vertex isnt visited
set curr variable
for each vertex
    if not visited
        make visited
        set a very high low distance
        set current vertex
        for each vertex
            as long as vertex isnt current
                get the edge weight of those 2 vertices
                if that distance is new low
                    set that distance as low and append to path
```

2. Describe the programming environment you used to create the Python application.
I programmed my project in visual studio code version 1.75.1 with python version 3.9. 6 on my M1 macbook pro running macos 12.6.

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

4. Explain the capability of your solution to scale and adapt to a growing number of packages.
My solution would scale and function properly but it wouldn’t be the fastest if you got up into very high numbers of packages. The big O of my algorithm does not scale very well so it would have speed issues. The hash table would scale well because on computers nowadays there is quite a lot of memory so I wouldn't be worried about the table taking up too much memory.

5. Discuss why the software is efficient and easy to maintain.
It’s efficient because given any reasonable number of packages the algorithms and data structures I used have good big O’s. It’s easy to maintain because there is not a lot of code and it's well broken down into different classes and files.

6. Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).
One of if not the biggest strength of the hash table i used in my code is the fact that looking up objects is O(n). That makes it very efficient no matter how large the table is. Another strength is how efficient inserting is. One of the downsides to hash tables is if there are many items that end up hashed into the same bucket it can slow the table down and make it harder to use. Another

downside is if you need to iterate through some of the items in a hash table that can be slow since you have to hash every time.

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.

A way you could use the nearest neighbor algorithm and a hash table together to store the package data is you could sort the packages using the algorithm and hash them into the table using their given position that the algorithm gives.

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

The relationship that I based the data structure off of is the package id. I used the id since it is unique and that means the data structure would have an easy time getting good hash values to store the data.

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.

The main strength that made me use the nearest neighbor algorithm is how easy it is to implement. Another strength that made me want to use it is how it only needs 1 parameter to function properly and that was perfect for the problem at hand.

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

The algorithm delivers all packages in 89.7 miles, delivers all the packages on time, delivers all packages according to their notes. All of these points can be seen through the execution of option 1 in the program.

3. Identify two other named algorithms, different from the algorithm implemented in the solution, that would meet the requirements in the scenario.

The first other algorithm that came to mind was the greedy algorithm. The next algorithm I looked at was the A* pathfinding algorithm.

a. Describe how *each* algorithm identified in part I3 is different from the algorithm used in the solution.

The algorithm I used finds the package with the shortest distance from the current package and adds that to the list. The greedy algorithm is a more general algorithm that chooses the best next option based on really any attribute(price of an object for example). The a* algorithm is different because it takes into account the weight of the entire path, not just the weight of the current edge.

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

The main thing I would change is make better use of classes and their data instead of using random variables all over the main function.

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.

The packages were delivered in 89.7 miles, were delivered on time, according to their notes, my hashtable was efficient with a lookup function, this can be verified through the execution of my program.

a. Explain how the time needed to complete the look-up function is affected by changes in the number of packages to be delivered.
The lookup function doesn't change because I used a hash table where the lookup function is O(n).

b. Explain how the data structure space usage is affected by changes in the number of packages to be delivered.
The hash table uses up a lot of space when taking in big numbers of packages because the space has to be allocated when the hash table is created so its taking up a lot of space from the very beginning instead of just slowly growing.

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.
The number of trucks or cities wouldn't change the hash table very much because they are just variables inside each of my items; they aren't actual items themselves.

2. Identify two other data structures that could meet the same requirements in the scenario.
The first other data structure that comes to mind is a linked list, the second structure is just a normal dictionary.
a. Describe how *each* data structure identified in part K2 is different from the data structure used in the solution.
A linked list is very different because instead of being stored in a table it's stored in a line so in order to get the last package you'd have to go through every other package. A dictionary is similar to a hash table but items are just thrown into a dictionary in order where a hash table uses a hash function to store items.