# **User Side of Project completed by “Rohail Taha”.**

I implemented the user side of our café app. I implemented the following functionalities:

* Authenticating the user before login.
* Sign up a new user
* View café menu and place items in the cart.
* Remove items from cart, empty cart, view items of cart.
* Confirm order after adding items to cart.
* User can see all his past orders, see the order status.

Data Structures usedare **Hash Table with BST chaining for collisions, Linked Lists**.

**Brief Details of each class follows:**

**1. UserInterface:** Has all the user interface. This is a console app (no GUI), so the interface is implemented by printed text.

**2. UserTable:** Represents a hash table data structure for storing all the users registered in the app.It has a **User** array of length 26 that represents the hash table.

**3. User:** Represents user that logged into the app. When app starts, all users from file “users.csv” are read and loaded into the hash table data structure.

**4. Cart:** Has cart items for an order. Each cart object belongs to a specific café.. This class contains a linked list of all items that are currently added to the cart.

**5. CartItem:** Represents an item that is added to the cart.

**6. Order:** Has **OrderDetails**

**7. OrderDetails:** Has all the details of a particular order.

**8. PastOrders:** AllPast orders made by a particular user. This class has linked list of all the **pastOrder** in a descending sorted order (most recent order is added to the head)

**9. PastOrder:** Past order of a particular user. Each past order has linked list of all the items (**cartItem**) that were ordered.

**10. CommonFunctionalities:** Contains utility methods that return a unique id.

**Time Complexities and Aysmptotic Analysis**

**1. Adding a user to hash table (excluding reading from file):**  
**Worst Case: O(n)**, All the usernames are hashed to the same index. The BST tree at that index has height n where n are the number of users. New user is added as a leaf in that tree, so n traversals. **Average Case** **O(logm)**, On average each index of User array has m users where m = n / 26 (26 are the number of buckets). The BST with m nodes has height logm, so traversal takes logm time. **Best Case O(1)**, The user is added at an index that is empty.

**2. Printing all the users in the hash table:**  
**O(n)** as we have to traverse n users/nodes.

**3. Get a user from the hash table:**  
**Worst Case O(n),** All the usernames are hashed to the same index. The BST tree at that index has height n. The user to be fetched is a leaf node in that tree, so we have to traverse n nodes. **Average Case** **O(logm)**, On average each index of User array has m users where m = n / 26 (26 are the number of indexes). The BST with m nodes has height logm, so traversal takes logm time. **Best Case O(1)**, The user to be fetched is a head of the BST tree at the hashed index.

**4. Removing a user from hash table (excluding removing from file):**  
**Worst Case O(n)**, All the usernames are hashed to the same index. The BST tree at that index has height n. The user to be removed is a leaf node in that tree, so we have to traverse n nodes. **Average Case** **O(logm)**, On average each index of User array has m users where m = n / 26 (26 are the number of indexes). The BST with m nodes has height logm, so traversal takes logm time. **Best Case O(1)**, The user is removed from an index that has only the root of the BST.

**5. Adding an item to cart:**  
**O(1)** in all cases as the item is added to the end of linked list of cart items using the tail pointer.

**6. Removing an item from cart:**  
**Worst Case: O(n)**, The item to be removed is at the end of the cart so n traversals **Average Case** **O(n)**, If we do the maths the average of the time taken for the item to be found at every place is asymptotically bound by the the set O(n). **Best Case O(1)**, The cart item at head is to be removed.

**7. Print all cart items:**  
**O(n)** for all cases as there are n traversals.

**8.** **Search for a cart item in the list:**  
**Worst Case O(n)**, The item to search is at the end of the cart so n traversals. **Average Case** **O(n)**, If we do the maths the average of the time taken for the item to be found at every place is a function that is asymptotically bound by the the set O(n). **Best Case O(1)**, The cart item is at head.

**9. Check if the cart is empty:**  
**O(1)** as it only has to check whether the head pointer is null or not.

**10. Empty cart:**  
**O(1)** as it involves setting the head and tail pointer to null.

**11. Save ordered items to the file “ordered Items.csv”:**  
**O(n)** for all cases as we traverse through each cart Item and append it to the file “orderedItems.csv”.

**11. Add all those Items to a “past” order that belonged to that order:**  
**O(n)** for all as we loop through n cart items and add each to the past order.

**12. Add a past order to pastOrders list**  
**O(1)** for all cases as the past order is always added to the head.

**13. Print all the past orders**  
**O(n \* m)** where n = no of past orders and m = average number of items in each past order.

**Hash Table Representation**

**= User**

0

1

2

3

4

0

0

0

0

0

0

0

6

5

26

……………………………………………….

**Each = User has linked list of past orders represented as:**

Past order 1

Past order 2

Past order 3

Past order 4

Past order 2

Past order 2

Past order 2

Cart item 1

Cart item 2

Cart item 3

Cart item 4

Cart item 1

Cart item 2

Cart item 3

Cart item 1

Cart item 2

Cart item 1

Cart item 2

Cart item 3

Cart item 1

Cart item 2

Cart item 3

Cart item 4

**A cart has linked list of cart items for an order**