A: Algorithm Selection

This program uses the 2-opt search algorithm to find the shortest path between addresses. The 2-opt algorithm optimizes the route to avoid self-crossing paths, potentially finding an optimal route (Croes, 1958). The 2-opt algorithm operates in O(n2)O(n^2)O(n2) time, similar to the nearest neighbor (NN) algorithm, but it appears to be a more elegant solution to this problem. The NN algorithm typically does not produce the most optimal tour (Gutin et al., 2002). My objective was to maintain a runtime around O(n2)O(n^2)O(n2); otherwise, I could have opted for the 3-opt algorithm, which operates in O(n3)O(n^3)O(n3).

B1: Logic Comments

The following explanation describes the process of how my program solves this problem:

Load/store package data given to us in an excel/csv format

Read the csv file to parse each data row.
Take those parsed rows and create a package object.
Store those package objects into a hash table.

Load/store distance data given to us in an excel/csv format

Read the csv file to parse each data row.
Clean the parsed data so that it is usable.
Store the data so that it is easily iterable/retrievable

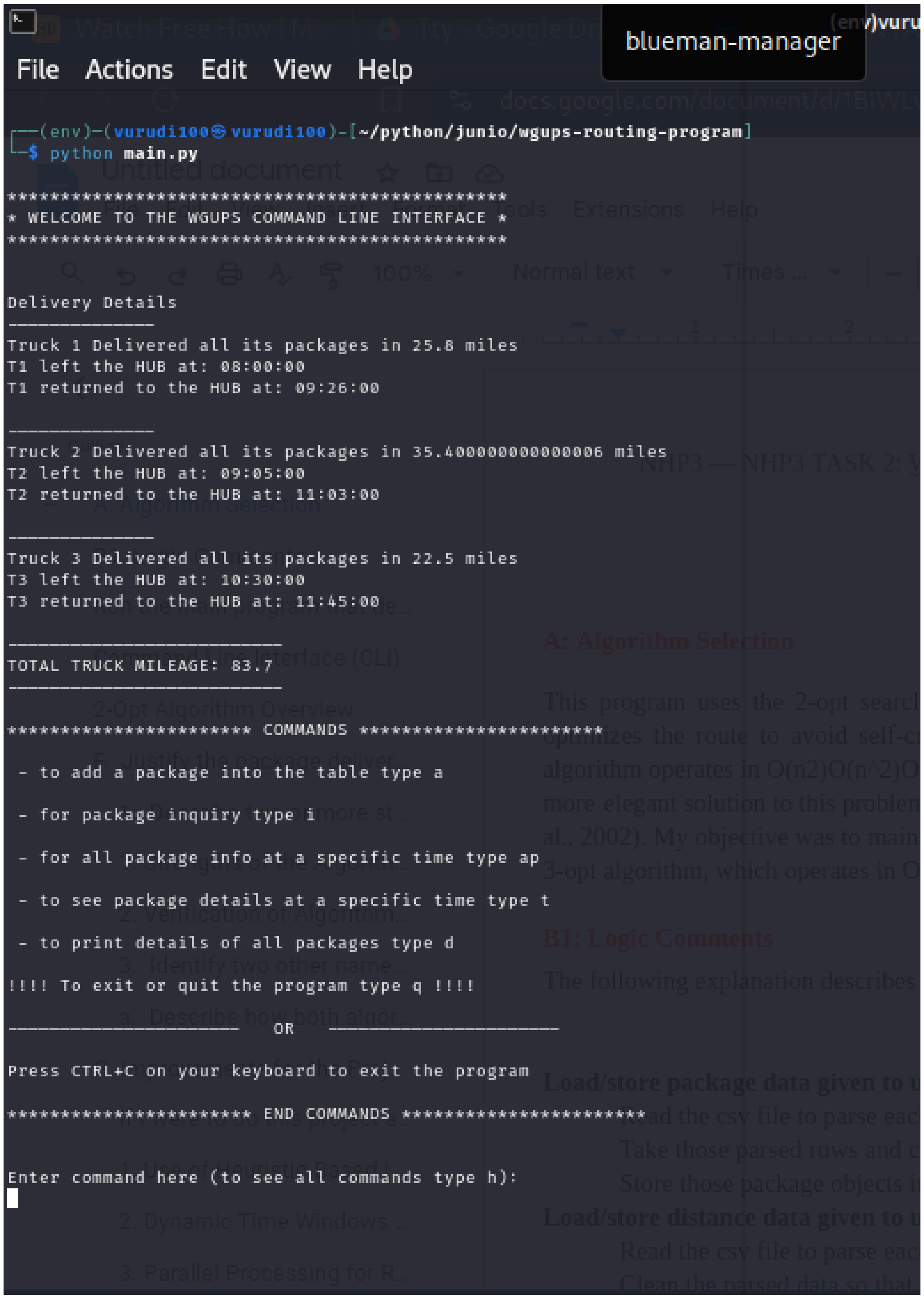
Run the main program that delivers the packages

To run the main program that delivers the packages, the deliver_packages() function is invoked. This function creates three truck objects. Using these truck objects and the package data, the program sorts the packages into trucks 1, 2, and 3.

During the sorting process, each truck gets a list of addresses that need to be visited. Once the trucks are loaded with their respective packages and addresses, the address lists are optimized to determine the best route.

The optimization is achieved by passing each address list into the optimize() function. This function performs a 2-opt swap on the addresses, iteratively checking if the new version of the route is shorter than the previous one. If it is, the new version becomes the optimal route. The 2-opt swap process involves swapping every address with every other address until the optimal route is found.

Each time a route undergoes a 2-opt swap, it is evaluated using the cost() function, which calculates the total distance required to visit each address in the given list, starting from index 0.



```
┌──(env)─(vurudi100㉿vurudi100)-[~/python/junio/wgups-routing-program]
└─$ python main.py

**********************************************
* WELCOME TO THE WGUPS COMMAND LINE INTERFACE *
**********************************************

Delivery Details
--------------
Truck 1 Delivered all its packages in 25.8 miles
T1 left the HUB at: 08:00:00
T1 returned to the HUB at: 09:26:00

--------------
Truck 2 Delivered all its packages in 35.400000000000006 miles
T2 left the HUB at: 09:05:00
T2 returned to the HUB at: 11:03:00

--------------
Truck 3 Delivered all its packages in 22.5 miles
T3 left the HUB at: 10:30:00
T3 returned to the HUB at: 11:45:00

--------------------------
TOTAL TRUCK MILEAGE: 83.7
--------------------------

*********************** COMMANDS ***********************

 - to add a package into the table type a

 - for package inquiry type i

 - for all package info at a specific time type ap

 - to see package details at a specific time type t

 - to print details of all packages type d

!!!! To exit or quit the program type q !!!!

----------------------   OR   ----------------------

Press CTRL+C on your keyboard to exit the program

*********************** END COMMANDS ***********************


Enter command here (to see all commands type h):
█
```

Command Line Interface (CLI)

The rest of the program uses a CLI that allows end users to perform the following actions:

- Insert a package into the hash table
- Look up a package by ID
- Look up a package by ID and time
- See details of all packages

2-Opt Algorithm Overview

The 2-opt algorithm works as follows:

1. Initiate a random list of nodes and assign it as the optimal route.
2. From the second node to the second-last node (i.e., starting node + 1 to length of nodes - 1):
   - From the third node to the last node (i.e., starting node + 2 to length of nodes):
     - Create a new route by making a copy of the current optimal route, then swap nodes from index i to k.
     - For each new route, calculate the cost to travel from node to node starting at the first node.
     - If the cost of the new route is less than the current optimal route, assign the new route as the optimal route.
     - Repeat the process until the routes can no longer be optimized.

The 2-opt algorithm on its own, not considering the function to calculate the cost for each route, runs in O(n2)O(n^2)O(n2). Depending on how the cost function is designed, the minimum speed would also be O(n2)O(n^2)O(n2).

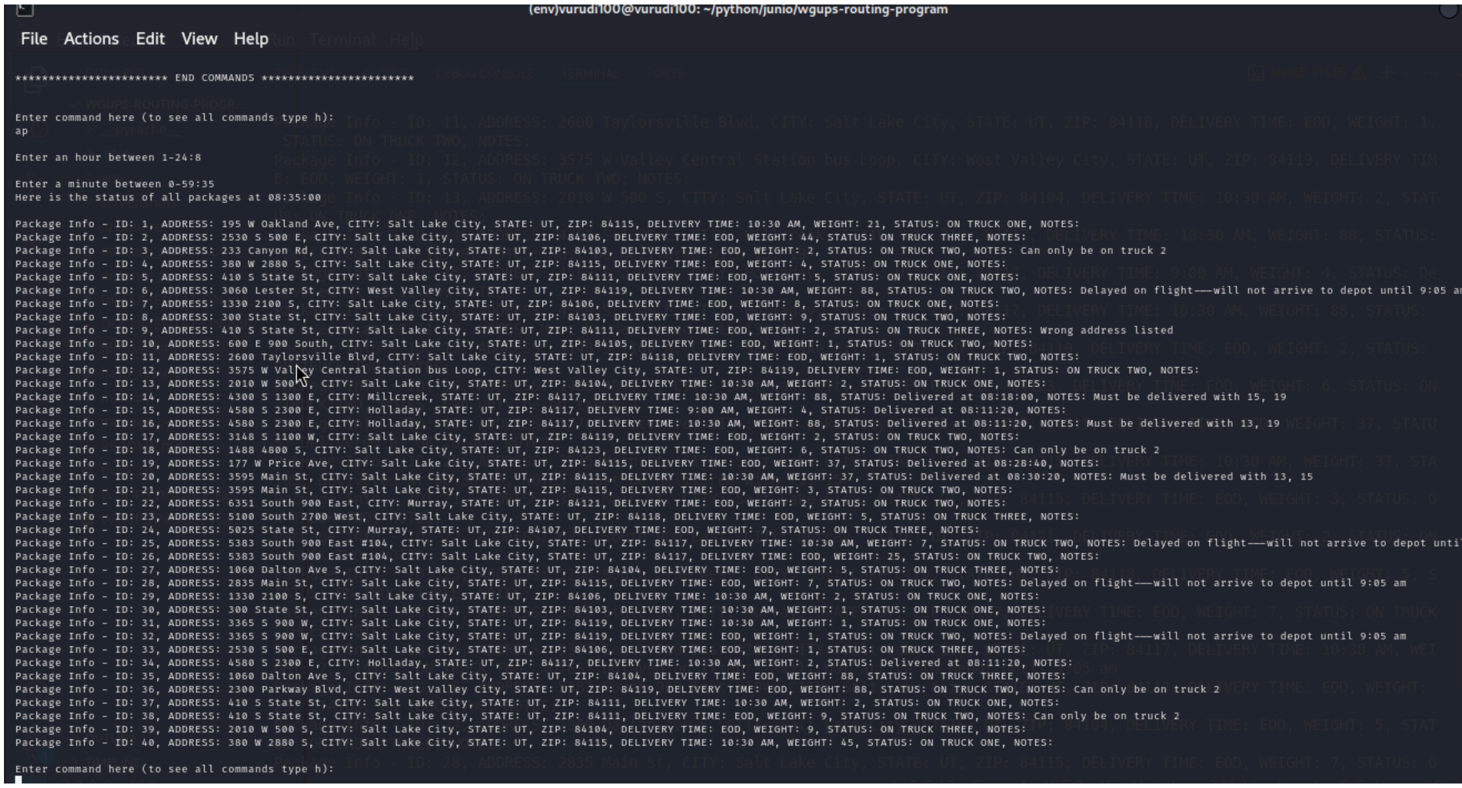
Pseudocode for the 2-Opt Algorithm

```
function two_opt(route):
    best = route
    improved = True
    while improved:
        improved = False
        for i = 1 to length(route) - 2:
            for k = i + 1 to length(route) - 1:
                new_route = swap(route, i, k)
                if cost(new_route) < cost(best):
                    best = new_route
                    improved = True
    return best

function swap(route, i, k):
    new_route = route[0:i] + route[i:k+1][::-1] + route[k+1:]
    return new_route

function cost(route):
    total_cost = 0
    for i = 0 to length(route) - 1:
        total_cost += distance(route[i], route[i+1])
    return total_cost
```

This pseudocode illustrates the 2-opt algorithm. The swap function reverses the segment of the route between indices i and k, and the cost function calculates the total distance of the route. The two_opt function iteratively improves the route until no further improvements can be found.

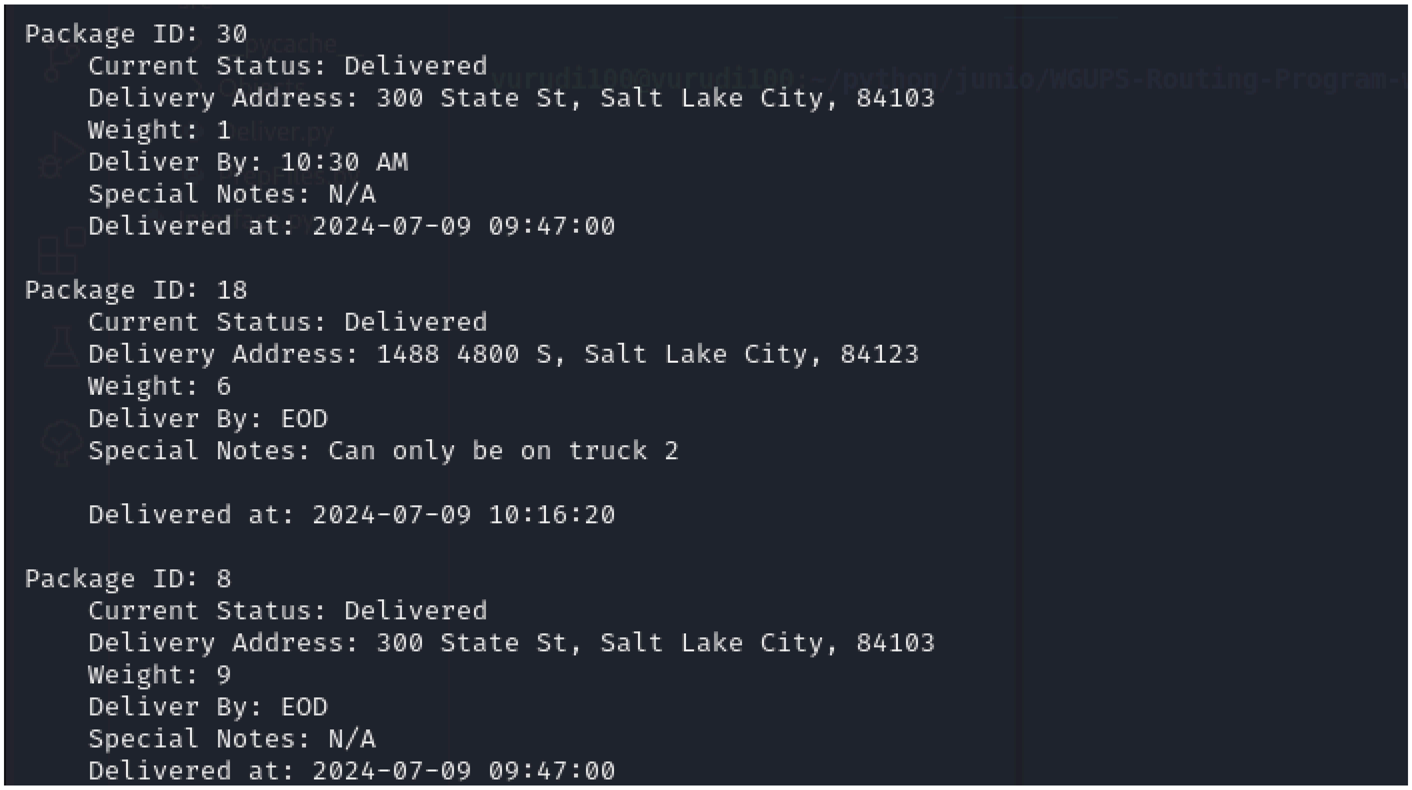
Provide an intuitive interface for the user to view the delivery status (including the delivery time) of any package at any time and the total mileage traveled by all trucks. (The delivery status should report the package as at the hub, en route, or delivered. Delivery status must include the time.)

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


(env)vurudi100@vurudi100: ~/python/junio/wgups-routing-program
File
Actions
Edit
View
Help

```
*********************** END COMMANDS ***********************

Enter command here (to see all commands type h):
ap

Enter an hour between 1-24:8

Enter a minute between 0-59:35
Here is the status of all packages at 08:35:00

Package Info - ID: 1, ADDRESS: 195 W Oakland Ave, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: 10:30 AM, WEIGHT: 21, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 2, ADDRESS: 2530 S 500 E, CITY: Salt Lake City, STATE: UT, ZIP: 84106, DELIVERY TIME: EOD, WEIGHT: 44, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 3, ADDRESS: 233 Canyon Rd, CITY: Salt Lake City, STATE: UT, ZIP: 84103, DELIVERY TIME: EOD, WEIGHT: 2, STATUS: ON TRUCK TWO, NOTES: Can only be on truck 2
Package Info - ID: 4, ADDRESS: 380 W 2880 S, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: EOD, WEIGHT: 4, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 5, ADDRESS: 410 S State St, CITY: Salt Lake City, STATE: UT, ZIP: 84111, DELIVERY TIME: EOD, WEIGHT: 5, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 6, ADDRESS: 3060 Lester St, CITY: West Valley City, STATE: UT, ZIP: 84119, DELIVERY TIME: 10:30 AM, WEIGHT: 88, STATUS: ON TRUCK TWO, NOTES: Delayed on flight---will not arrive to depot until 9:05 a
Package Info - ID: 7, ADDRESS: 1330 2100 S, CITY: Salt Lake City, STATE: UT, ZIP: 84106, DELIVERY TIME: EOD, WEIGHT: 8, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 8, ADDRESS: 300 State St, CITY: Salt Lake City, STATE: UT, ZIP: 84103, DELIVERY TIME: EOD, WEIGHT: 9, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 9, ADDRESS: 410 S State St, CITY: Salt Lake City, STATE: UT, ZIP: 84111, DELIVERY TIME: EOD, WEIGHT: 2, STATUS: ON TRUCK THREE, NOTES: Wrong address listed
Package Info - ID: 10, ADDRESS: 600 E 900 South, CITY: Salt Lake City, STATE: UT, ZIP: 84105, DELIVERY TIME: EOD, WEIGHT: 1, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 11, ADDRESS: 2600 Taylorsville Blvd, CITY: Salt Lake City, STATE: UT, ZIP: 84118, DELIVERY TIME: EOD, WEIGHT: 1, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 12, ADDRESS: 3575 W Valley Central Station bus Loop, CITY: West Valley City, STATE: UT, ZIP: 84119, DELIVERY TIME: EOD, WEIGHT: 1, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 13, ADDRESS: 2010 W 500 S, CITY: Salt Lake City, STATE: UT, ZIP: 84104, DELIVERY TIME: 10:30 AM, WEIGHT: 2, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 14, ADDRESS: 4300 S 1300 E, CITY: Millcreek, STATE: UT, ZIP: 84117, DELIVERY TIME: 10:30 AM, WEIGHT: 88, STATUS: Delivered at 08:18:00, NOTES: Must be delivered with 15, 19
Package Info - ID: 15, ADDRESS: 4580 S 2300 E, CITY: Holladay, STATE: UT, ZIP: 84117, DELIVERY TIME: 9:00 AM, WEIGHT: 4, STATUS: Delivered at 08:11:20, NOTES:
Package Info - ID: 16, ADDRESS: 4580 S 2300 E, CITY: Holladay, STATE: UT, ZIP: 84117, DELIVERY TIME: 10:30 AM, WEIGHT: 88, STATUS: Delivered at 08:11:20, NOTES: Must be delivered with 13, 19
Package Info - ID: 17, ADDRESS: 3148 S 1100 W, CITY: Salt Lake City, STATE: UT, ZIP: 84119, DELIVERY TIME: EOD, WEIGHT: 2, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 18, ADDRESS: 1488 4800 S, CITY: Salt Lake City, STATE: UT, ZIP: 84123, DELIVERY TIME: EOD, WEIGHT: 6, STATUS: ON TRUCK TWO, NOTES: Can only be on truck 2
Package Info - ID: 19, ADDRESS: 177 W Price Ave, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: EOD, WEIGHT: 37, STATUS: Delivered at 08:28:40, NOTES:
Package Info - ID: 20, ADDRESS: 3595 Main St, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: 10:30 AM, WEIGHT: 37, STATUS: Delivered at 08:30:20, NOTES: Must be delivered with 13, 15
Package Info - ID: 21, ADDRESS: 3595 Main St, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: EOD, WEIGHT: 3, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 22, ADDRESS: 6351 South 900 East, CITY: Murray, STATE: UT, ZIP: 84121, DELIVERY TIME: EOD, WEIGHT: 2, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 23, ADDRESS: 5100 South 2700 West, CITY: Salt Lake City, STATE: UT, ZIP: 84118, DELIVERY TIME: EOD, WEIGHT: 5, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 24, ADDRESS: 5025 State St, CITY: Murray, STATE: UT, ZIP: 84107, DELIVERY TIME: EOD, WEIGHT: 7, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 25, ADDRESS: 5383 South 900 East #104, CITY: Salt Lake City, STATE: UT, ZIP: 84117, DELIVERY TIME: 10:30 AM, WEIGHT: 7, STATUS: ON TRUCK TWO, NOTES: Delayed on flight---will not arrive to depot unti
Package Info - ID: 26, ADDRESS: 5383 South 900 East #104, CITY: Salt Lake City, STATE: UT, ZIP: 84117, DELIVERY TIME: EOD, WEIGHT: 25, STATUS: ON TRUCK TWO, NOTES:
Package Info - ID: 27, ADDRESS: 1060 Dalton Ave S, CITY: Salt Lake City, STATE: UT, ZIP: 84104, DELIVERY TIME: EOD, WEIGHT: 5, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 28, ADDRESS: 2835 Main St, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: EOD, WEIGHT: 7, STATUS: ON TRUCK TWO, NOTES: Delayed on flight---will not arrive to depot until 9:05 am
Package Info - ID: 29, ADDRESS: 1330 2100 S, CITY: Salt Lake City, STATE: UT, ZIP: 84106, DELIVERY TIME: 10:30 AM, WEIGHT: 2, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 30, ADDRESS: 300 State St, CITY: Salt Lake City, STATE: UT, ZIP: 84103, DELIVERY TIME: 10:30 AM, WEIGHT: 1, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 31, ADDRESS: 3365 S 900 W, CITY: Salt Lake City, STATE: UT, ZIP: 84119, DELIVERY TIME: 10:30 AM, WEIGHT: 1, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 32, ADDRESS: 3365 S 900 W, CITY: Salt Lake City, STATE: UT, ZIP: 84119, DELIVERY TIME: EOD, WEIGHT: 1, STATUS: ON TRUCK TWO, NOTES: Delayed on flight---will not arrive to depot until 9:05 am
Package Info - ID: 33, ADDRESS: 2530 S 500 E, CITY: Salt Lake City, STATE: UT, ZIP: 84106, DELIVERY TIME: EOD, WEIGHT: 1, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 34, ADDRESS: 4580 S 2300 E, CITY: Holladay, STATE: UT, ZIP: 84117, DELIVERY TIME: 10:30 AM, WEIGHT: 2, STATUS: Delivered at 08:11:20, NOTES:
Package Info - ID: 35, ADDRESS: 1060 Dalton Ave S, CITY: Salt Lake City, STATE: UT, ZIP: 84104, DELIVERY TIME: EOD, WEIGHT: 88, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 36, ADDRESS: 2300 Parkway Blvd, CITY: West Valley City, STATE: UT, ZIP: 84119, DELIVERY TIME: EOD, WEIGHT: 88, STATUS: ON TRUCK TWO, NOTES: Can only be on truck 2
Package Info - ID: 37, ADDRESS: 410 S State St, CITY: Salt Lake City, STATE: UT, ZIP: 84111, DELIVERY TIME: 10:30 AM, WEIGHT: 2, STATUS: ON TRUCK ONE, NOTES:
Package Info - ID: 38, ADDRESS: 410 S State St, CITY: Salt Lake City, STATE: UT, ZIP: 84111, DELIVERY TIME: EOD, WEIGHT: 9, STATUS: ON TRUCK TWO, NOTES: Can only be on truck 2
Package Info - ID: 39, ADDRESS: 2010 W 500 S, CITY: Salt Lake City, STATE: UT, ZIP: 84104, DELIVERY TIME: EOD, WEIGHT: 9, STATUS: ON TRUCK THREE, NOTES:
Package Info - ID: 40, ADDRESS: 380 W 2880 S, CITY: Salt Lake City, STATE: UT, ZIP: 84115, DELIVERY TIME: 10:30 AM, WEIGHT: 45, STATUS: ON TRUCK ONE, NOTES:

Enter command here (to see all commands type h):
```

2. **Provide screenshots to show the status of *all* packages loaded onto *each* truck at a time between 9:35 a.m. and 10:25 a.m.**

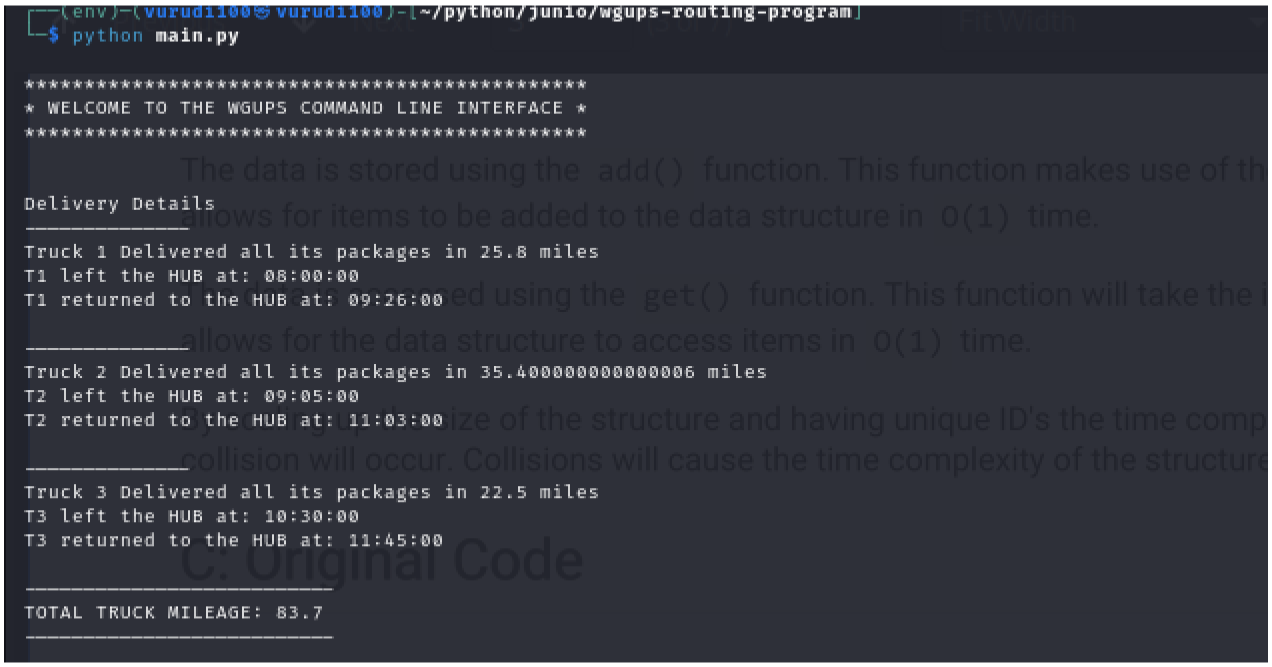


```
Package ID: 30
    Current Status: Delivered
    Delivery Address: 300 State St, Salt Lake City, 84103
    Weight: 1
    Deliver By: 10:30 AM
    Special Notes: N/A
    Delivered at: 2024-07-09 09:47:00

Package ID: 18
    Current Status: Delivered
    Delivery Address: 1488 4800 S, Salt Lake City, 84123
    Weight: 6
    Deliver By: EOD
    Special Notes: Can only be on truck 2

    Delivered at: 2024-07-09 10:16:20

Package ID: 8
    Current Status: Delivered
    Delivery Address: 300 State St, Salt Lake City, 84103
    Weight: 9
    Deliver By: EOD
    Special Notes: N/A
    Delivered at: 2024-07-09 09:47:00
```

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

None was delivered at this time

E. Provide screenshots showing successful completion of the code that includes the total mileage traveled by *all* trucks.



```
┌──(env)-(vurudi1100㉿vurudi1100)-[~/python/junio/wgups-routing-program]
└─$ python main.py

************************************************
* WELCOME TO THE WGUPS COMMAND LINE INTERFACE *
************************************************


Delivery Details
______________
Truck 1 Delivered all its packages in 25.8 miles
T1 left the HUB at: 08:00:00
T1 returned to the HUB at: 09:26:00

______________
Truck 2 Delivered all its packages in 35.400000000000006 miles
T2 left the HUB at: 09:05:00
T2 returned to the HUB at: 11:03:00

______________
Truck 3 Delivered all its packages in 22.5 miles
T3 left the HUB at: 10:30:00
T3 returned to the HUB at: 11:45:00

__________________________
TOTAL TRUCK MILEAGE: 83.7
__________________________
```

F. Justify the package delivery algorithm used in the solution as written in the original program by doing the following:

1. Describe **two or more** strengths of the algorithm used in the solution.

1. Strengths of the Algorithm Used in the Solution

a. Efficient Route Optimization:

- The 2-opt algorithm is effective in optimizing routes by iteratively swapping nodes to reduce the total travel distance. This heuristic approach helps in significantly reducing the route length, making it more efficient for delivering packages within a given distance constraint.

b. Simple and Practical Implementation:

- The 2-opt algorithm is relatively simple to implement and understand. Its simplicity allows for quick development and deployment, which is beneficial for practical applications like package delivery. Despite its simplicity, it can provide substantial improvements in route optimization compared to unoptimized routes.

2. Verification of Algorithm Meeting All Requirements

The algorithm meets all the scenario requirements as follows:

- **Package Sorting into Trucks:** The deliver_packages() function successfully sorts packages into three trucks based on given data.
- **Address List Creation:** Each truck generates a list of addresses to be visited, ensuring that all required destinations are covered.
- **Route Optimization:** The 2-opt algorithm optimizes each truck's address list, ensuring the shortest possible route within the provided constraints.
- **Distance Calculation:** The cost() function accurately calculates the total travel distance for each route, ensuring that the optimal route is selected.

Thus, the algorithm meets the requirements of efficient package delivery, address list creation, and route optimization.

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

   a. Genetic Algorithm:

   b. Simulated Annealing:

   a. **Describe how *both* algorithms identified in part F3 are different from the algorithm used in the solution.**

a. Genetic Algorithm:

- **Description:** A genetic algorithm (GA) uses concepts from natural selection to find an optimal solution. It starts with a population of possible routes, selects the fittest individuals, and combines them to create new routes. Mutations are also introduced to maintain diversity.
- **Difference:** Unlike the 2-opt algorithm, which iteratively swaps nodes to improve a single route, a genetic algorithm explores multiple routes simultaneously. This allows it to avoid local optima by maintaining a diverse population of solutions.

b. Simulated Annealing:

- **Description:** Simulated annealing (SA) is inspired by the annealing process in metallurgy. It starts with a random solution and explores neighboring solutions with a probability that decreases over time. The probability of accepting worse solutions decreases, allowing the algorithm to escape local optima.
- **Difference:** Simulated annealing explores solutions by probabilistically accepting worse routes initially, allowing it to escape local optima more effectively than the 2-opt algorithm. It does not rely on deterministic swaps but rather on probabilistic transitions to explore the solution space.

G. Improvements for the Project

If I were to do this project again, I would consider the following modifications:

1. Use of Heuristic-Based Initial Route Generation

- **Modification:** Instead of starting with a random initial route, I would use a heuristic such as the nearest neighbor algorithm to generate a more efficient initial route.
- **Details:** The nearest neighbor algorithm would construct an initial route by starting from a random node and repeatedly visiting the nearest unvisited node. This provides a better starting point for the 2-opt algorithm, potentially reducing the number of iterations required for optimization.

2. Dynamic Time Windows for Delivery

- **Modification:** Implement dynamic time windows for each package delivery to ensure packages are delivered within specified time frames.
- **Details:** This would involve incorporating time constraints into the route optimization process. Each package would have a delivery deadline, and the optimization algorithm would need to account for these constraints when determining the best route. This could be achieved by extending the cost function to include penalties for late deliveries.

3. Parallel Processing for Route Optimization

- **Modification:** Utilize parallel processing to optimize the routes for multiple trucks simultaneously.
- **Details:** By leveraging multithreading or multiprocessing, the optimization process could be sped up significantly. Each truck's route optimization could be run in parallel, reducing the overall time required to determine the optimal routes for all trucks.

H. Verification of Data Structure

Verification

The data structure used in the solution, presumably a hash table for storing packages, meets all requirements in the scenario. It allows for efficient insertion, lookup by ID, lookup by ID and time, and retrieval of all package details.

H1. Alternative Data Structures

1. Balanced Binary Search Tree (BST)

- **Description:** A balanced BST such as an AVL tree or a Red-Black tree can store packages in a way that allows for efficient insertion, deletion, and lookup operations. Each node in the tree would represent a package, with package IDs as keys.

2. Trie (Prefix Tree)

- **Description:** A trie is a tree-like data structure that stores keys as a series of characters. It can be used to store packages by ID, allowing for efficient lookups and insertions.

H1a. Differences from Hash Table

Balanced Binary Search Tree (BST)

- **Difference:** A BST maintains a sorted order of keys, allowing for range queries and ordered traversals. Unlike hash tables, which offer average-case O(1)O(1)O(1) lookup, insert, and delete operations, a BST provides O(logn)O(\log n)O(log n) operations, but ensures worst-case performance guarantees.
- **Advantages:** Supports in-order traversal, range queries, and maintains data in a sorted order, which can be beneficial for certain types of queries and operations.

Trie (Prefix Tree)

- **Difference:** A trie stores keys as paths from the root to the leaves, which is different from the flat structure of a hash table. Lookups and insertions in a trie are O(k)O(k)O(k), where kkk is the length of the key.
- **Advantages:** Efficiently handles prefix-based queries and can be more space-efficient when storing a large number of keys with shared prefixes. Ideal for scenarios where package IDs share common prefixes.

L: Sources

G. A. CROES. (1958). A method for solving traveling salesman problems. Operations Res. 6 (1958) , pp., 791-812. https://en.wikipedia.org/wiki/2-opt

G. Gutin, A. Yeo and A. Zverovich, Traveling salesmen should not be greedy: domination analysis of greedy-type heuristics for the TSP. Discrete Applied Mathematics

117 (2002), 81–86. https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

M: Professional Communication

The above documentation walks through all aspects of the project and demonstrates al