Core Algorithm Overview

**Stated Problem:**

The Western Governors University Parcel Service (WGUPS) needs to determine an efficient route and delivery distribution for their Daily Local Deliveries (DLD) because packages are not currently being consistently delivered by their promised deadline. The Salt Lake City DLD route has three trucks, two drivers, and an average of 40 packages to deliver each day. Each package has specific criteria and delivery requirements.

**A. Algorithm:** The **greedy algorithm** was primarily used in this assignment.

**B1 & B3 Algorithm’s logic and spacetime complexity:**

1. The information from the csv file is used to create package objects.

The time complexity of creating an object is **O(1).**

For row in package\_info\_csv:

Key = row[0]

Location\_number = row[1]

………

new\_package = Package (key, location\_number, address, city

state, zip, deadline, weight, notes)

1. These packages are inserted into the hash table and into priority lists based on their deadline.

The time complexity of inserting an object into a list and hash map is O(1)+O(1)=**O(1)**

If ‘9:00’ in package.deadline:

high\_priority\_packages.add(package)

if “9:30” in package.deadline

medium\_priority\_packages.add(package)

else:

low\_priority\_package.add(package)

hashmap.add(package)

1. The packages are also inserted into restricted lists and linked\_lists based on text in the package.notes field.

The time complexity of inserting an object into a list is **O(1)**

If “Truck 1” in truck.name or “Truck 3” in truck.name AND “truck 2” in package.notes:

Truck.restricted\_packages.add(package)

1. The truck will begin filling the truck in the order the packages will be delivered. To do this, the package will look in the highest priority list with remaining packages in it and find the package inside that has the shortest distance to the truck’s current location.

The time complexity of searching each item in the list to find the lowest distance is **O(n)**

(Truck’s current\_location is the last added package’s location.)

If high\_priority\_packages.size() > 0:

For loop to find the package with the lowest distance to the truck’s current location

Repeat with the medium and low priority lists.

…………………………..

1. Check to see if there are any lesser priority packages nearby or on the way from the truck’s current location and the next\_priority\_package decided on in step 3. Add those packages to the truck.bunch list and then after the for loop add the next\_priority\_location last. The order of the packages determine delivery order.

The time complexity of searching each item in the list to find packages on the way is **O(n)**

For package in combined\_list\_of\_all\_packages:

If the package’s distance from truck.current\_location + the package’s distance from the next\_priority\_package is less than 1.05x the distance from truck.current\_location to the next\_priority\_package:

truck.bunch.add(package) (packages will be added in bunches)

truck.bunch.add(next\_priority\_package)

1. Add the bunch to the truck.packages list. Also clear the bunch and also delete the package from its priority list.

The time complexity of appending the bunch to the truck.packages list is O(N). Also there is a nested for loop to remove the package from the package’s priority list. **O(N) \* O(N) = O(N^2)**

For package in truck.bunch:

Truck.packages.append(package)

For package2 in priolist\_of\_package:

If package2.key == package.key

Priolist.remove(package2)

1. Steps 3-6 are repeated until the truck has 15+ packages. It then possibly adds one more package on the way back to the hub.

Time complexity is **O(n)** because it searches the entire list for a package that matches the requirements.

For package in combined\_list\_of\_all\_packages:

If the package’s distance from truck.current\_location + the package’s distance from the hub is less than 1.05x the distance from truck.current\_location to the hub

truck.packages.add(package)

1. Repeat steps 3-6 for truck 3. Lastly, repeat the steps for truck 2 (truck two is last because it has the least restrictions)
2. The program will iterate over each package in each truck.packages and calculate miles and add time to each truck’s truck.time. The truck.time after reaching a package\_location will be that package’s estimated delivery time.

Time complexity is **O(n)** because it iterates over each object in the list.

For package in truck.packages:

If the expected delivery time > user\_input\_time:

If truck.start\_time > user\_input\_time:

Package.status = ‘At hub’

Else:

Package.status = ‘In transit’

Else:

Package.status = ‘Delivered at’ + expected\_delivery\_time ‘by’ truck.name

**The runtime complexity of the algorithm is O(1) + O(1) + O(1) + O(n) + O(n) + O(n^2) + O(n) + O(n) = O(N^2)**

**B2. Programming Environment:**

This project was created using Python version 3.10 and the IDE PyCharm on Windows 10. Hardware – I9 9900k processor

**B4 Capabilities to scale**

There was a bug, initially, that arises when all the trucks filled and emptied and attempted to fill again. If the bug still exists, the program is one bug fix away and one while loop away from being able to scale with any number of packages. The only restriction is the data format in the csv files must match the required datatype of the algorithm. One extra field was added to the Package objects by added it into the original csv files to bypass a need for a third csv file to connect the other two.

**B5 Software Efficiency and Maintainability:**

The algorithm isn’t inefficient because at worst, its time complexity is O(n^2). The software is also easy to maintain because the comments describing each method are very well written and should eliminate most confusion in trying to read the code. For more complicated parts of the algorithm, extra comments are added after each line of code to explain exactly what is meant to be going on during that part of the method.

**B6 Hash Tables – Pros & Cons:**

**Pros:**

Insertion and deletion is very efficient with a time complexity of O(1). Look up, which was required in retrieving the data for the end user also has a time complexity of O(1). This data structure is very efficient compared to other data structures when using it with large amounts of data.

**Cons:**

When dealing with small amounts of data, other data structures could possibly have a better time complexity than a hash table’s O(1).

**D Hash Tables & The Greedy Algorithm:**

With this program, data must be imported and then manipulated depending on input from the end-user. The greedy algorithm will decide the order that the packages will be delivered, exactly what time each delivery will take place, and on which truck. The algorithm will then call upon the hash table to change some of the package object’s data to suit the needs of the end-user.

**I 1:** This algorithm is very easy to implement because of its simplicity. Because of its simplicity, collaboration on a project making use of it would be much more easy than using a more complicated algorithm. Another benefit is easy scalability because how simple the algorithm is.

**I 3:** TheDivide and conquer algorithm or dynamic programming could have been used instead. With the greedy algorithm, the next step is determined by what will bring the quickest/largest gain as soon as possible. With dynamic programming, the next step is determined based on the current problem and the answers from previous interdependent subproblems. With the divide and conquer algorithm, the order doesn’t really matter too much because each subproblem is independent of each other.

**J:** If I did this project again, I would have spent more time evaluating the different algorithms and would decide on something a little more challenging but better.

**K1A:** The time for the look up function of a hash table is not affected, except for collisions, if there is a change in the number of packages to be delivered.

**K1B:** The space complexity of a hash table is O(N) meaning if the hash table has more packages, it would therefore take up more space.

**K1C:** Adding more cities or trucks, alone, would not change the lookup speed or the size of the space taken by the hash table as long as the number of packages stayed the same. Trucks aren’t stored in the hash table and the name of the city doesn’t affect anything.

**K2:** A binary search tree or a heap could have been used, instead, to complete the project. Even though both data structures are considered “hierarchical data structures” (*Overview of Data Structures: Set 2*), their time complexities differ greatly. A binary search tree needs a time complexity of O(n \* log(n)) to be created, meanwhile a heap can be created with O(n). Also, inserting and removing from a heap only takes O(log(n)) time, while the same in a BST takes O(n).(*Baeldung*) These two data structures likely would have been better for this project because they can be more efficient with low amounts of data. Hash tables on the other hand, are generally better with large amounts of data.

**Sources and Citations:**

Baeldung. “Heap vs Binary Search Tree.” *Baeldung on Computer Science*, 25 Aug. 2021, https://www.baeldung.com/cs/heap-vs-binary-search-tree.

“Overview of Data Structures: Set 2, *GeeksforGeeks*, 9 Nov. 2021, https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/.