Kathryn Fisher

Sept 4th, 2023

C950 – Data structures and algorithms

1. Identify a named self-adjusting algorithm (e.g., “Nearest Neighbor algorithm,” “Greedy algorithm”) that you used to create your program to deliver the packages.
   1. Nearest Neighbor Algorithm

B.  Write an overview of your program, in which you do the following:

1.  Explain the algorithm’s logic using pseudocode.

* Read in the Packages CSV into the hash table.
* Read in the distance CSV into an address dictionary and distances.
* Schedule the packages on each truck depending on the delivery time, address, and special notes.
* Find the nearest neighbor for the next package.
  + Check to see what the next delivery deadline is
  + Check if there are any similar addresses.
* Deliver the package to the address and repeat the process.

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

Pycharm

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

Operation time for this program will depend on a few factors:

* How many packages there are?
* How is the Data formatted in the CSV?

If the data is sorted, and it’s a straight insert, then it’s a 0(n) however, I have some manipulation in this program, so it makes it 0(n log n) due to factoring the delivery deadline and the similar address.

def shortestPathAlgorithm(truck, currentTime):  
 *"""  
 This function works along with the nearest neighbor algorithm and calculates the shortest path to the next package.  
 This is dependent on the delivery deadline and the distance between packages.  
 Big-O( O(n log n)) – due to the for loop running N times and then another n times for the nested for loop.  
 :param truck:  
 :param currentTime:  
 :return:  
 """* min\_distance = 20  
 min\_package = None  
 truck\_index = 0  
 package\_list = truck.getPackageList()  
 package\_info\_list = []  
 package\_delivery\_list = []  
 pack\_change = datetime.timedelta(hours=10, minutes=20)  
 difference = currentTime.seconds - pack\_change.seconds  
 if truck.getTruckName() == "truck3" and 0 < difference < 1400:  
 package = hashtable.search(9)  
 package.set\_deliveryAddress("410 S State St")  
 for p in package\_list: *# gets all the packageID package objects* package = hashtable.search(p)  
 package.set\_deliveryStatus("En Route")  
 package\_info\_list.append(package)  
 package\_info\_list.sort(key=lambda x: x.get\_deliveryDeadline(), reverse=False) *# sorts by delivery deadline* min\_pack\_deadline\_info = package\_info\_list[0]  
 min\_pack\_deadline = package\_info\_list[0].get\_deliveryDeadline().seconds *# min deadline based on sort* for i in package\_info\_list:  
 if min\_pack\_deadline == i.get\_deliveryDeadline().seconds:  
 package\_delivery\_list.append(i)  
 else:  
 continue  
 for pack in package\_delivery\_list:  
 address = pack.get\_deliveryAddress()  
 index = addressDict[address]  
 if truck\_index > index:  
 package\_dist = float(distances[truck\_index][index])  
 else:  
 package\_dist = float(distances[index][truck\_index])  
 if package\_dist < min\_distance:  
 min\_package = package  
 min\_distance = package\_dist  
 final = []  
 for p in package\_info\_list:  
 address = min\_pack\_deadline\_info.get\_deliveryAddress()  
 address\_split = address.split()  
 address\_new = ""  
 for i in address\_split:  
 if address\_split[0] == i:  
 continue  
 else:  
 address\_new += " " + i  
 if address\_new in p.get\_deliveryAddress() and p.get\_packageIDNumber() != 9:  
 *# if min\_pack\_deadline\_info.get\_deliveryAddress() == p.get\_deliveryAddress() and p.get\_packageIDNumber() != 9:* final.append(p)  
 else:  
 continue  
 return truck\_index, min\_distance, min\_package, final

def nearestneighbor(truck, currentTime, addressDict, distances, hashtable):  
 *"""  
 truck-list is set in main, and then we call the nearest neighbor algorithm per truck list  
 Big-O( O(n log n))  
 :param truck:  
 :param currentTime:  
 :param addressDict:  
 :param distances:  
 :param hashtable:  
 """* last\_package = None  
 count = 0  
 while truck.getPackageCount() > 0:  
 *# print(truck.getPackageCount())* truck\_index, min\_distance, min\_package, final = shortestPathAlgorithm(truck, currentTime)  
 minutes = round(((min\_distance \* 60) / 18.0))  
 currentTime = currentTime + datetime.timedelta(minutes=minutes)  
 if final.\_\_len\_\_() == 0:  
 min\_package.set\_deliverTime(currentTime)  
 truck.setTruckTime(currentTime)  
 truck.setTruckMiles(min\_distance)  
 *# print(truck.getTruckMiles().\_\_str\_\_() + "miles")* min\_package.set\_deliveryStatus("Delivered")  
 *# print(min\_package.get\_deliverTime())* truck\_package\_list = truck.getPackageList()  
 print(  
 truck.getTruckName() + " " + min\_package.get\_deliveryAddress() + " " + min\_package.get\_deliveryStatus() + " " + str(  
 min\_package.get\_packageIDNumber()) + " at " + str(min\_package.get\_deliverTime()) + " Due at " +  
 str(min\_package.get\_deliveryDeadline()) + " Packages remaining: " + str(truck.getPackageCount()))  
 truck\_package\_list.remove(min\_package.get\_packageIDNumber())  
 last\_package = min\_package  
 truck.setPackageList(truck\_package\_list)  
 truck.setPackageCount(len(truck\_package\_list))  
 else:  
 truck.setTruckMiles(min\_distance)  
 truck.setTruckTime(currentTime)  
 for f\_pack in final:  
 f\_pack.set\_deliverTime(currentTime)  
 *# print(truck.getTruckMiles().\_\_str\_\_() + "miles")* f\_pack.set\_deliveryStatus("Delivered")  
 *# print(min\_package.get\_deliverTime())* truck\_package\_list = truck.getPackageList()  
 print(  
 truck.getTruckName() + " " + f\_pack.get\_deliveryAddress() + " " + f\_pack.get\_deliveryStatus() + " " + str(  
 f\_pack.get\_packageIDNumber()) + " at " + str(f\_pack.get\_deliverTime()) + " Due at " +  
 str(f\_pack.get\_deliveryDeadline()) + " Packages remaining: " + str(truck.getPackageCount()))  
 truck\_package\_list.remove(f\_pack.get\_packageIDNumber())  
 truck.setPackageList(truck\_package\_list)  
 truck.setPackageCount(len(truck\_package\_list))

4.  Explain the capability of your solution to scale and adapt to a growing number of packages.  
 This program can take in an infinite number of packages through its CSV and hash table class. It can adapt with the delivery deadlines as well.

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

The software can record and report the trucks status and ensure that packages are being delivered as well as being delivered on time. Once the delivery starts, its as easy as sitting back and letting it run.

6.  Discuss the strengths and weaknesses of the self-adjusting data structures (e.g., the hash table).

* Strengths
  + Fast Access to elements based on Keys (Hash table in data structures 2023)
  + Time complexity is O(1) (Hash table in data structures 2023)
  + Fast with big datasets (Hash table in data structures 2023)
* Weaknesses
  + Many Collisions depending on how the hash table is designed (Hash table in data structures 2023)
  + Possible to unevenly distribute the keys (Hash table in data structures 2023)

C.  Write an original program to deliver *all* the packages, meeting *all* requirements, using the attached supporting documents “Salt Lake City Downtown Map,” “WGUPS Distance Table,” and the “WGUPS Package File.”

1.  Create an identifying comment within the first line of a file named “main.py” that includes your first name, last name, and student ID.

See attached Code.

2.  Include comments in your code to explain the process and the flow of the program.

See attached Code.

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.

Hash Table

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

The nearest neighbor algorithm works with the hash table class to retrieve packages from the hash table. Once it retrieves the package usually by the search method, it then can use the methods in the package class to retrieve the information that is with that package.

E.  Develop a hash table, without using *any* additional libraries or classes, that has an insertion function that takes the following components as input and inserts the components into the hash table:

•   package ID number

•   delivery address

•   delivery deadline

•   delivery city

•   delivery zip code

•   package weight

•   delivery status (e.g., delivered, en route)

def insert(self, package):

*"""  
 This is the function that inserts each package into the hashtable.  
 Big-O(O(n))  
 :param package:  
 """* index = int(package.packageIDNumber) % 10  
 self.hashtable[index].append(package)

F.  Develop a look-up function that takes the following components as input and returns the corresponding data elements:

•   package ID number

•   delivery address

•   delivery deadline

•   delivery city

•   delivery zip code

•   package weight

•   delivery status (i.e., “at the hub,” “en route,” or “delivered”), including the delivery time

def search(self, packageID: int):  
 *"""  
 This function searches for a particular package in the hashtable and returns the package  
 Big-O(O(n))  
 :type packageID: int  
 """* index = packageID % 10  
 for package in self.hashtable[index]:  
 if packageID == package.get\_packageIDNumber():  
 return package  
 return None

G.  Provide an interface for the user to view the status and info (as listed in part F) of *any* package at *any* time, and the total mileage traveled by *all* trucks. (The delivery status should report the package as *at the hub*, *en route*, or *delivered*. Delivery status *must* include the time.)

A screenshot of a computer

Description automatically generated

1. Provide screenshots to show the status of *all* packages at a time between 8:35 a.m. and 9:25 a.m.

A screenshot of a computer

Description automatically generated

1. Provide screenshots to show the status of *all* packages at a time between 9:35 a.m. and 10:25 a.m.

A screenshot of a computer

Description automatically generated

3.  Provide screenshots to show the status of *all* packages at a time between 12:03 p.m. and 1:12 p.m.

A screenshot of a computer

Description automatically generated

H.  Provide a screenshot or screenshots showing successful completion of the code, free from runtime errors or warnings, that includes the total mileage traveled by *all* trucks.

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

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.

* It can handle multi – class programs.
* No assumptions about the data beforehand!

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

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

Dijkstras algorithm, Greedy’s algorithm

1. Describe how *each* algorithm identified in part I3 is different from the algorithm used in the solution.
   1. Dijkstras algorithm
      1. This algorithm could pick an unvisted vertex, or in this case package, and then calculate the distance to it and every package there after. In our case, Nearest Neighbor only looks at the next package and not all the packages.
   2. Floyd-Warshall Algorithm
      1. This algorithm calculates the shortest distance between every single pair of nodes, then combines the shortest ones to get the final answer. Nearest Neighbor only looks at the next package distance and doesn’t take into consideration all the rest of the remaining packages.

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

I think if I did this project again, I would start with trying to pair the packages together by similar addresses. I spent a lot of time trying to get the packages to arrive ontime but I did not notice until testing the requirements that many of the packages are going to similar addresses. I think this could lead to a decrease in miles per truck and less complexity.

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.

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 time that it takes for the lookup function is important due to the number of packages that are inserted into the program. The more packages that need to be delivered, the more data the program must look through to get the correct package.

1. Explain how the data structure space usage is affected by changes in the number of packages to be delivered.

The bigger the data structure is (the hash table in this case) the more time it will take to find the right packages and perform the delivery. This is like smaller the data structure is, the less work the program needs to do.

1. 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.

If there are more trucks in the program, it’s possible that the program will need to call the search function more often to retrieve details for more than 1 truck at a time. If there was more than one city, the hash table would need to index the packages differently so that the retrieval of the data is faster.

2.  Identify **two** other data structures that could meet the same requirements in the scenario.

a.  Describe how *each* data structure identified in part K2 is different from the data structure used in the solution.

* Graph
  + A Graph is a data structure that has vertices and edges. In our program the vertices would be the packages and then the edges would be the distances. This is different than the hash table that uses an index to hold the package data.
* Binary Search Tree
  + A binary search Tree is made up of nodes or lists, and these are linked to other nodes. So, in our situation, the nodes would be the packages and then the other nodes would be linked to this. This is a little like what we did in the hash table, however the hash table had no recollection on all of the future nodes, just the one that would be coming next.

L.

Code:

Main Class:

*"""  
First name: Kathryn Fisher  
Class: C950 - Data Structures and Algorithms  
Student ID:  
  
Description: This is the main class that does all the manipulations of the data file as well as the delivery of the packages.  
Big-O( O(n log n))  
"""*import csv  
import datetime  
import re  
from tkinter import Tk  
  
from Classes import GUI  
from Classes import Truck  
from Classes.Hashtable import HashTable  
from Classes.package import Package  
  
  
def importPackages():  
 *"""  
 This class imports all the package information from the CSV and saves it to the hashtable via insert method.  
 Big-O( O(n log n))  
 :return: hashtable, special\_pack, remaining\_Pack  
 """* hashtable = HashTable()  
 *# Read in Package file* with open("WGUPS Package File copy.csv") as f:  
 d = list(csv.reader(f, delimiter=","))  
 remaining\_Pack = []  
 special\_pack = []  
 for i in d:  
 packageIDNumber = int(i[0])  
 delivery\_address = i[1]  
 delivery\_city = i[2]  
 delivery\_state = i[3]  
 delivery\_zip = i[4]  
 if i[5] == "EOD":  
 delivery\_deadline = datetime.timedelta(hours=16, minutes=0)  
 else: *# if there is a time listed in the column* time = re.split('(?<=\D)(?=\d)', i[5])  
 hour = None  
 minute = None  
 for r in time:  
 if ":" in r:  
 r = r.replace(":", "")  
 hour = int(r)  
 else:  
 if "AM" in r:  
 r = r.rstrip("AM ")  
 else:  
 hour = hour + 12  
 r = r.rstrip("PM")  
 minute = int(r.rstrip())  
 delivery\_deadline = datetime.timedelta(hours=hour, minutes=minute)  
 package\_weight = i[6]  
 notes = i[7]  
 package\_status = "At the Hub"  
 pack = Package(packageIDNumber, delivery\_address, delivery\_city, delivery\_state, delivery\_zip,  
 delivery\_deadline, package\_weight, package\_status, notes)  
 hashtable.insert(pack)  
 *# this adds the packages with any notes to the special pack list, otherwise it adds it to the remaining pack list.* if notes != "":  
 special\_pack.append(pack)  
 else:  
 remaining\_Pack.append(pack)  
 return hashtable, special\_pack, remaining\_Pack  
  
  
def specialPack(special\_pack, remaining\_Pack):  
 *"""  
 this function schedules the special packs depending on what the notes says and they also ensure that the special pack ids are not assigned in the remaining packs as well  
 Big-O( O(n log n))  
 :param special\_pack:  
 :param remaining\_Pack:  
 :return: remaining\_Pack, truck1, truck2, truck3  
 """* remaining\_Pack = remaining\_Pack  
 for pack in special\_pack:  
 notes = pack.get\_notes()  
 if "truck 2" in notes:  
 truck2.addPackage(pack.get\_packageIDNumber())  
 truck2.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif "truck3" in notes:  
 truck3.addPackage(pack.get\_packageIDNumber())  
 truck3.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif "truck1" in notes:  
 truck1.addPackage(pack.get\_packageIDNumber())  
 truck1.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif "Delayed" in notes:  
 truck2.addPackage(pack.get\_packageIDNumber())  
 truck2.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif "Wrong" in notes:  
 truck3.addPackage(pack.get\_packageIDNumber())  
 truck3.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif "Must" in notes:  
 original\_pack\_deadline = pack.get\_deliveryDeadline()  
 package\_condition = re.split('(?<=\D)(?=\d)',  
 notes) *# this splits the notes up if it has the word must. Only due to the fact that some packages MUST be on the truck with others* truck\_choice = None  
 min\_deadline = datetime.timedelta(hours=16, minutes=0)  
 all\_packages = [pack.get\_packageIDNumber()]  
 for packID in package\_condition:  
 packID = packID.replace(', ', '')  
 if packID.isdigit():  
 packID = int(packID)  
 current\_pack = hashtable.search(packID)  
 all\_packages.append(packID)  
 if current\_pack in special\_pack:  
 special\_pack.remove(current\_pack)  
 if current\_pack in remaining\_Pack:  
 remaining\_Pack.remove(current\_pack)  
 package = hashtable.search(packID)  
 if (package.get\_deliveryDeadline()).seconds < min\_deadline.seconds:  
 min\_deadline = package.get\_deliveryDeadline()  
 if min\_deadline <= truck1.getMinDeliveryDeadline():  
 for packID in all\_packages:  
 if truck2.search(packID):  
 truck2.getPackageList().remove(packID)  
 elif truck3.search(packID):  
 truck3.getPackageList().remove(packID)  
 elif truck1.search(packID):  
 continue  
 else:  
 truck1.addPackage(packID)  
 truck1.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif min\_deadline <= truck2.getMinDeliveryDeadline():  
 for packID in all\_packages:  
 if truck1.search(packID):  
 truck1.getPackageList().remove(packID)  
 elif truck3.search(packID):  
 truck3.getPackageList().remove(packID)  
 elif truck2.search(packID):  
 continue  
 else:  
 truck2.addPackage(packID)  
 truck2.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 elif min\_deadline <= truck3.getMinDeliveryDeadline():  
 for packID in all\_packages:  
 if truck2.search(packID):  
 truck2.getPackageList().remove(packID)  
 elif truck1.search(packID):  
 truck1.getPackageList().remove(packID)  
 elif truck3.search(packID):  
 continue  
 else:  
 truck3.addPackage(packID)  
 truck3.setMinDeliveryDeadline(pack.get\_deliveryDeadline())  
 truck1.setPackageCount(len(truck1.getPackageList()))  
 truck2.setPackageCount(len(truck2.getPackageList()))  
 truck3.setPackageCount(len(truck3.getPackageList()))  
 return remaining\_Pack, truck1, truck2, truck3  
  
  
def SchedulePackages(hash\_table, truck1, truck2, truck3, remaining\_Pack):  
 *"""  
 This function schedules the remaining packages. It uses the sorted method to put the packages in order of delivery deadline so that  
 the loop is already in order by soonest deadline.  
 Big-O( O(n log n))  
 :param hash\_table:  
 :param truck1:  
 :param truck2:  
 :param truck3:  
 :param remaining\_Pack:  
 :return:  
 """* while remaining\_Pack.\_\_len\_\_() != 0:  
 min\_deadline = remaining\_Pack[0].get\_deliveryDeadline().seconds  
 packlist = []  
 for sched\_pack in sorted(remaining\_Pack, key=lambda x: x.get\_deliveryDeadline(), reverse=False):  
 if (  
 sched\_pack.get\_deliveryDeadline()).seconds <= min\_deadline: *# checks for the soonest delivery deadline adds only those particular packages to list* min\_deadline = (sched\_pack.get\_deliveryDeadline()).seconds  
 package\_info = {"package": sched\_pack, "deliveryDeadline": min\_deadline}  
 packlist.append(package\_info)  
 remaining\_Pack.remove(sched\_pack)  
 else:  
 continue  
 truck\_list = [truck1, truck2, truck3]  
 truck\_list.sort(key=lambda x: x.getMinDeliveryDeadline(),  
 reverse=False) *# sorts the truck lists by the soonest delivery deadline* count = len(packlist)  
 for p in packlist:  
 package = p['package']  
 min\_deadline = p['deliveryDeadline']  
 if min\_deadline == truck\_list[0].getMinDeliveryDeadline():  
 packID = package.get\_packageIDNumber()  
 truck\_list[0].getPackageList().append(packID)  
 elif min\_deadline == 37800:  
 if len(truck2.getPackageList()) < 11:  
 packID = package.get\_packageIDNumber()  
 truck2.getPackageList().append(packID)  
 elif len(truck1.getPackageList()) < 14:  
 packID = package.get\_packageIDNumber()  
 truck1.getPackageList().append(packID)  
 else:  
 packID = package.get\_packageIDNumber()  
 truck3.getPackageList().append(packID)  
 elif min\_deadline == 57600:  
 if len(truck3.getPackageList()) < 16:  
 packID = package.get\_packageIDNumber()  
 truck3.getPackageList().append(packID)  
 elif len(truck1.getPackageList()) < 16:  
 packID = package.get\_packageIDNumber()  
 truck1.getPackageList().append(packID)  
 else:  
 packID = package.get\_packageIDNumber()  
 truck2.getPackageList().append(packID)  
 else: *# less than a certain amount of packages* if truck1.getPackageCount() < 17:  
 packID = package.get\_packageIDNumber()  
 truck1.getPackageList().append(packID)  
 elif truck3.getPackageCount() < 17:  
 packID = package.get\_packageIDNumber()  
 truck3.getPackageList().append(packID)  
 else:  
 packID = package.get\_packageIDNumber()  
 truck2.getPackageList().append(packID)  
 return truck1, truck2, truck3  
  
  
def importDistance():  
 *"""  
 this function saves all the data from the distance CSV and saves the index in the address dictionary and then the distances in a list.  
 Big-O( O(n))  
 :return:distances, addressDict  
 """* addressDict = {}  
 distances = []  
 *# reads in the distance CSV* with open("WGUPS Distance Table copy.csv") as f:  
 reader = csv.reader(f, delimiter=',')  
 index = 0  
 for row in reader:  
 address = row[0]  
 row.pop(0)  
 addressDict[address] = index  
 index += 1  
 distances.append(row)  
 return distances, addressDict  
  
  
def shortestPathAlgorithm(truck, currentTime):  
 *"""  
 This function works along with the nearest neighbor algorithm and calculates the shortest path to the next package.  
 This is dependent on the delivery deadline and the distance between packages.  
 Big-O( O(n log n))  
 :param truck:  
 :param currentTime:  
 :return:  
 """* min\_distance = 20  
 min\_package = None  
 truck\_index = 0  
 package\_list = truck.getPackageList()  
 package\_info\_list = []  
 package\_delivery\_list = []  
 pack\_change = datetime.timedelta(hours=10, minutes=20)  
 difference = currentTime.seconds - pack\_change.seconds  
 if truck.getTruckName() == "truck3" and 0 < difference < 1400:  
 package = hashtable.search(9)  
 package.set\_deliveryAddress("410 S State St")  
 for p in package\_list: *# gets all the packageID package objects* package = hashtable.search(p)  
 package.set\_deliveryStatus("En Route")  
 package\_info\_list.append(package)  
 package\_info\_list.sort(key=lambda x: x.get\_deliveryDeadline(), reverse=False) *# sorts by delivery deadline* min\_pack\_deadline\_info = package\_info\_list[0]  
 min\_pack\_deadline = package\_info\_list[0].get\_deliveryDeadline().seconds *# min deadline based on sort* for i in package\_info\_list:  
 if min\_pack\_deadline == i.get\_deliveryDeadline().seconds:  
 package\_delivery\_list.append(i)  
 else:  
 continue  
 for pack in package\_delivery\_list:  
 address = pack.get\_deliveryAddress()  
 index = addressDict[address]  
 if truck\_index > index:  
 package\_dist = float(distances[truck\_index][index])  
 else:  
 package\_dist = float(distances[index][truck\_index])  
 if package\_dist < min\_distance:  
 min\_package = package  
 min\_distance = package\_dist  
 final = []  
 for p in package\_info\_list:  
 address = min\_pack\_deadline\_info.get\_deliveryAddress()  
 address\_split = address.split()  
 address\_new = ""  
 for i in address\_split:  
 if address\_split[0] == i:  
 continue  
 else:  
 address\_new += " " + i  
 if address\_new in p.get\_deliveryAddress() and p.get\_packageIDNumber() != 9:  
 *# if min\_pack\_deadline\_info.get\_deliveryAddress() == p.get\_deliveryAddress() and p.get\_packageIDNumber() != 9:* final.append(p)  
 else:  
 continue  
 return truck\_index, min\_distance, min\_package, final  
  
  
def nearestneighbor(truck, currentTime, addressDict, distances, hashtable):  
 *"""  
 truck-list is set in main, and then we call the nearest neighbor algorithm per truck list  
 Big-O( O(n log n))  
 :param truck:  
 :param currentTime:  
 :param addressDict:  
 :param distances:  
 :param hashtable:  
 """* last\_package = None  
 count = 0  
 while truck.getPackageCount() > 0:  
 *# print(truck.getPackageCount())* truck\_index, min\_distance, min\_package, final = shortestPathAlgorithm(truck, currentTime)  
 minutes = round(((min\_distance \* 60) / 18.0))  
 currentTime = currentTime + datetime.timedelta(minutes=minutes)  
 if final.\_\_len\_\_() == 0:  
 min\_package.set\_deliverTime(currentTime)  
 truck.setTruckTime(currentTime)  
 truck.setTruckMiles(min\_distance)  
 *# print(truck.getTruckMiles().\_\_str\_\_() + "miles")* min\_package.set\_deliveryStatus("Delivered")  
 *# print(min\_package.get\_deliverTime())* truck\_package\_list = truck.getPackageList()  
 print(  
 truck.getTruckName() + " " + min\_package.get\_deliveryAddress() + " " + min\_package.get\_deliveryStatus() + " " + str(  
 min\_package.get\_packageIDNumber()) + " at " + str(min\_package.get\_deliverTime()) + " Due at " +  
 str(min\_package.get\_deliveryDeadline()) + " Packages remaining: " + str(truck.getPackageCount()))  
 truck\_package\_list.remove(min\_package.get\_packageIDNumber())  
 last\_package = min\_package  
 truck.setPackageList(truck\_package\_list)  
 truck.setPackageCount(len(truck\_package\_list))  
 else:  
 truck.setTruckMiles(min\_distance)  
 truck.setTruckTime(currentTime)  
 for f\_pack in final:  
 f\_pack.set\_deliverTime(currentTime)  
 *# print(truck.getTruckMiles().\_\_str\_\_() + "miles")* f\_pack.set\_deliveryStatus("Delivered")  
 *# print(min\_package.get\_deliverTime())* truck\_package\_list = truck.getPackageList()  
 print(  
 truck.getTruckName() + " " + f\_pack.get\_deliveryAddress() + " " + f\_pack.get\_deliveryStatus() + " " + str(  
 f\_pack.get\_packageIDNumber()) + " at " + str(f\_pack.get\_deliverTime()) + " Due at " +  
 str(f\_pack.get\_deliveryDeadline()) + " Packages remaining: " + str(truck.getPackageCount()))  
 truck\_package\_list.remove(f\_pack.get\_packageIDNumber())  
 truck.setPackageList(truck\_package\_list)  
 truck.setPackageCount(len(truck\_package\_list))  
  
  
*# initializes the trucklists*truck\_1\_list = []  
truck\_2\_list = []  
truck\_3\_list = []  
*# initializes the truck objects*truck1 = Truck.Truck("truck1", truck\_1\_list, datetime.timedelta(hours=8, minutes=0))  
truck2 = Truck.Truck("truck2", truck\_2\_list, datetime.timedelta(hours=9, minutes=5))  
truck3 = Truck.Truck("truck3", truck\_3\_list, None)  
*# imports the distances and the packages*distances, addressDict = importDistance()  
hashtable, special\_pack, remaining\_Pack = importPackages()  
*# Assigns the packages to the correct truck based on the special notes.*remaining\_Pack, truck1, truck2, truck3 = specialPack(special\_pack, remaining\_Pack)  
*# assigns the rest of the remaining packages to the trucks based on delivery time and amount of packages already on the truck*truck1, truck2, truck3 = SchedulePackages(hashtable, truck1, truck2, truck3, remaining\_Pack)  
root = Tk() *# starts the GUI*G = GUI.GUI(root, hashtable, truck1, truck2, truck3) *# initializes the GUI class with data we are wanting to display  
# sorts through each truck and assigns the package id's to the correct truck*for truck in [truck1, truck2, truck3]:  
 package\_list = truck.sort(truck.getPackageList(), hashtable)  
 truck.setPackageList(package\_list)  
 list = truck.getAllPackages(hashtable)  
 for i in list:  
 print()  
 G.refresh() *# refreshes the chart* root.update() *# Updates the window*root.after(1000, root.update()) *# updates the window after 1000 ms*print("truck1: " + str(truck1.getPackageCount()))  
nearestneighbor(truck1, truck1.getStartTime(), addressDict, distances, hashtable) *# starts delivery for Truck1*G.refresh()  
root.update()  
root.after(1000, root.update())  
print("truck1:" + str(truck1.getTruckMiles()))  
print("truck2: " + str(truck2.getPackageCount()))  
nearestneighbor(truck2, truck2.getStartTime(), addressDict, distances, hashtable) *# starts delivery for Truck2*G.refresh()  
root.update()  
print("truck2:" + str(truck2.getTruckMiles()))  
truck3.setStartTime(truck1.getTruckTime()) *# updates the start time based on truck1*print("truck3: " + str(truck3.getPackageCount()))  
nearestneighbor(truck3, truck3.getStartTime(), addressDict, distances, hashtable) *# starts delivery for Truck3*G.refresh()  
root.update()  
print("truck3:" + str(truck3.getTruckMiles()))  
Total\_Truck\_Miles = truck1.getTruckMiles() + truck2.getTruckMiles() + truck3.getTruckMiles()  
print("Total Truck Miles: " + Total\_Truck\_Miles.\_\_str\_\_())  
root.after(1000, root.update())  
G.getResult()  
root.mainloop() *# keeps GUI up waiting on user input*

Package Class:

*"""  
First name: Kathryn Fisher  
Class: C950 - Data Structures and Algorithms  
Student ID:  
  
Description: Package class that holds all the package information.  
Big-O( O(1))  
"""*import datetime  
  
  
class Package:  
  
 def \_\_init\_\_(self, package\_ID\_Number, delivery\_address, delivery\_city, delivery\_state, delivery\_zip\_code,  
 delivery\_deadline, package\_weight, delivery\_status, notes):  
 *"""  
 Class Constructor  
 Big-O(O(1))  
 :param package\_ID\_Number: int  
 :param delivery\_address: string  
 :param delivery\_city: string  
 :param delivery\_state: string  
 :param delivery\_zip\_code: int  
 :param delivery\_deadline: datetime.timedelta  
 :param package\_weight: int  
 :param delivery\_status: string  
 :param notes: string  
 """* self.packageIDNumber = package\_ID\_Number  
 self.deliveryAddress = delivery\_address  
 self.deliveryCity = delivery\_city  
 self.deliveryState = delivery\_state  
 self.deliveryZipCode = delivery\_zip\_code  
 self.deliveryDeadline = delivery\_deadline  
 self.packageWeight = package\_weight  
 self.deliveryStatus = delivery\_status  
 self.notes = notes  
 self.deliverTime = datetime.timedelta(hours=0,minutes=0)  
  
 def \_\_str\_\_(self):  
 *"""  
 This function returns the string of the package object.  
 Big-O(O(1))  
 :return: str(self.packageIDNumber) + " " + self.deliveryAddress  
 """* return str(self.packageIDNumber) + " " + self.deliveryAddress  
  
 def \_\_repr\_\_(self):  
 *"""  
 This function returns the representation of the package object.  
 Big-O(O(1))  
 :return: str(self.packageIDNumber) + " " + self.deliveryAddress  
 """* return str(self.packageIDNumber) + " " + self.deliveryAddress  
  
 def get\_packageIDNumber(self):  
 *"""  
 Get value of package ID Number  
 Big-O( O(1))  
 :return: PackageIDNumber  
 """* return self.packageIDNumber  
  
 def set\_packageIDNumber(self, package\_ID\_NUM):  
 *"""  
 set the value of the package ID Number  
 Big-O( O(1))  
 :param package\_ID\_NUM: int  
 """* self.packageIDNumber = package\_ID\_NUM  
  
 def get\_deliveryAddress(self):  
 *"""  
 Get Value of delivery address  
 Big-O( O(1))  
 :return: deliveryAddress  
 """* return self.deliveryAddress  
  
 def set\_deliveryAddress(self, delivery\_address):  
 *"""  
 set the value of the deliveryAddress  
 Big-O( O(1))  
 :param delivery\_address:  
 """* self.deliveryAddress = delivery\_address  
  
 def get\_deliveryDeadline(self):  
 *"""  
 Get Value of delivery deadline  
 Big-O( O(1))  
 :return: deliveryDeadline  
 """* return self.deliveryDeadline  
  
 def set\_deliveryDeadline(self, delivery\_deadline):  
 *"""  
 set the value of the delivery deadline  
 Big-O( O(1))  
 :param delivery\_deadline:  
 """* self.deliveryDeadline = delivery\_deadline  
  
 def get\_deliveryCity(self):  
 *"""  
 Get Value of delivery city  
 Big-O( O(1))  
 :return: deliveryCity  
 """* return self.deliveryCity  
  
 def set\_deliveryCity(self, delivery\_city):  
 *"""  
 set the value of the delivery city  
 Big-O( O(1))  
 :param delivery\_city:  
 """* self.deliveryCity = delivery\_city  
  
 def get\_deliveryState(self):  
 *"""  
 Get Value of delivery state  
 Big-O( O(1))  
 :return: deliveryState  
 """* return self.deliveryState  
  
 def set\_deliveryState(self, delivery\_state):  
 *"""  
 set the value of the delivery state  
 Big-O( O(1))  
 :param delivery\_state:  
 """* self.deliveryState = delivery\_state  
  
 def set\_deliveryZipCode(self, deliveryZipCode):  
 *"""  
 set the value of the delivery zip code  
 Big-O( O(1))  
 :param deliveryZipCode:  
 """* self.deliveryZipCode = deliveryZipCode  
  
 def get\_deliveryZipCode(self):  
 *"""  
 Get Value of package Zipcode  
 Big-O( O(1))  
 :return: deliveryZipcode  
 """* return self.deliveryZipCode  
  
 def get\_packageWeight(self):  
 *"""  
 Get Value of package weight  
 Big-O( O(1))  
 :return: packageWeight  
 """* return self.packageWeight  
  
 def set\_PackageWeight(self, packageWeight):  
 *"""  
 set the value of the package weight  
 Big-O( O(1))  
 :param packageWeight:  
 """* self.packageWeight = packageWeight  
  
 def get\_notes(self):  
 *"""  
 Get Value of Package Notes  
 Big-O( O(1))  
 :return: notes  
 """* return self.notes  
  
 def set\_notes(self, notes):  
 *"""  
 set the value of the package notes  
 Big-O( O(1))  
 :param notes:  
 """* self.notes = notes  
  
 def get\_deliverTime(self):  
 *"""  
 Get Value of delivery Time  
 Big-O( O(1))  
 :return: deliverTime  
 """* return self.deliverTime  
  
 def set\_deliverTime(self, time):  
 *"""  
 set the value of the delivery time  
 Big-O( O(1))  
 :param time:  
 """* self.deliverTime = time  
  
 def set\_deliveryStatus(self, delivery\_status):  
 *"""  
 set the value of the delivery status  
 Big-O( O(1))  
 :param delivery\_status:  
 """* self.deliveryStatus = delivery\_status  
  
 def get\_deliveryStatus(self):  
 *"""  
 Get Value of delivery Status  
 Big-O( O(1))  
 :return: deliveryStatus  
 """* return self.deliveryStatus

Hashtable class:

*"""  
First name: Kathryn Fisher  
Class: C950 - Data Structures and Algorithms  
Student ID:  
Description: This class is the data structure that holds the package information and stores it by its index.  
Big-O( O(n^2))  
"""*class HashTable:  
  
 def \_\_init\_\_(self):  
 *"""  
 Class constructor  
 Big-O(O(n))  
 """* self.hashtable = []  
 for i in range(10):  
 self.hashtable.append([])  
  
 def insert(self, package):  
 *"""  
 This is the function that inserts each package into the hashtable.  
 Big-O(O(n))  
 :param package:  
 """* index = int(package.packageIDNumber) % 10  
 self.hashtable[index].append(package)  
  
 def remove(self, package):  
 *"""  
 This function removes the packages from the hashtable after its delivered.  
 Big-O(O(n))  
 :param package:  
 """* index = int(package.packageIDNumber) % 10  
 self.hashtable[index].remove(package)  
  
 def search(self, packageID: int):  
 *"""  
 This function searches for a particular package in the hashtable and returns the package  
 Big-O(O(n))  
 :type packageID: int  
 """* index = packageID % 10  
 for package in self.hashtable[index]:  
 if packageID == package.get\_packageIDNumber():  
 return package  
 return None  
  
 def findPackage(self, address):  
 *"""  
 this function finds the package object that holds the address that the program is requesting.  
 Big-O( O(n^2))  
 """* for i in self.hashtable:  
 hashi = list(i)  
 for x in hashi:  
 if x.get\_deliveryAddress() in address:  
 sending\_pack = x  
 return sending\_pack  
 return None  
  
 def getAllPackages(self):  
 *"""  
 this function gets all the packages in the hashtable no matter what truck object is requesting the information  
 Big-O( O(n^2))  
 """* package\_list = []  
 for i in self.hashtable:  
 packages = list(i)  
 for package in packages:  
 package\_list.append(package)  
 return package\_list  
  
 def \_\_str\_\_(self):  
 *"""  
 This function returns the string of the hashtable object.  
 Big-O(O(n))  
 """* retstr = ""  
 for i in range(10):  
 retstr += str(i) + ":" + str(self.hashtable[i])  
 retstr += '\n'  
 return retstr  
  
 def \_\_repr\_\_(self):  
 *"""  
 This function returns the representation (string) of the hashtable object.  
 Big-O(O(n))  
 """* retstr = ""  
 for i in range(10):  
 retstr += str(i) + ":" + str(self.hashtable[i])  
 retstr += '\n'  
 return retstr

GUI Class:

*"""  
First name: Kathryn Fisher  
Class: C950 - Data Structures and Algorithms  
Student ID:  
  
Description: This class controls the user interface of the program. It utilizes the tkinter package.  
Big-O( O(n log n))  
"""*import datetime  
import tkinter as tk  
import tkinter.font as tkFont  
from tkinter import ttk  
  
  
class GUI:  
 def \_\_init\_\_(self, root, hashtable, truck1, truck2, truck3):  
 *"""  
 GUI class constructor  
 Big-O(O(n))  
 :param root: main screen  
 :param hashtable: hashtable from main class  
 :param truck1: truck object  
 :param truck2: truck object  
 :param truck3: truck object  
 """  
 # setting title* self.hashtable = hashtable  
 self.truck1 = truck1  
 self.truck2 = truck2  
 self.truck3 = truck3  
 self.root = root  
 self.list\_of\_pack = []  
 root.title("Package Delivery Program - Supervisor") *# title of the screen  
 # setting window size* width = 1550  
 height = 900  
 screenwidth = root.winfo\_screenwidth()  
 screenheight = root.winfo\_screenheight()  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 root.geometry(alignstr)  
 root.resizable(width=True, height=True)  
 self.treeView() *# handles the chart that the package info is stored on* self.getResult() *# gets the results for the reports* def getReport(self, time1, time2, newWindow):  
 *"""  
 This function returns the information requested by the evaluation for specific times in the program.  
 Big-O(O(n))  
 :param time1:  
 :param time2:  
 :param newWindow:  
 """* col = ["PackageID", "PackageStatus",  
 "Delivery Time"] *# this is the three columns used in the treeview for the report window* tree = ttk.Treeview(newWindow, columns=col, height=40, show='headings')  
 for c in col: *# takes each column and inserts it into the treeview.* col\_name = c.\_\_str\_\_()  
 tree.column(col\_name, width=80)  
 tree.heading(col\_name, text=col\_name)  
 tree.grid(row=0, column=1, sticky='nsew') *# signifies where the columns start* start\_time = datetime.timedelta(hours=8, minutes=0) *# start time of the first truck* for n in self.list\_of\_pack: *# iterates through the pack and searches for any package between the duration requested.  
 # it also factors in if there are any packages that were delivered after the start time to the time of the first constraint.* if time1.seconds <= n.get\_deliverTime().seconds <= time2.seconds:  
 tree.insert('', tk.END, values=(  
 n.get\_packageIDNumber(), n.get\_deliveryStatus(), n.get\_deliverTime()))  
 elif start\_time.seconds <= n.get\_deliverTime().seconds <= time1.seconds:  
 tree.insert('', tk.END, values=(  
 n.get\_packageIDNumber(), "Delivered", n.get\_deliverTime()))  
 else:  
 tree.insert('', tk.END, values=(  
 n.get\_packageIDNumber(), "En Route", datetime.timedelta(hours=0, minutes=0)))  
  
 tree.grid(row=0, column=1, sticky='nsew')  
  
 def treeView(self):  
 *"""  
 This function initializes all the widgets, Buttons, and charts for the treeview.  
 Big-O(O(1))  
 """* pack\_list = []  
 root, tree, pack\_list, col = self.populatebox(self.root)  
 Exit\_button = tk.Button(root)  
 Exit\_button["bg"] = "#efefef"  
 ft = tkFont.Font(family='Times', size=10)  
 Exit\_button["font"] = ft  
 Exit\_button["fg"] = "#000000"  
 Exit\_button["justify"] = "center"  
 Exit\_button["text"] = "Exit"  
 Exit\_button.place(x=20, y=800, width=141, height=30)  
 Exit\_button["command"] = self.Exit\_button\_command  
  
 Report\_Button1 = tk.Button(root)  
 Report\_Button1["bg"] = "#efefef"  
 ft = tkFont.Font(family='Times', size=10)  
 Report\_Button1["font"] = ft  
 Report\_Button1["fg"] = "#000000"  
 Report\_Button1["justify"] = "center"  
 Report\_Button1["text"] = "Report between 8:35 a.m. and 9:25 a.m"  
 Report\_Button1.place(x=960, y=750, width=200, height=30)  
 Report\_Button1["command"] = self.Report\_Button1\_Command  
  
 Report\_Button2 = tk.Button(root)  
 Report\_Button2["bg"] = "#efefef"  
 ft = tkFont.Font(family='Times', size=10)  
 Report\_Button2["font"] = ft  
 Report\_Button2["fg"] = "#000000"  
 Report\_Button2["justify"] = "center"  
 Report\_Button2["text"] = "Report between 9:35 a.m. and 10:25 a.m"  
 Report\_Button2.place(x=960, y=800, width=200, height=30)  
 Report\_Button2["command"] = self.Report\_Button2\_Command  
  
 Report\_Button3 = tk.Button(root)  
 Report\_Button3["bg"] = "#efefef"  
 ft = tkFont.Font(family='Times', size=10)  
 Report\_Button3["font"] = ft  
 Report\_Button3["fg"] = "#000000"  
 Report\_Button3["justify"] = "center"  
 Report\_Button3["text"] = "Report between 12:03 p.m. and 1:12 p.m"  
 Report\_Button3.place(x=960, y=850, width=200, height=30)  
 Report\_Button3["command"] = self.Report\_Button3\_Command  
  
 def getResult(self):  
 *"""  
 This specifically handles the final mileage for the trucks shown on the bottom tree view and then adds a label to display the total after the program is complete.  
 Big-O(O(n))  
 """* col = ["Truck", "Total Miles"]  
 truck\_tree = ttk.Treeview(self.root, columns=col, height=5, show='headings')  
 truck\_tree.place(x=300, y=1500, width=5, height=5)  
 truck\_tree.column("Truck", minwidth=5)  
 truck\_tree.heading("Truck", text="Truck")  
 truck\_tree.column("Total Miles", minwidth=5)  
 truck\_tree.heading("Total Miles", text="Total Miles")  
 truck\_tree.insert('', tk.END, values=(  
 self.truck1.getTruckName(), self.truck1.getTruckMiles()))  
 truck\_tree.insert('', tk.END, values=(  
 self.truck2.getTruckName(), self.truck2.getTruckMiles()))  
 truck\_tree.insert('', tk.END, values=(  
 self.truck3.getTruckName(), self.truck3.getTruckMiles()))  
 truck\_tree.grid(row=1, column=0)  
 Total\_Truck\_Miles = self.truck1.getTruckMiles() + self.truck2.getTruckMiles() + self.truck3.getTruckMiles()  
  
 Total\_Truck\_Miles = self.truck1.getTruckMiles() + self.truck2.getTruckMiles() + self.truck3.getTruckMiles()  
 TruckMilesLabel = ttk.Label(self.root, text="Total Truck Miles:")  
 TruckMilesLabel.place(x=1200, y=800)  
  
 TruckMilesResultLabel = ttk.Label(self.root, text=Total\_Truck\_Miles)  
 TruckMilesResultLabel.place(x=1310, y=800)  
  
 def Report\_Button1\_Command(self):  
 *"""  
 This function controls what happens after the report button 1 is pressed  
 Big-O(O(n)  
 """* newWindow = tk.Toplevel(self.root)  
  
 *# sets the title of the new window* newWindow.title("Report between 8:35 a.m. and 9:25 a.m")  
  
 *# sets the geometry of toplevel* screenwidth = self.root.winfo\_screenwidth()  
 screenheight = self.root.winfo\_screenheight()  
 width = 900  
 height = 900  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 newWindow.geometry(alignstr)  
 time1 = datetime.timedelta(hours=8, minutes=35)  
 time2 = datetime.timedelta(hours=9, minutes=25)  
 self.getReport(time1, time2, newWindow)  
  
 def Report\_Button2\_Command(self):  
 *"""  
 This function controls what happens after the report button 2 is pressed  
 Big-O(O(n))  
 """* newWindow = tk.Toplevel(self.root)  
  
 *# sets the title of the  
 # Toplevel widget* newWindow.title("Report between 9:35 a.m. and 10:25 a.m")  
  
 *# sets the geometry of toplevel* screenwidth = self.root.winfo\_screenwidth()  
 screenheight = self.root.winfo\_screenheight()  
 width = 900  
 height = 900  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 newWindow.geometry(alignstr)  
 time1 = datetime.timedelta(hours=9, minutes=35)  
 time2 = datetime.timedelta(hours=10, minutes=25)  
 self.getReport(time1, time2, newWindow)  
  
 def Report\_Button3\_Command(self):  
 *"""  
 This function controls what happens after the report button 3 is pressed  
 Big-O(O(n))  
 """* newWindow = tk.Toplevel(self.root)  
  
 *# sets the title of the  
 # Toplevel widget* newWindow.title("Report between 12:03 p.m. and 1:12 p.m")  
  
 *# sets the geometry of toplevel* screenwidth = self.root.winfo\_screenwidth()  
 screenheight = self.root.winfo\_screenheight()  
 width = 900  
 height = 900  
 alignstr = '%dx%d+%d+%d' % (width, height, (screenwidth - width) / 2, (screenheight - height) / 2)  
 newWindow.geometry(alignstr)  
 time1 = datetime.timedelta(hours=12, minutes=3)  
 time2 = datetime.timedelta(hours=13, minutes=12)  
 self.getReport(time1, time2, newWindow)  
  
 def populatebox(self, root):  
 *"""  
 this function controls the chart on the main screen, which displays all the packages  
 Big-O(O(n log n))  
 :param root:  
 :return:  
 """* list\_of\_pack = []  
 truck1\_pack = self.truck1.getAllPackages(hashtable=self.hashtable)  
 truck2\_pack = self.truck2.getAllPackages(hashtable=self.hashtable)  
 truck3\_pack = self.truck3.getAllPackages(hashtable=self.hashtable)  
 list\_of\_pack.extend(truck1\_pack)  
 list\_of\_pack.extend(truck2\_pack)  
 list\_of\_pack.extend(truck3\_pack)  
 col = list\_of\_pack[0].\_\_dir\_\_()  
 col = col[0:10]  
  
 tree = ttk.Treeview(root, columns=col, height=40, show='headings')  
 for c in col:  
 col\_name = c.\_\_str\_\_()  
 if "State" in col\_name or "Zip" in col\_name or "packageID" in col\_name or "Status" in col\_name or "Weight" in col\_name:  
 tree.column(col\_name, width=20)  
 tree.heading(col\_name, text=col\_name)  
 else:  
 tree.column(col\_name, width=20)  
 tree.heading(col\_name, text=col\_name)  
 tree.grid(row=0, column=0, sticky=tk.NSEW)  
 truck1\_pack = self.truck1.getAllPackages(hashtable=self.hashtable)  
 for n in truck1\_pack:  
 tree.insert('', tk.END, values=(  
 n.get\_packageIDNumber(), n.get\_deliveryAddress(), n.get\_deliveryCity(), n.get\_deliveryState(),  
 n.get\_deliveryZipCode(),  
 n.get\_deliveryDeadline(), n.get\_packageWeight(), n.get\_deliveryStatus(), n.get\_notes(),  
 n.get\_deliverTime()))  
 tree.grid(row=0, column=0, sticky='nsew')  
 scrollbar = ttk.Scrollbar(root, orient=tk.VERTICAL, command=tree.yview())  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.grid(row=0, column=1, sticky='ns')  
 self.list\_of\_pack = sorted(list\_of\_pack, key=lambda x: x.get\_packageIDNumber(), reverse=False)  
 return root, tree, list\_of\_pack, col  
  
 def refresh(self):  
 *"""  
 This function refreshes the treeview after each truck has completed its deliveries.  
 Big-O(O(n log n))  
 """* col = self.list\_of\_pack[0].\_\_dir\_\_()  
 col = col[0:10]  
 *# for item in col:  
 # all\_pack\_box.insert(END, item)  
 # for items in list\_of\_pack:  
 # all\_pack\_box.insert(items, END)* tree = ttk.Treeview(self.root, columns=col, height=40, show='headings')  
 for c in col:  
 col\_name = c.\_\_str\_\_()  
 if "State" in col\_name or "Zip" in col\_name or "packageID" in col\_name or "Status" in col\_name or "Weight" in col\_name:  
 tree.heading(col\_name, text=col\_name)  
 tree.column(col\_name, width=100)  
 else:  
 tree.heading(col\_name, text=col\_name)  
 tree.column(col\_name, minwidth=0)  
 tree.grid(row=0, column=0, sticky=tk.NSEW)  
 for n in self.list\_of\_pack:  
 tree.insert('', tk.END, values=(  
 n.get\_packageIDNumber(), n.get\_deliveryAddress(), n.get\_deliveryCity(), n.get\_deliveryState(),  
 n.get\_deliveryZipCode(), n.get\_deliveryDeadline(), n.get\_packageWeight(), n.get\_deliveryStatus(),  
 n.get\_notes(), n.get\_deliverTime()))  
 tree.grid(row=0, column=0, sticky='nsew')  
 scrollbar = ttk.Scrollbar(self.root, orient=tk.VERTICAL, command=tree.yview())  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.grid(row=0, column=1, sticky='ns')  
 self.list\_of\_pack = sorted(self.list\_of\_pack, key=lambda x: x.get\_packageIDNumber(), reverse=False)  
  
 def Exit\_button\_command(self):  
 *"""  
 This function closes out the program.  
 Big-0(O(1))  
 """* exit()

Truck Class:

*"""  
First name: Kathryn Fisher  
Class: C950 - Data Structures and Algorithms  
Student ID:  
  
Description: This class holds all the truck information such as the truck name, package list etc  
Big-O( O(n log n))  
"""*import datetime  
  
  
class Truck:  
  
 def \_\_init\_\_(self, name, package\_list, truck1\_start\_time):  
 *"""  
 Class Constructor  
 Big-O(1)  
 :param name:  
 :param package\_list:  
 :param truck1\_start\_time:  
 """* self.truckName = name  
 self.packageList = package\_list  
 self.packageCount = len(self.packageList)  
 self.miles = 0  
 self.start\_time = truck1\_start\_time  
 self.truckTime = None  
 self.deadline = datetime.timedelta(hours=16, minutes=00)  
  
 def \_\_str\_\_(self):  
 *"""  
 returns the string representation of the truck name  
 Big-O(0(1))  
 :return:  
 :rtype:  
 """* return self.truckName + " " + self.miles  
  
 def \_\_repr\_\_(self):  
 *"""  
 returns the string representation of the truck name  
 Big-O(0(1))  
 :return:  
 :rtype:  
 """* return self.truckName + " " + self.miles  
  
 def setStartTime(self, StartTime):  
 *"""  
 Sets Value for start time  
 Big-O(O(1))  
 :param StartTime:  
 """* self.start\_time = StartTime  
  
 def getStartTime(self):  
 *"""  
 gets value of start time  
 Big-O(O(1))  
 :return:  
 """* return self.start\_time  
  
 def setTruckTime(self, TruckTime):  
 *"""  
 Sets Value for truck time  
 Big-O(O(1))  
 :param TruckTime:  
 """* self.truckTime = TruckTime  
  
 def getTruckTime(self):  
 *"""  
 gets value of start time  
 Big-O(O(1))  
 :return:  
 """* return self.truckTime  
  
 def setPackageCount(self, package\_count):  
 *"""  
 Sets Value for Package Count  
 Big-O(O(1))  
 :param package\_count:  
 :type package\_count: int  
 """* self.packageCount = package\_count  
  
 def getPackageCount(self):  
 *"""  
 gets value of package count  
 Big-O(O(1))  
 :return:  
 :rtype: int  
 """* return len(self.getPackageList())  
  
 def setTruckName(self, name):  
 *"""  
 Sets Value for truck name  
 Big-O(O(1))  
 :param name:  
 :type name:  
 """* self.truckName = name  
  
 def getTruckName(self):  
 *"""  
 gets value of truck name  
 Big-O(O(1))  
 :return:  
 :rtype: str  
 """* return self.truckName  
  
 def setPackageList(self, package\_list: list):  
 *"""  
 Sets Value for package list  
 Big-O (O(1))  
 :param package\_list:  
 :type package\_list: List[int]  
 """* package\_count: int = package\_list.\_\_len\_\_()  
 self.setPackageCount(package\_count)  
 self.packageList = package\_list  
  
 def getPackageList(self):  
 *"""  
 gets value of package list  
 Big-O(O(1))  
 :return:  
 :rtype: List  
 """* return self.packageList  
  
 def setTruckMiles(self, miles):  
 *"""  
 Sets Value for truck Miles  
 Big-O (O(1))  
 :param miles:  
 :type miles: float  
 """* self.miles = miles + self.miles  
  
 def getTruckMiles(self):  
 *"""  
 gets Value for truck Miles  
 Big-O (O(1))  
 :return:  
 :rtype: int  
 """* return self.miles  
  
 def setMinDeliveryDeadline(self, deadline):  
 *"""  
 Big-O (O(1))  
 Sets Value for Min Delivery Deadline  
 :param deadline:  
 :type deadline: datetime.timedelta  
 """* difference = deadline - self.deadline  
 if difference > datetime.timedelta(minutes=1):  
 self.deadline = self.deadline  
 else:  
 self.deadline = deadline  
  
 def getMinDeliveryDeadline(self):  
 *"""  
 gets Value for truck Miles  
 Big-O (O(1))  
 :return:  
 :rtype: datetime.timedelta  
 """* return self.deadline  
  
 def sort(self, packages, hashtable):  
 *"""  
 Sorts the packages in order of the delivery deadline  
 Big-O(N log n)  
 :param packages:  
 :type packages: List[int]  
 :param hashtable:  
 :type hashtable: Classes.Hashtable.HashTable  
 :return:  
 :rtype: List[int]  
 """* package\_order = []  
 for pack in packages:  
 package\_info = hashtable.search(pack)  
 package\_order.append(package\_info)  
 packages.remove(pack)  
 self.setMinDeliveryDeadline(package\_order[0].get\_deliveryDeadline())  
 for pack in sorted(package\_order, key=lambda x: x.get\_deliveryDeadline(), reverse=False):  
 packID = pack.get\_packageIDNumber()  
 packages.append(packID)  
 return packages  
  
 def search(self, PackID):  
 *"""  
 Searches for a particular packID  
 Big-0(N)  
 :param PackID:  
 :type PackID: int  
 :return:  
 :rtype: bool  
 """* pack\_list = self.getPackageList()  
 if PackID in pack\_list:  
 return True  
 else:  
 return False  
  
 def getAllPackages(self, hashtable):  
 *"""  
 Gets all the packages for the entire program  
 Big-O(N)  
 :param hashtable:  
 :return:  
 """* pack\_list = self.getPackageList()  
 new\_list = []  
 for i in pack\_list:  
 package = hashtable.search(i)  
 new\_list.append(package)  
 return new\_list  
  
 def addPackage(self, package):  
 *"""  
 Adds a package to the package list  
 Big-O(0(1))  
 :param package:  
 """* self.getPackageList().append(package)

**Citations**

Ayt. (2023, June 4). *Hash table in data structures*. Medium. https://levelup.gitconnected.com/hash-table-in-data-structures-9df2301d352a