Big O Analysis

Upon opening up the program, the initial method that reads in all of the data from the first text file is called to occupy the database with the restaurant distances and menu items. This initial method that reads both menu items, and distances, runs at O (n2) time due to the fact that each restaurant has to be accessed (n) as well as the fact that the distances to the other restaurants is read in as well (n within n, or n2). Proceeding the distances, the menu items are also capable of O(n) performance, but they are apart from the distances, so the execution time is only dependent on the total of restaurants, just like the distances (n within n, or n2). Updating the restaurants from the new restaurants has a similar execution time since it is performing the same task. Loading menu items in to the table view on the Menu Items tab will perform in O(n2) time due to the fact that all of the restaurants must have their menu items loaded (n) in addition to the number of menu items for each restaurant (n within n, or n2). The Distances tab shares the same complexity as the Menu Items tab due to the fact that all of the restaurants must be loaded (n), as well as the distances to the other restaurants (n within n, or n2).

The first trip option, being the Visit all Available Restaurants choice, runs the getFullTrip method that first checks if the query to start the trip has successfully run. After confirming that the query has been executed, the program then runs a while loop until the restaurants vector is empty. This is a vector that contains all of the restaurants that are going to be attended on this trip. A query is run that puts all of the restaurants in order of ascending distances and then the restaurants are read and stored. A second loop is input to check and make sure that the restaurants that have been visited will not output any longer. This while loop within a while loop executes at O(n2) time. After leaving this while loop the method removeFromVec is called, and restaurants are removed from the initial vector that was passed in that contained all of the restaurants. By removing the restaurants, the outer while loop is slowly brought to an end, and the loop ends, returning a queue that is created from the restaurants that were being removed from the initial vector, thus returning an ordered vector with the correct shortest route for the trip.

If the query is executable (1)

Remove a restaurant from the vector of restaurants (n)

While restaurants is not empty (n)

If the query is executable (n) (1)

While restaurants have been visited in the queue (n) (n)

Restaurant equals current restaurant (n) (n) (1)

Push the value of the restaurant in to a vector (n) (n)

Remove a restaurant from the vector of restaurants (n) (1)

Return the queue (1)

Highest complexity is O(n2)

The second trip option, being the Plan Trip With Starting Point and # of Restaurants choice, runs the getCustomTrip method that checks if the query was successful; then the method isInRestaurants checks if the current restaurant is in the vector of trip restaurants. This performs at constant time. A loop runs and keeps continuously checking for a restaurant name that is in the vector of restaurants to be visited or until the end of the vector is reached. This method performs in O(n) time. The restaurant then removes this initial value out of the vector, and then checks if the vector is empty in a while loop. This runs in O(n) time. After entering the while loop the program checks to see if the query is executable if it is, the program then checks for the restaurant to be in the vector of restaurants on the trip. This execution run in O(n) within O(n), or O(n2). The next query is executed, and the second while loop is entered. Within this while loop an if statement checks again to see if the restaurant is in the vector of restaurants, this runs in O(n) within O(n) within O(n), or O(n3). After finishing the inner loop, the program stores a pair in to a queue by pushing, and then removes the restaurant from the vector. The method returns a queue of all of the restaurants in order.

If the query is executable (1)

If a restaurant isn’t in the vector of restaurants for the trip (n)

While the next query is executable (n)

If the restaurants is in the vector of restaurants (n) (n)

While the vector of restaurants is not empty (n)

If the next query is executable (n) (1)

If the restaurant is not in the vector of restaurants (n) (n)

While the next query is available (n) (n)

If a restaurant is in the vector of restaurants (n) (n) (n)

Push the restaurant in to a queue for the trip (n) (n)

Remove the pushed restaurant from the input vector (n) (n)

Return the queue (1)

Highest complexity is O(n3)

The third trip option, being the Choose Exactly Which Restaurants to Visit, runs the getStartTrip method that checks if the query is executable. If so then the method is entered and the first restaurant is pushed in to the trip queue. This runs in constant time. After the first restaurant is pushed to the queue, it is removed from the vector of restaurants on the trip. The method then checks to see if the vector of restaurants is empty and makes sure that the number of restaurants left to visit is less than 0. This executes in O(n) time. The method then checks if the query executed, and if so the program queries the next query. The program then if a restaurant has been visited in a while loop. This is a loop that is within another loop, or O(n2) complexity.

After the second loop exits, the program pushes the restaurant, and then removes it from the current trip vector. The end of the program returns the queue for the vector.

If the query is executable (1)

Push the current restaurant to the queue (1)

While the vector of restaurants is not empty (n)

If the query is executable (n) (1)

While a certain restaurant has not been visited (n) (n)

Store the restaurant from the query (n) (n) (1)

Push the restaurant in to the queue (n) (1)

Remove the restaurant from the trip vector (n) (1)

Return the queue (1)

Highest complexity is O(n2)

Our project includes an STL list in our CustomTripSelectorWidget as one of our data structures. Another data structure used is a queue, which we use to store all of the restaurants locations for our trips. The third data structure we used was a vector, which we used for reading in all of the restaurant names that are to be visited on the trips.