**Part A: Algorithm Identification**

The algorithm used is the Nearest Neighbor Algorithm.

**Part B1: Logic Comments**

Load package list and current location

For each package in package list:

-Find the distance to all packages’ address

-Save the lowest distance to current location

-Set current location to package’s address

-Remove package from list

End

**Part B2: Development Environment**

Software:

OS: Windows 10 64-Bits

Python Version: 3.9.2 64-Bits

Hardware:

CPU: Intel i7-7700K 4.20GHz

RAM: 32 GB

GPU: GTX 1070

Storage: 512 GB SSD

**Part B3: Space-Time And Big-O**

|  |  |  |
| --- | --- | --- |
| **ChainHash.py** | **Space Complexity** | **Time Complexity** |
| 7 | O(N) | O(N) |
| 12 | O(1) | O(1) |
| 18 | O(1) | O(1) |
| 25 | O(N) | O(N) |
| 33 | O(1) | O(1) |
| 38 | O(N^2) | O(N^2) |
| **Total** | N^2+2N+3 = O(N^2) | N^2+2N+3 = O(N^2) |

|  |  |  |
| --- | --- | --- |
| **Package.py** | **Space Complexity** | **Time Complexity** |
| 5 | O(1) | O(1) |
| 19 | O(1) | O(1) |
| 24 | O(1) | O(1) |
| 29 | O(1) | O(1) |
| 34 | O(1) | O(1) |
| 39 | O(1) | O(1) |
| 44 | O(1) | O(1) |
| 49 | O(1) | O(1) |
| 54 | O(1) | O(1) |
| 59 | O(1) | O(1) |
| 64 | O(1) | O(1) |
| 69 | O(1) | O(1) |
| 73 | O(1) | O(1) |
| 78 | O(1) | O(1) |
| 84 | O(1) | O(1) |
| 89 | O(1) | O(1) |
| **Total** | 16 = O(1) | 16 = O(1) |

|  |  |  |
| --- | --- | --- |
| **ReadCSV.py** | **Space Complexity** | **Time Complexity** |
| 11 | O(N) | O(N) |
| 25 | O(1) | O(1) |
| 33 | O(N^2) | O(N^2) |
| 43 | O(N) | O(N) |
| **Total** | N^2+2N+1 = O(N^2) | N^2+2N+3 = O(N^2) |

|  |  |  |
| --- | --- | --- |
| **Delivery.py** | **Space Complexity** | **Time Complexity** |
| 21 | O(1) | O(1) |
| 31 | O(N^4) | O(N^4) |
| 68 | O(N^4) | O(N^4) |
| 104 | O(N^4) | O(N^4) |
| **Total** | 3N^4+1 = O(N^4) | 3N^4+1 = O(N^4) |

|  |  |  |
| --- | --- | --- |
| **Main.py** | **Space Complexity** | **Time Complexity** |
| 17 | O(N^3) | O(N^3) |
| **Total** | O(N^3) | O(N^3) |

|  |  |  |
| --- | --- | --- |
| **File** | **Space Complexity** | **Time Complexity** |
| ChainHash | O(N^2) | O(N^2) |
| Package | O(1) | O(1) |
| ReadCSV | O(N^2) | O(N^2) |
| Delivery | O(N^4) | O(N^4) |
| Main | O(N^3) | O(N^3) |
| **Total** | N^4+N^3+2N^2+1 = O(N^4) | N^4+N^3+2N^2+1 =O( N^4) |

**Part B4: Scalability and Adaptability**

The hash table is designed to handle a big number of packages, the more packages that are added the longer it could take to search in the hash table; however, the size of the hash table can be increased to reduce the average amount of packages per hash key.

The algorithm can scale and adapt to a growing number of packages. The nearest neighbor algorithm loops through each package in the list until it finds the closest package to deliver, if I add more packages this process will still return the closest address.

**Part B5: Software Efficiency and Maintainability**

The software is easy to maintain because it only requires two files as input without any other arguments and the code is split into different modules for easier understanding.

It is efficient because it does not consume much memory to run and the time it takes to complete the calculations is very fast.

**Part B6: Self-Adjusting Data Structures**

The main strength of a hash table is the speed to access items which can save us resources like time and processing power. Its average time-complexity is O(1).

One weakness of a hash table is that it does not allow null or empty values. Unlike SQL databases, hash tables need to perform a function to create the key to store the value which can’t be done with null or empty data.

**Part D: Data Structure**

The data structure that I used was a chaining hash table. It performed well with the algorithm, and it is self-adjusting to any number of packages.

**Part D1: Explanation of Data Structure**

The chaining hash table is a list with sub-listed items. Packages are loaded based on their ID to sub-lists, this is helpful because the nearest neighbor algorithm iterates through each package and the chaining hash table allows us to return the values much faster.

**Part G1: First Status Check**

Table

Description automatically generated

**Part G2: Second Status Check**

**Table

Description automatically generated**

**Part G3: Third Status Check**

**Table

Description automatically generated**

**Part H: Screenshots of Code Execution**

Text

Description automatically generated

**Part I1: Strengths of the Chosen Algorithm**

The algorithm is efficient for the distance requirement as it delivers all the packages in less than 140 miles total.

The algorithm can handle any number of packages as it adapts to the list given to find the closest distance.

**Part I2: Verification of Algorithm**

The algorithm meets all the requirements. The total distance traveled is 123.56 miles which is less than the limit of 140 miles. The packages are delivered to the correct destination and before any deadlines. The trucks travel at a speed of 18 miles per hour, leave no earlier than 8:00 a.m. and only have a maximum of 16 packages.

**Part I3: Other Possible Algorithms**

Greedy algorithm and Dijkstra’s algorithm.

**Part I3A: Algorithm Differences**

The greedy algorithm only checks for the next steps and doesn’t iterate through the whole list of options which means some packages will be ignored at any given time. Nearest neighbor iterates through the whole list of packages.

The Dijkstra’s algorithm finds the shortest path from the starting location to all others, the nearest neighbor algorithm finds the current closest location without comparing multiple paths.

**Part J: Different Approach**

I would like to load the trucks automatically by distance and priority to reduce the mileage. I would also like to add seconds to the time calculations to increase accuracy. Implementing a better delivery algorithm that would look at all possible paths and find the shortest one.

**Part K1: Verification of Data Structure**

The chaining hash table has methods to insert components into the hash table without using any additional libraries or classes.

**Part K1A: Efficiency**

The lookup function must iterate through the hash table which takes O(n) to complete when calculating what package to deliver, if there are more packages then the time to complete the algorithm will increase as well.

**Part K1B: Overhead**

The hash table will increase its size if there are more packages. More space will be use and there will be more packages per hash key.

**Part K1C: Implications**

Changes in the number of trucks would not influence the look-up time and the space would still be O(N).

Changes in the number of cities would not influence the look-up time, but it increases the space to O(N^2) because they would reference each other.

**Part K2: Other Data Structures**

Graphs and heap.

**Part K2A: Data Structure Differences**

The graph data structure stores nodes that are connected by edges. The difference is that I could store the packages using the distance as the connecting edges instead of a hash function, the time complexity is O(N) to go over each adjacent edge.

The heap is a type of binary tree in which parent node is compared to the children node. It could be used to create a priority queue in which the packages that need to be delivered first will be at the top of the tree. The hash table used in my program is different because it stores the values by a hash function instead of priority. The heap data structure has a time complexity of O(N).