**Core Algorithm Overview**

**Derek Mclaws**

**C950**

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

I will be using a simple greedy algorithm approach to find the package with the least amount of distance from current truck location.

**B1. Explain the algorithm’s logic using pseudocode.**

1. Manually load trucks based off constraints (delivery deadline, packages delivered together, and must be on truck 2. truck.packages = [1,2,3,4,5].
2. Truck.packages is then looped through retrieving corresponding address and distance data for each package. During the loop the package address and distance is compared to current location to find the package with smallest distance from current location.

For package in packagelist:

distanceBetween(fromAddresss, package\_address)

1. Once the optimal package is found the delivery is made. Package is marked delivered, removed from truck.packages, and miles traveled are added in the truck objects total miles traveled.
2. These steps are repeated until all the truck packages are delivered.

**B.2 Describe the programming environment you used to create the Python application.**

Written in Python v3.9 programming language. Using PyCharm Community v2021.2.3 andMicrosoft Windows 10 operating system.

**B.3 Evaluate the space-time complexity of each major segment of the program, and the entire program, using big-O notation.**

**Main.py:** O(N) This file runs user interface and data displaying functions.

**Functions:** allPackageData() – O(n), truckMileage() – O(1), individualPackage() – O(1), lookupPackageInfo() - O(1), ui() - O(1)

**distance.py:** O(N^2) This file handles loading distance data into a two-dimensional array, and distance lookups.

**Functions:** loadDistanceData() – O(N^2), distanceBetween() – O(1), minDistanceFrom() – O(N)

**Package.py:** O(1) This file handles package object creation.

**Functions:** lookupPrint() – O(1), get\_Address() – O(1), get\_Delivery\_Status – O(1), set\_Delivery\_Status() – O(1)

**hashTable.py:** O(N). This file creates and handles hashtable objects as well as reads in all package data.

**Functions:** init() - O(N), insert() – O(N), search() – O(N), remove() –O(N), loadPackageData() – O(N)

**truck.py:** O(N) This file creates and handles truck objects. Handles main delivery algorithm.

**Functions:** deliver\_package – O(1), truckDeliverPackages() – O(N)

**B.4 Explain the capability of your solution to scale and adapt to a growing number of packages.**

The application would require a few re-designs as it scales. As the number of packages grows, the loading of the trucks would not work optimally. The trucks are loaded manually based on user’s knowledge of the package constraints. The application does not determine the efficiency of loading the trucks. As the package count grows, the user’s time and effort increases. The trucks maximum package limit as well as other constraints would also present a need for changes by adding trucks and drivers. As for the handling of the packages and distances, the greedy algorithm is very scalable.

**B.5 Discuss why the software is efficient and easy to maintain.**

The application is easy to maintain since the code was organized in a clean object-oriented way that separates specific object actions and can be intuitively followed. The application is well documented through inline comments which explains how each function works and relates to the rest of the program. The logic of the algorithm is easy to follow and understand.

**B.6 Discuss the strengths and weaknesses of the self-adjusting data structures. (**hash-table**)**

**Strengths:** The hash-table implemented in the application is advantageous in relation to data retrieval speed. The assignment of key-value pairs allows for each piece of data to be identified uniquely, making data lookup fast and efficient.

**Weaknesses:** One of the greatest weaknesses of the hash-table data structure is that collisions can occur. As the collision count increases the bucket size increases as well, this can slow down data retrieval speed and require the use of rehashing methods.

**D, K.1, K.2 Explain how your data structure accounts for the relationship between the data points you are storing.**

The data structure implemented with this application’s algorithm is a chaining hash table. It takes the package Id’s as the unique identifier(key). The key is then hashed using a modulo method and the subsequent package data is stored in the assigned bucket. Chaining is used to address collision occurrences between hashed keys by storing a pointer to a list that contains all the keys in that location.

**K.1** The hash table implemented fulfills all project requirements as it efficiently stores and retrieves the required package data. The hash table’s search function serves as an integral part of the algorithm to retrieve package information used to calculate delivery routes. The application’s user interface reporting options are used to verify that all package delivery deadlines are met as well as total mileage for all trucks is under 140 miles. Total mileage traveled for all trucks was 96.8 miles.

**K.1A**, **K.1B, K.1C** Searching, inserting, and deleting methods of the hash table are coded to loop through a list of pointers and have a worst-case time complexity of O(n). The hash table implemented in this project has an initial bucket size of 10. As the number of packages increase, each bucket size would also increase in the number of keys to loop through. The look-up function would become increasingly slower due to the increased number of keys in each bucket. Increasing the number of initial buckets would allow for better distribution of hashed keys and reduce the number of iterations needed to be performed on bucket key lists. Ultimately the hash table bucket size should be adjusted and optimized as the number of packages increases. As the application grows with increased trucks and cities, there are few aspects about the application that should be considered. The increased number of trucks can optimize delivery time within deadline and mileage constraints, but overall time complexity of the algorithm would not be affected.

**K.2** Two other data types that could be used instead would be a stack. A stack could be used with an optimization algorithm to determine efficient delivery before storing into the data structure. This would allow for packages to be pushed onto the stack in the order for them to be delivered and simply popped when delivered. It could be easier for users to follow the data. A stack would operate on O(1). Similarly, the linked- list structure could be created by optimizing delivery before hand and having each node of data(package) point to the package that is next to be delivered.

**I.1 Describe at least two strengths for the algorithm used in the solution.**

The main strength of the algorithm used is that logic is simple and easy to understand. It made it easier for me to understand what the closest package is to go to deliver while having a think less deliver more attitude. The algorithm was also easy to code as it used programming logic that is common and well known.

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

**Heuristic Algorithm:** A heuristic algorithm can be used to design a solution that satisfies the problem as quickly as possible. Greedy and Heuristic approaches can be similar in achieving a globally optimal solution, but the main difference between them is speed. As the greedy algorithm worst case time complexity can be high, taking a heuristic approach can reduce time complexity. The heuristic approach may sacrifice finding the optimal solution for the sake of delivering a solution quickly.

**Dynamic Programming:** This type of algorithm breaks down the problem into sub-problems and work recursively to find the optimal solution. Greedy algorithm chooses the best solution now and works to achieve a globally optimal solution. Whereas dynamic programming considers all possible cases and chooses the best. Dynamic Programming is generally slower than the greedy algorithm.

**J. Describe what you would do differently.**

If I were to do this project differently, I would first design a way to optimally load packages onto trucks. The trucks were loaded manually through user logic, which was time consuming and inefficient for real world use. To add to the project, I would create a more graphical user interface, where more functionalities would be built in such as visually viewing delivery routes. Personally, I would have studied more about the implemented data structure (hash table) early on to understand how I needed to call the data. I got confused and it slowed my development time.