Final Project

Data Structures

Project Name: “Shopping Cart Tracker”

Group Members : Nathan Darien Tanner, Abdullah Akmal Sutoyo, Carlo Nathanael Bessie

Student ID: 2602225656, 2602236685, 2602239320

Class: L2BC

Table of Contents

Cover Page

Contents

Abstract

A. Background

I. Problem Definition

II. Main Objective

B. Data Structures

I. ArrayList

II. LinkedList

III. HashSet

C. Solution

I. Implementation of data structure

II. Benchmark results

III. Time complexity analysis

IV. Space complexity analysis

D. Conclusion

E. Appendix

Abstract

The task that was assigned to us to complete this project is to implement various data structures to a working program and make a series of methods for us to benchmark. We then need to analyze the time and space complexity of each method in the code to be able to compare it against each data structure that we use. The purpose of this report is to show the result of our testing and the comparison of every data structure to know which data structure is the most suitable for our program.

A. Background

a. Problem Definition

We chose to create a shopping cart tracker that could handle a huge amount of items for this project. The reason is because people who usually shop in minimarkets or supermarkets experience difficulty in organizing items that want to be purchased and enlist in their shopping cart, especially if there is a large amount of items that need to be organized. Looking at the items one by one inside the shopping cart is not a very efficient way to handle the items inside the shopping cart. Hence, we created a shopping cart tracker that could help alleviate this problem.

b. Main Objective

In our shopping cart tracker, we have several operations that can be used by the user to help track whatever is inside their shopping cart. The first operation is to insert an item inside the shopping cart by entering the item description from the name, price, and quantity of the item. The user can then manage the items inside the shopping cart by increasing the quantity or decreasing the quantity of items that match the user input. If the user decides to remove an item completely from the shopping cart, There is an operation that the user can utilize by entering the name and price of the item. There are also operations to clear the cart completely and calculate the total price of all the items that are available in the cart. Users can search for a specific item to check whether the item exists in the cart or not. The items in the shopping cart will then be organized based on the name and the price of the item. Items that have the same name and price will then be grouped and quantified into a list for people to be able to manage their items with ease.

B. Data Structures

The data structures that we chose are ArrayList, LinkedList and HashSet.

a. **ArrayList**

Advantages:

1. Quick access: ArrayList allows you to directly find items using their index, which means you can easily grab the item you want from your shopping cart without much effort.
2. Easy to go through: Going through the items in an ArrayList is faster because you don't have to search through different parts of your cart. It's like having all your items laid out neatly in front of you.
3. Convenient changes: Adding or placing new items at the end of your ArrayList is efficient. You can easily append or insert items without rearranging everything in your cart.

Disadvantages:

1. Tricky insertions and removals: If you need to add or remove items at specific positions in your shopping cart, it can be more complicated and time-consuming with an ArrayList. You might need to shift items around to make space for the new ones or close the gaps left by the removed items.
2. Challenging with a lot of items: When your shopping cart becomes really full, resizing it can be a hassle. It's like trying to expand your cart to accommodate more items, which takes extra time and effort, especially if you have many things to add.

b. **LinkedList**

Advantages:

1. Easy adding and removing: When it comes to adding or removing items in a LinkedList, it's quite efficient. You can quickly insert or delete items by simply adjusting the connections between the nodes, just like rearranging items in your shopping cart.
2. Flexible size: LinkedList can change its size dynamically, meaning it can grow or shrink as needed without having to reserve a fixed amount of space. It's like having a shopping cart that expands or contracts based on how many items you have.

3. Efficient memory usage: LinkedList doesn't require contiguous memory. It can allocate memory more flexibly, which can help reduce memory waste and make the most of the available space.

Disadvantages:

1. Slower access to specific items: Unlike ArrayList, LinkedList doesn't allow you to directly access elements by their index. If you want to find a specific item, you have to go through the list from the beginning until you reach the desired position. This can take more time, especially if you have a large list of items in your shopping cart.
2. Increased memory usage: LinkedList needs extra memory to store the connections between nodes. This additional memory overhead can be higher compared to using an ArrayList in your shopping cart.

c. HashSet

Advantages:

1. Quick searching: When you use a HashSet, finding whether an item exists in the set or not is super fast. It's like searching for an item in your shopping bag and getting an instant answer.
2. Efficient uniqueness: HashSet ensures that every item in the set is unique. This is useful when you want to make sure you don't have any duplicates in your shopping list. It keeps everything organized and prevents you from accidentally buying the same item twice.
3. No specific order: Items in a HashSet are not arranged in any particular order. This can be beneficial when the order of items doesn't matter much for your shopping needs. You can focus on the items themselves rather than their arrangement.

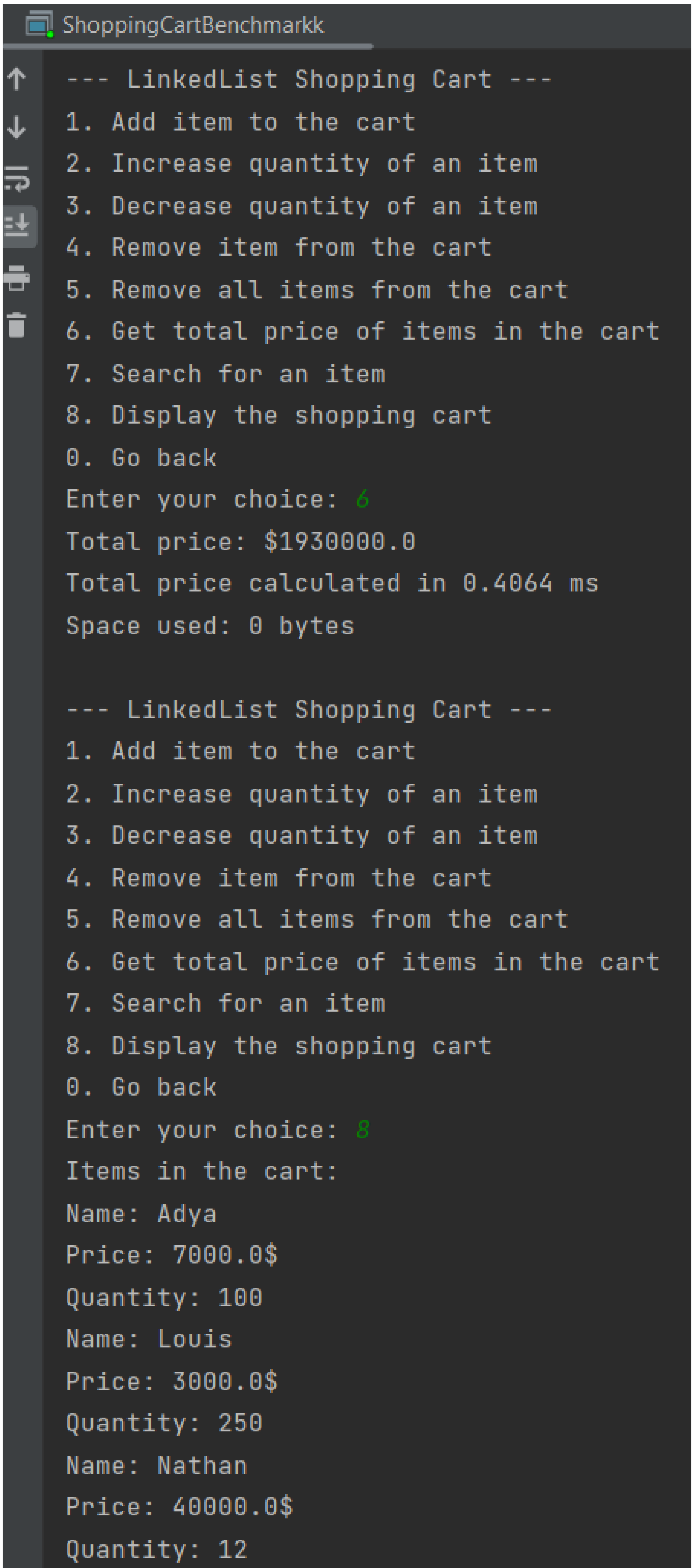
Disadvantages:

1. No direct index access: Unlike ArrayList, HashSet doesn't provide direct access to items based on their index or position. If you need to specifically retrieve an item by its position, it can be more challenging with a HashSet.
2. Unpredictable order: The order in which you iterate through items in a HashSet is not predictable. It may change over time, which can be a disadvantage if you rely on a specific order for your shopping tasks.

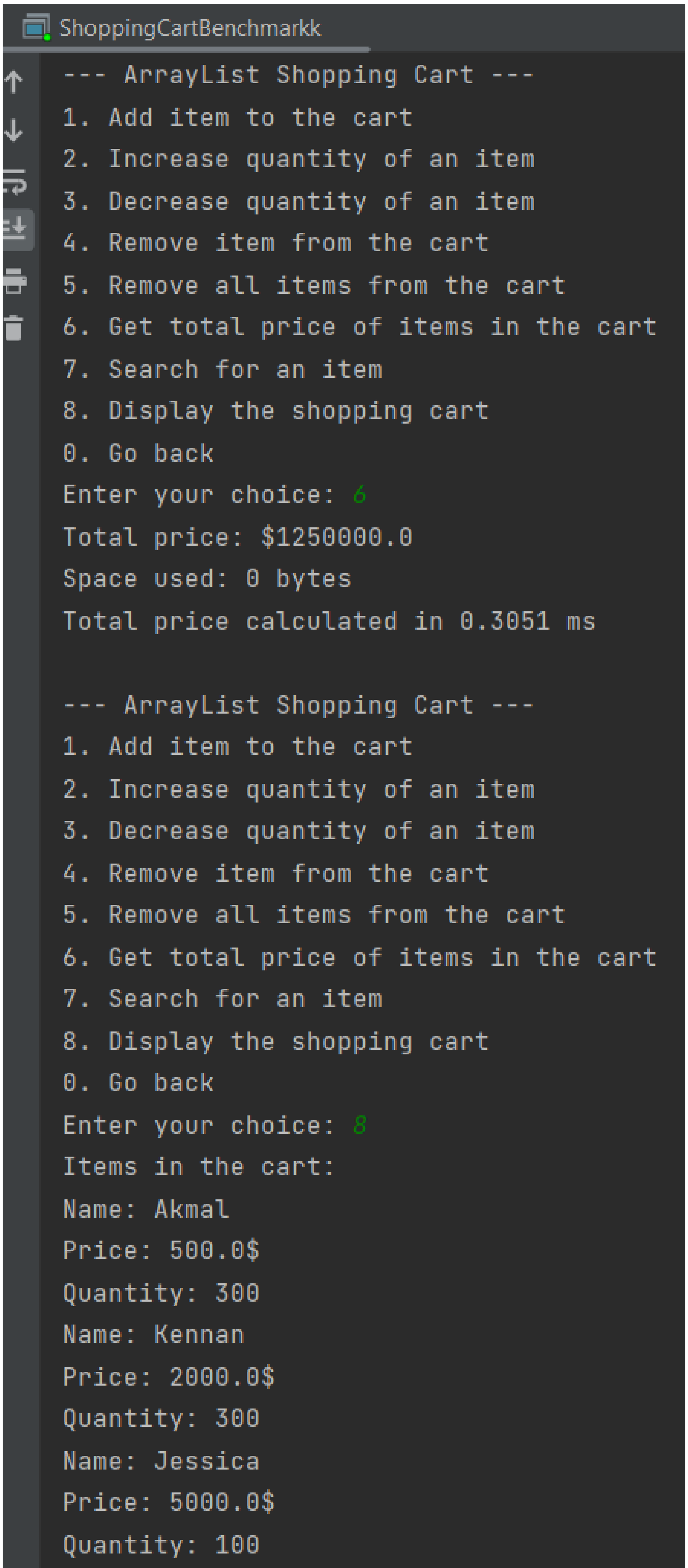
3. Hashing overhead: HashSet uses a hashing mechanism to store items, which requires additional computational resources. This can introduce some overhead compared to simpler data structures.

C. Solution

a. Implementation of data structures

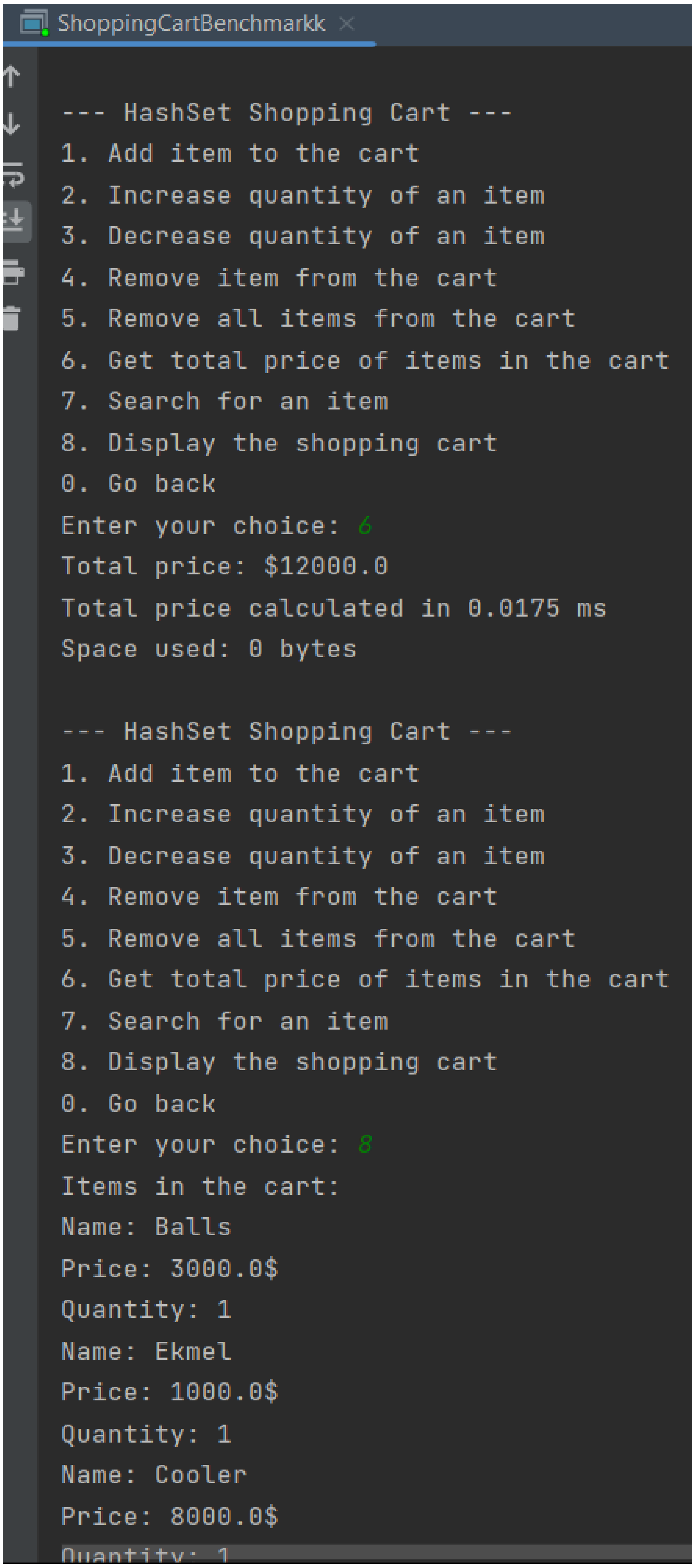


```
--- LinkedList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 6
Total price: $1930000.0
Total price calculated in 0.4064 ms
Space used: 0 bytes

--- LinkedList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 8
Items in the cart:
Name: Adya
Price: 7000.0$
Quantity: 100
Name: Louis
Price: 3000.0$
Quantity: 250
Name: Nathan
Price: 40000.0$
Quantity: 12
```



ShoppingCartBenchmarkk

```
--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 6
Total price: $1250000.0
Space used: 0 bytes
Total price calculated in 0.3051 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 8
Items in the cart:
Name: Akmal
Price: 500.0$
Quantity: 300
Name: Kennan
Price: 2000.0$
Quantity: 300
Name: Jessica
Price: 5000.0$
Quantity: 100
```

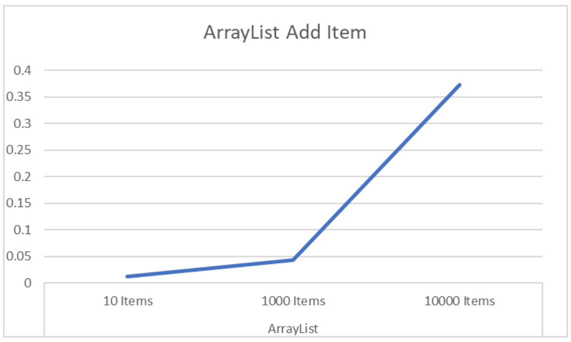


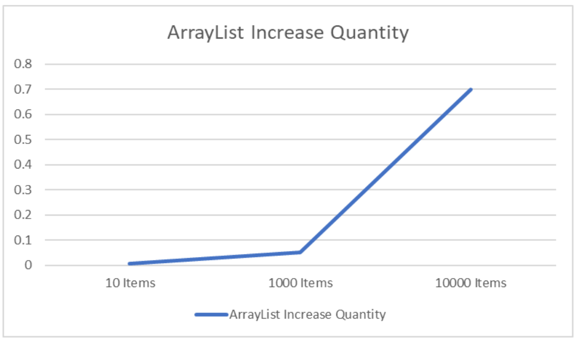
```

--- HashSet Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 6
Total price: $12000.0
Total price calculated in 0.0175 ms
Space used: 0 bytes

--- HashSet Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 8
Items in the cart:
Name: Balls
Price: 3000.0$
Quantity: 1
Name: Ekmel
Price: 1000.0$
Quantity: 1
Name: Cooler
Price: 8000.0$
Quantity: 1
```

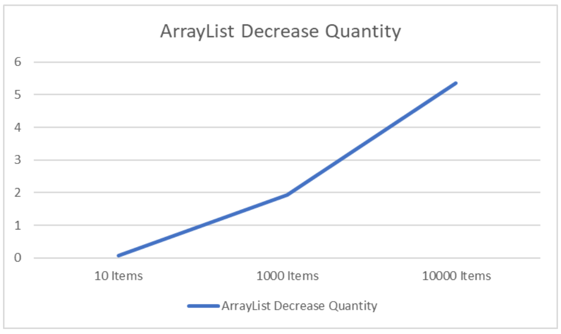
b. Benchmark results

The task that was assigned to us to complete this project is to implement various data structures to a working program and make a series of methods for us to benchmark. We then need to analyze the time and space complexity of each method in the code to be able to compare it against each data structure that we use. The purpose of this report is to show the result of our testing and the comparison of every data structure to know which data structure is the most suitable for our program

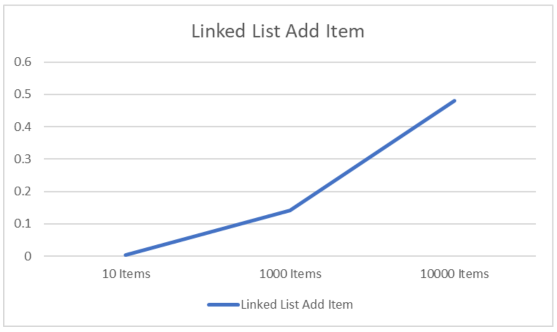
ArrayList Time Complexity Graph

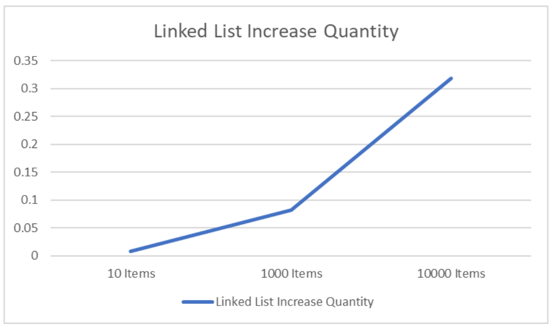


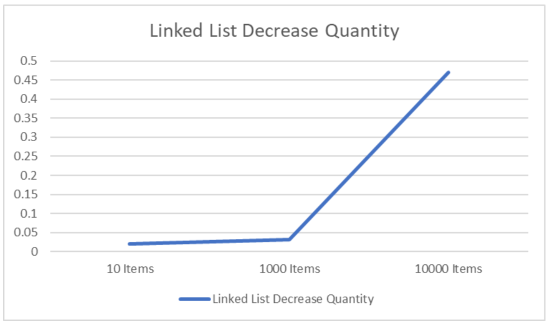




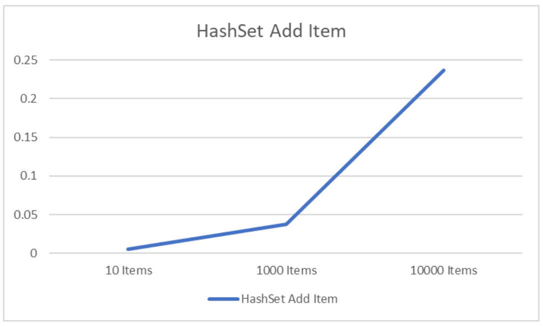
LinkedList Time Complexity Graph

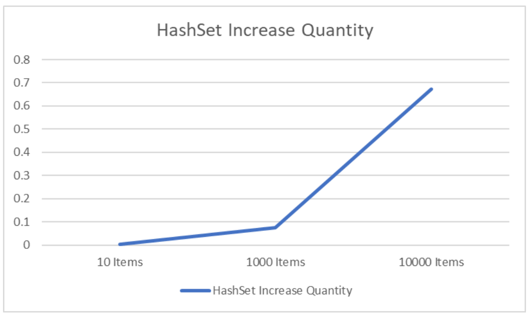


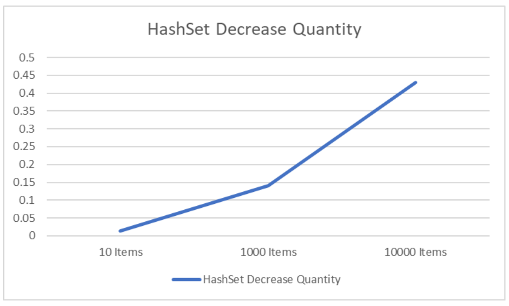




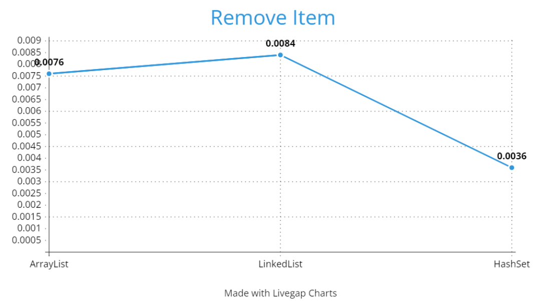
HashSet Time Complexity Graph

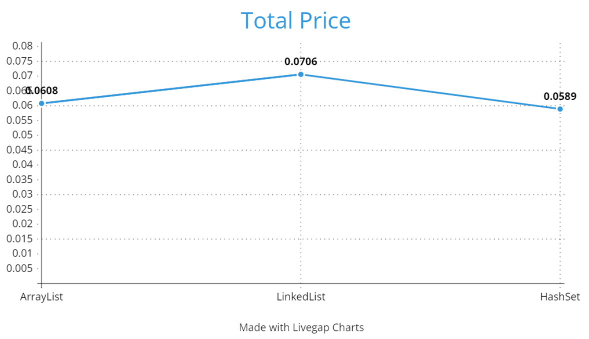


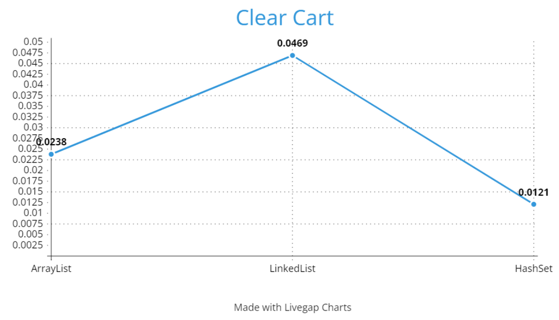


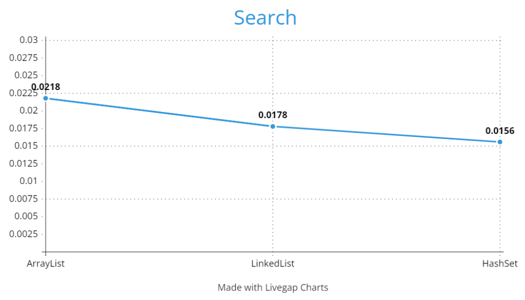


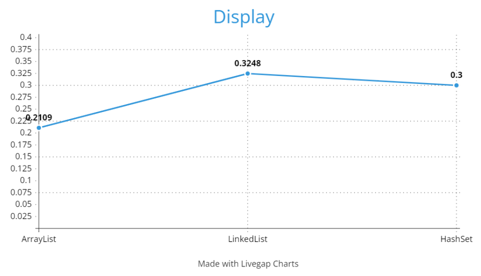
Space Complexity Graph











c. Time complexity analysis

1. Adding items to a cart:

For all data structures, this operation has a time complexity of O(n), since the loop iterates at the same amount of time as the quantity input from the user, and in each iteration, it performs a constant-time operation, which is adding the item to the data structure.

2. Increasing item quantity:

For all data structures, this operation has a time complexity of O(n* k). This is because the method needs to iterate through the ArrayList to find the item by name, which takes O(n) time complexity, and then add the item to the data structure "k" number of times, which takes O(k) time complexity.

3. Decreasing item quantity:

For all data structures, this operation has a time complexity of O(n * k). This is because the method needs to iterate through the ArrayList to find the item by name, which takes O(n) time complexity, and then delete the item to the data structure "k" number of times, which takes O(k) time complexity.

4. Remove item from cart:

For HashSet, this operation has a time complexity of O(1) because HashSet internally uses a hash table, which allows for efficient lookups and removals based on the hash values of the elements. For LinkedList and ArrayList, this operation has a time complexity of O(n), where n is the number of elements in the list. It needs to iterate over each element to remove all the elements with the same name.

5. Remove all items from cart:

For ArrayList, this operation has a time complexity of O(1) because it simply resets the size of the list to zero, effectively removing all elements. For LinkedList and HashSet, this operation has a time complexity of O(n), where n is the number of elements in the list. It needs to iterate over each element to remove all the elements.

6. Getting the total price:

For all data structures, this operation has a time complexity of O(n), since the loop iterates at the same amount of time as the amount of elements inside the data structure, and in each iteration, it performs a constant-time operation, which is adding up the prices.

7. Search for an item:

For ArrayList and LinkedList, this operation has a time complexity of O(n), where n is the number of elements in the list. In both data structures, accessing an element by index takes linear time, and since each item is accessed once, the time complexity is proportional to the number of items in the list. For HashSet, the time complexity is O(1) on average for this operation, but in the worst case, it can be O(n), where n is the number of elements in the set. This is because a HashSet uses hashing techniques to perform lookups efficiently.

8. Display the shopping cart:

For all data structures, this operation has a time complexity of O(n), since the loop iterates at the same amount of time as the amount of elements inside the data structure, and in each iteration, it performs a constant-time operation, which is printing the items in the cart.

d. Space complexity analysis

1. Adding items to a cart:

For ArrayList and LinkedList, this operation has a space complexity of O(n), as the ArrayList and LinkedList needs to store "n" number of items. The space required by both the ArrayList and LinkedList increases linearly with the parameter. However, for the HashSet, the time complexity is O(unique items), since HashSet guarantees uniqueness, duplicate items will not be stored multiple times. Therefore, if the item parameter remains the same for all iterations, the space complexity for a HashSet implementation will be O(unique items), rather than O(n).

2. Increasing item quantity:

The space complexity of this operation on all data structures is O(n), as it adds the item to the set “n" number of times. The space required by the data structures increases linearly with the quantity parameter.

3. Decreasing item quantity:

The space complexity of this operation on all data structures is O(1), as it does not create any additional data structures that scale with the input size. The space requirement remains constant regardless of the size of the cart or the quantity parameter.

4. Remove item from cart:

The space complexity is O(1) as it does not require additional memory.

5. Remove all items from the cart:

The space complexity of this operation on all data structures is O(1), as it does not require additional memory.

6. Getting the total price:

   The space complexity is O(1) as it only requires a constant amount of memory to store the variable for the total price.

7. Search for an item:

   The space complexity is O(1) as it only requires a constant amount of memory to store the variable for the total price.

8. Display the shopping cart:

   The space complexity is O(1) as it only requires a constant amount of memory to store the variables.

D. Conclusion

Based off the information we wrote above, these are the conclusion to our findings:

1. Adding items to a cart: All three data structures (ArrayList, LinkedList, HashSet) can efficiently handle adding items to a cart. However, HashSet is the most efficient because instead of having an O(n) space complexity, it has an O(unique items) space complexity, which means it loops less.
2. Increasing/decreasing item quantity: ArrayList and LinkedList are more suitable for this operation, as they allow direct access to elements by index. HashSet does not provide direct index-based access, so modifying quantities may be less convenient.
3. Removing an item from the cart: HashSet is the most efficient for removing an item with the same name, as a HashSet provides an average time complexity of O(1). On the other hand, both ArrayList and LinkedList require iterating over the elements to remove items, resulting in a linear time complexity of O(n).
4. Remove all items from the cart: ArrayList is the most efficient for removing all items in the cart, since it has an O(1) time complexity to remove all items, unlike LinkedList or HashSet.

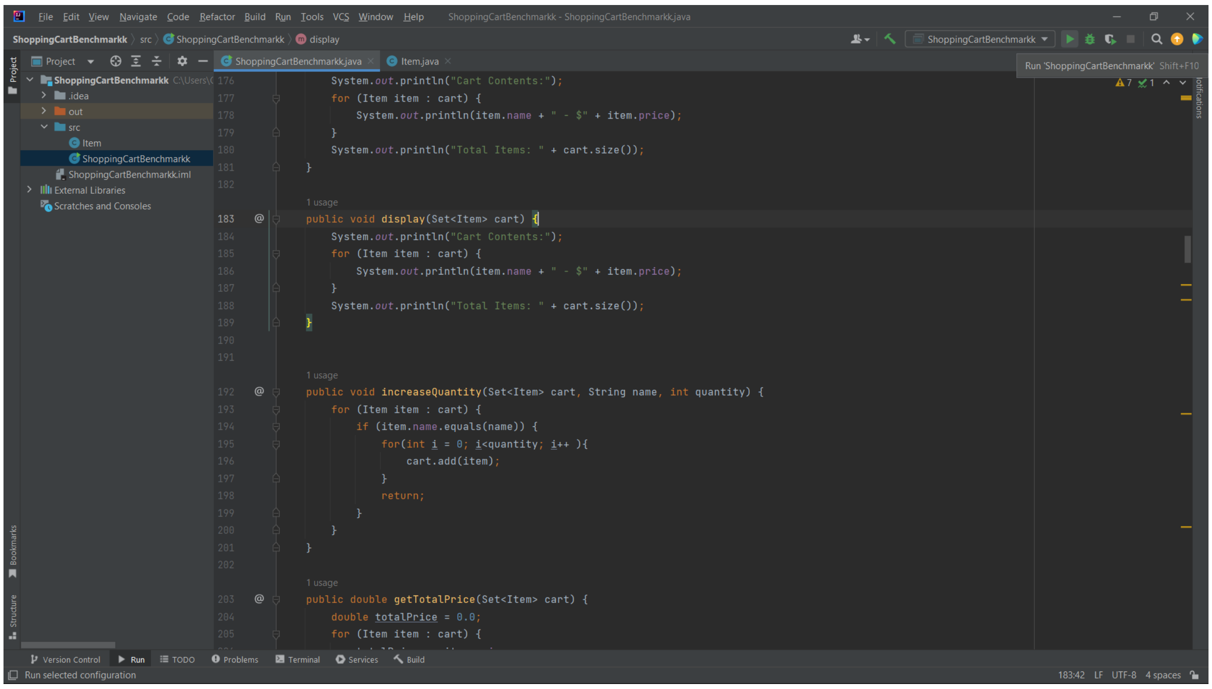
5. Getting the total price of items: All three data structures can be used for calculating the total price. The choice of data structure does not have a significant impact on this operation.
6. Searching for a specific item: HashSet is the most efficient for searching, as it provides constant-time search O(1) to determine whether an element exists in the set or not. ArrayList and LinkedList require linear-time traversal for searching, resulting in a less efficient search operation.
7. Displaying the shopping cart: ArrayList and LinkedList preserve the order of items, making them more suitable for displaying the shopping cart. HashSet does not provide a predictable order for iteration, so displaying the cart may not be in a specific order.

Based on these considerations, the best data structure for a shopping cart would likely be a HashSet. With a Hashset, you can use it for efficient searching and ensuring the uniqueness of items in the cart. This means that a HashSet has a more efficient time complexity to search for a specific item and remove an item with the same name, while also having a more efficient space complexity in the cart because it does not allow an item duplicate. However, HashSet isn't the most efficient at removing all items in the cart, and displaying the shopping cart might be a bit messy.

Appendix

Program Manual

Step 1: Download the source codes from GitHub and open it in a suitable IDE for java (preferrably IntelliJ). Navigate to the file named ‘ShoppingCartBenchmark’ and run the file by clicking the play button on the top right corner.



```
        System.out.println("Cart Contents:");
        for (Item item : cart) {
            System.out.println(item.name + " - $" + item.price);
        }
        System.out.println("Total Items: " + cart.size());
    }

    1 usage
    public void display(Set<Item> cart) {
        System.out.println("Cart Contents:");
        for (Item item : cart) {
            System.out.println(item.name + " - $" + item.price);
        }
        System.out.println("Total Items: " + cart.size());
    }


    1 usage
    public void increaseQuantity(Set<Item> cart, String name, int quantity) {
        for (Item item : cart) {
            if (item.name.equals(name)) {
                for(int i = 0; i<quantity; i++ ){
                    cart.add(item);
                }
                return;
            }
        }
    }

    1 usage
    public double getTotalPrice(Set<Item> cart) {
        double totalPrice = 0.0;
        for (Item item : cart) {
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
ShoppingCartBenchmarkk - ShoppingCartBenchmarkk.java
ShoppingCartBenchmarkk › src › ShoppingCartBenchmarkk › display
ShoppingCartBenchmarkk.java
Item.java
Run 'ShoppingCartBenchmarkk' Shift+F10
Project
ShoppingCartBenchmarkk
.idea
out
src
Item
ShoppingCartBenchmarkk
ShoppingCartBenchmarkk.iml
External Libraries
Scratches and Consoles
Version Control
Run
TODO
Problems
Terminal
Services
Build
Run selected configuration
183:42 LF UTF-8 4 spaces

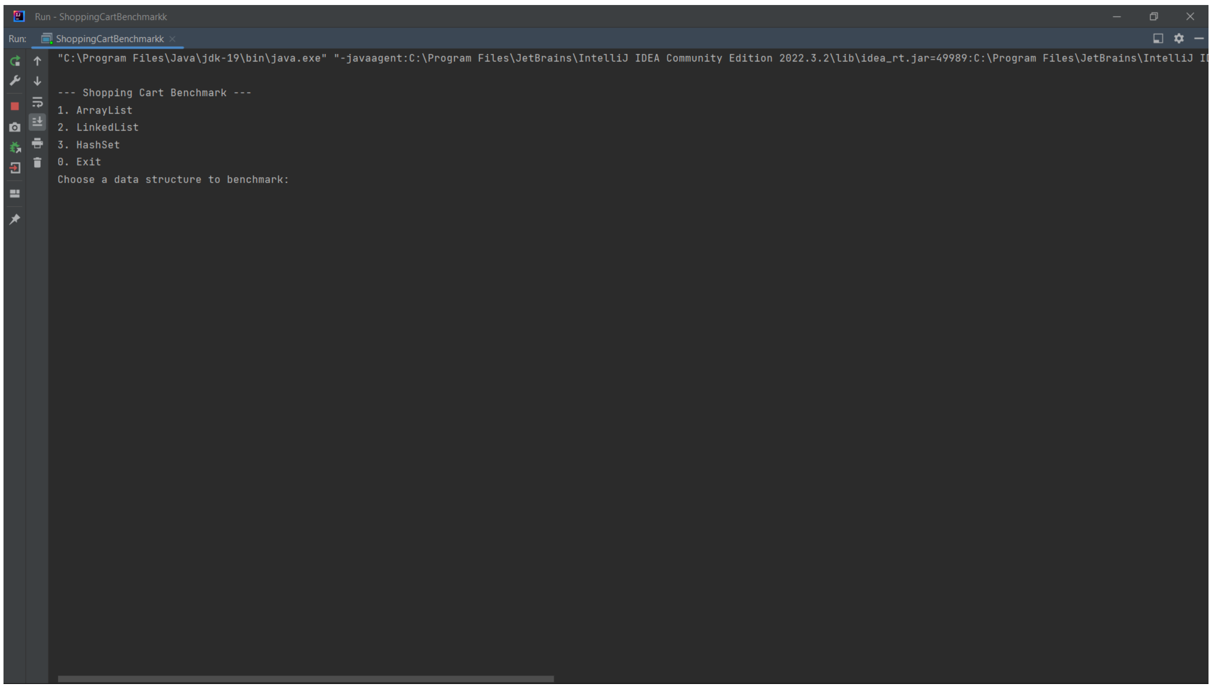
Step 2 : Once you run the file, you will be taken to the terminal where you wiil be given options of data structures:

(1). ArrayList

(2). LinkedList

(3). HashSet

These are the data structures that are available to be benchmarked by the code. Choose the data structure to test by entering the number corresponding to the data structure of choice.



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\lib\idea_rt.jar=49989:C:\Program Files\JetBrains\IntelliJ I

--- Shopping Cart Benchmark ---
1. ArrayList
2. LinkedList
3. HashSet
0. Exit
Choose a data structure to benchmark:
```

Step 3 : After choosing a data structure to benchmark, there will be functions that are available within the program displayed in the terminal:

(1) Add Item

(2) Increase Quantity

(3) Decrease Quantity

(4) Remove Item

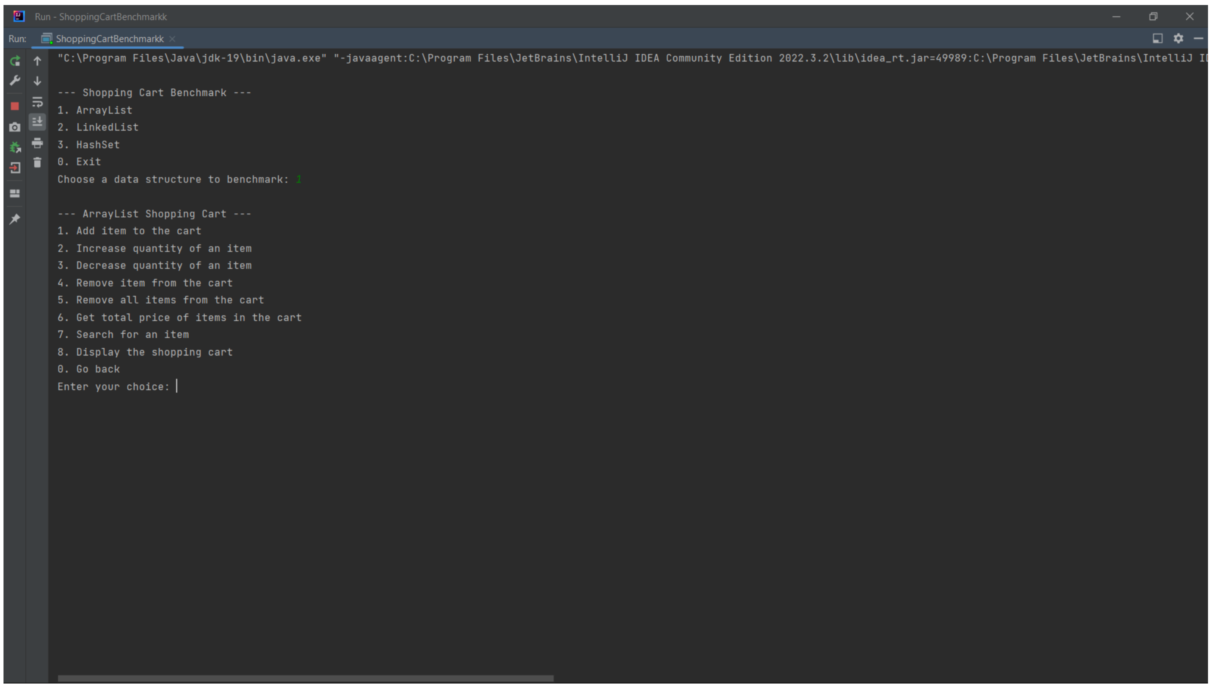
(5) Remove All Item

(6) Total Price

(7) Search Item

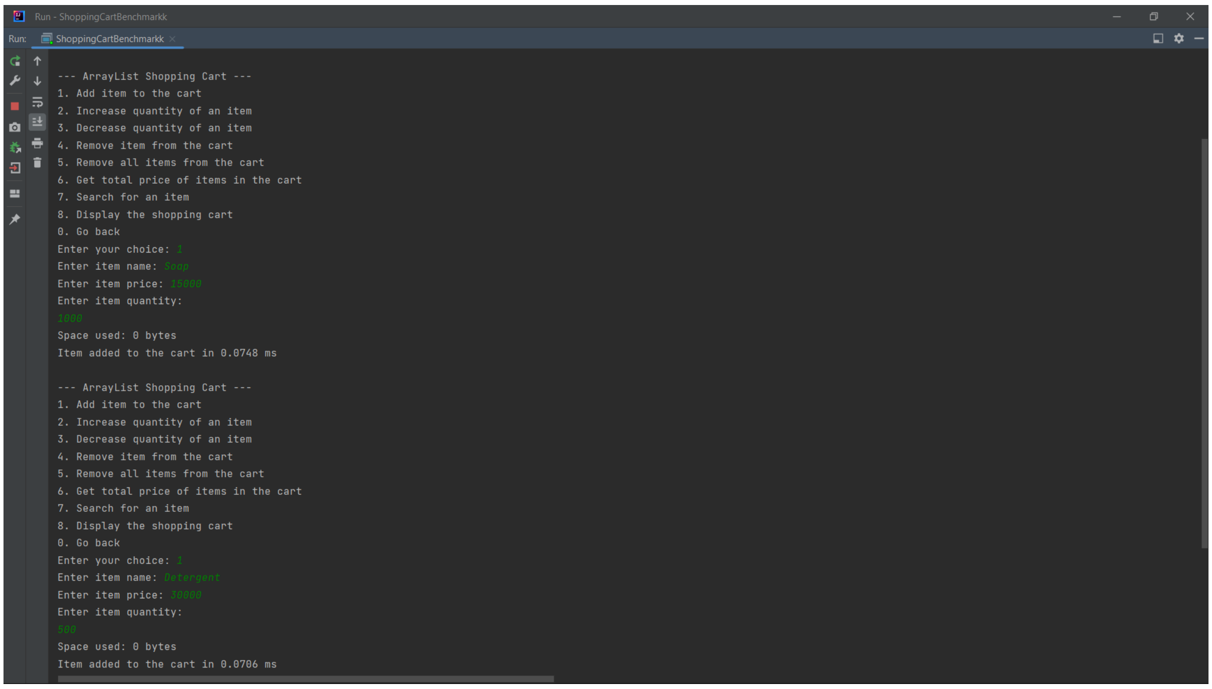
(8) Display Cart

(9) Back



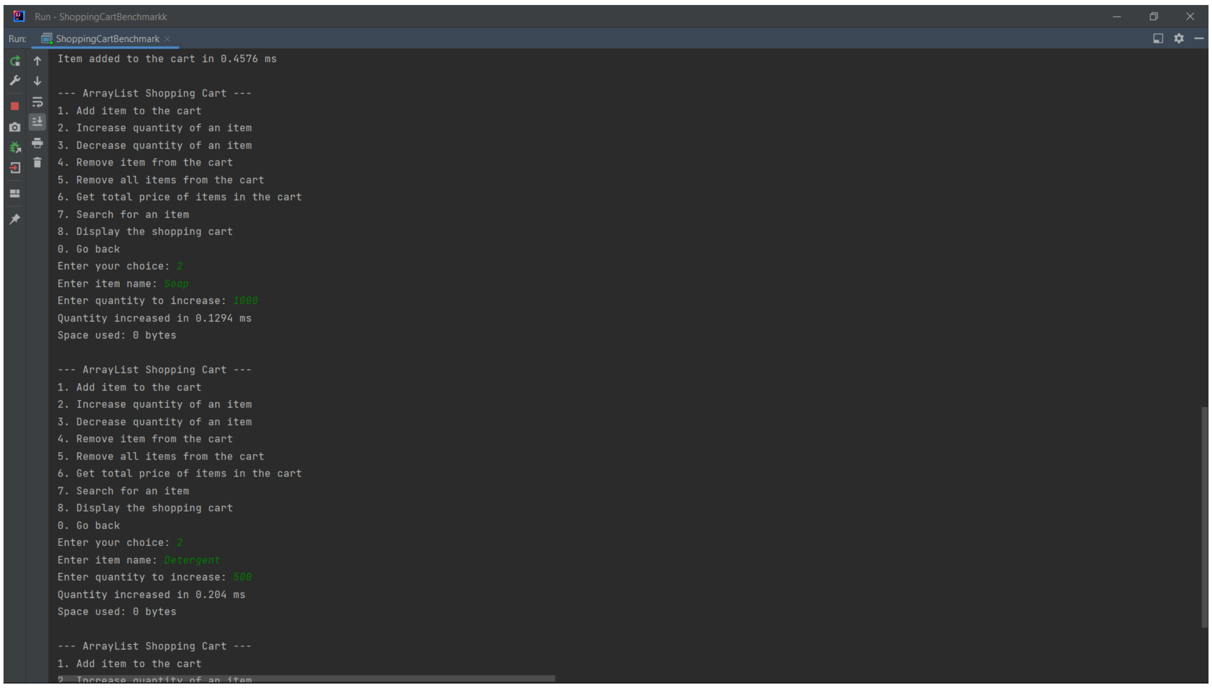
```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\lib\idea_rt.jar=49989:C:\Program Files\JetBrains\IntelliJ I

--- Shopping Cart Benchmark ---
1. ArrayList
2. LinkedList
3. HashSet
0. Exit
Choose a data structure to benchmark: 1

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice:
```

Option 1 (Add Item): After typing in ‘1’, you will be prompted to enter the item description (name, price, quantity) to be added to the cart. Once you enter it, the item will be stored in the data structure based on the amount entered.



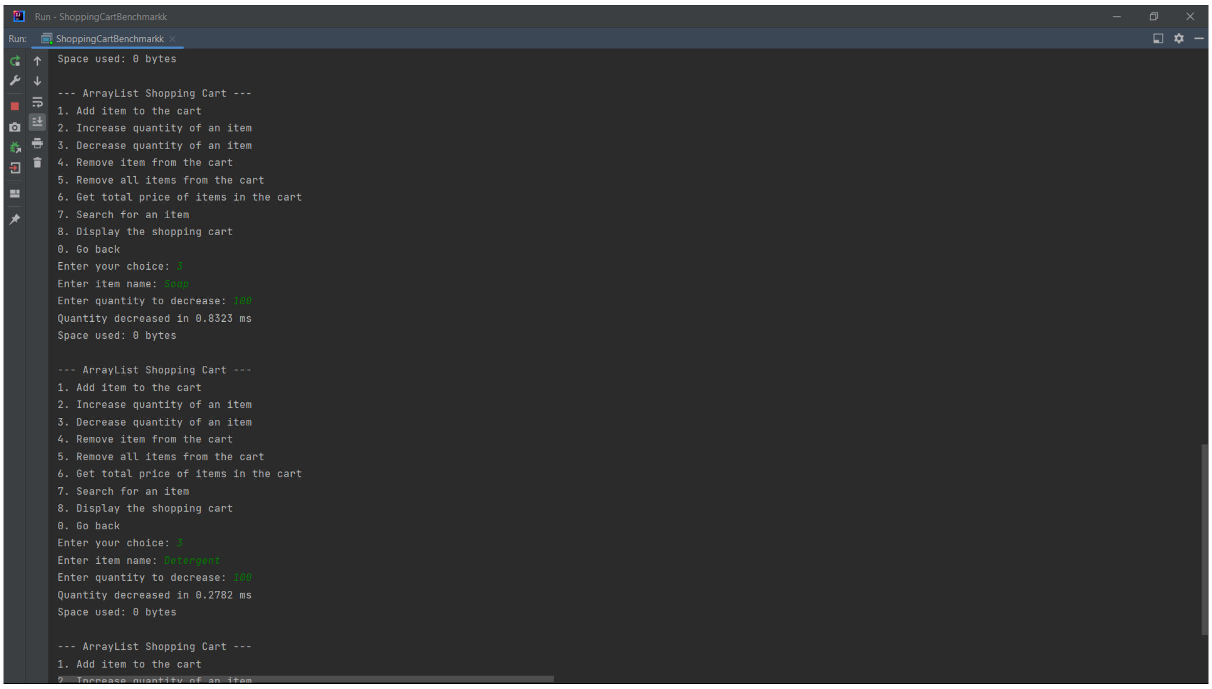
```
--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 1
Enter item name: Soap
Enter item price: 15000
Enter item quantity:
1000
Space used: 0 bytes
Item added to the cart in 0.0748 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 1
Enter item name: Detergent
Enter item price: 30000
Enter item quantity:
500
Space used: 0 bytes
Item added to the cart in 0.0706 ms
```

Option 2 (Increase Quantity) : After typing in '2', you will need to enter a name of an item that exist in the cart. Then enter the amount you want to increase of the selected item.



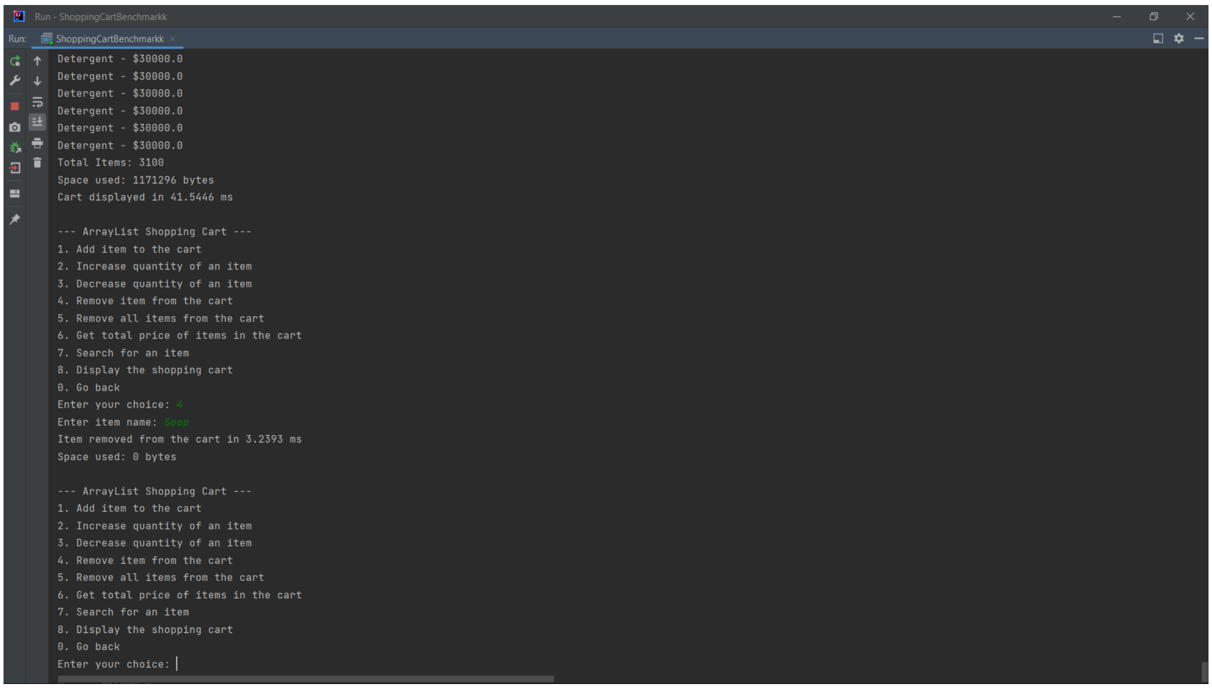
```
Item added to the cart in 0.4576 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 2
Enter item name: Soap
Enter quantity to increase: 1000
Quantity increased in 0.1294 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 2
Enter item name: Detergent
Enter quantity to increase: 500
Quantity increased in 0.204 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
```

Option 3 (Decrease Quantity) : After typing in '3', you will need to enter a name of an item that exist in the cart. Then enter the amount you want to reduce of the selected item.



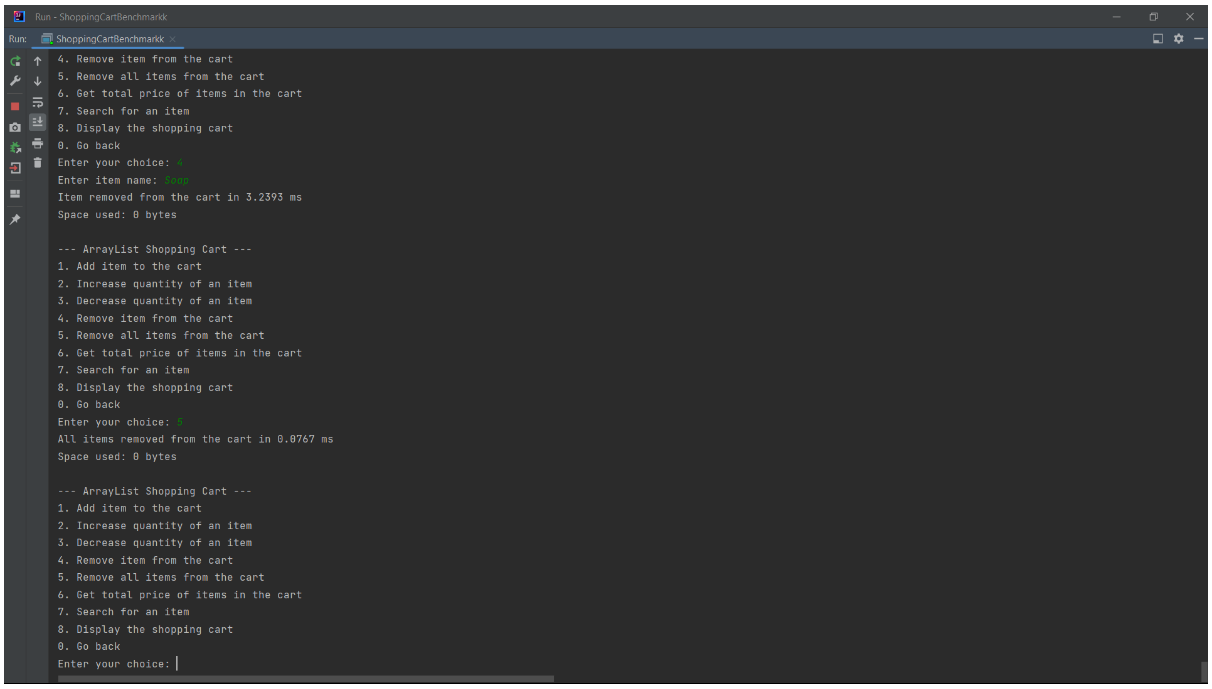
```
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 3
Enter item name: Soap
Enter quantity to decrease: 100
Quantity decreased in 0.8323 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 3
Enter item name: Detergent
Enter quantity to decrease: 100
Quantity decreased in 0.2782 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
```

Option 4 (Remove Item) : After typing in ‘4’, you will need to enter a name of an item that exist in the cart. Once you enter it, all instances of the item that matches the name entered will be deleted.



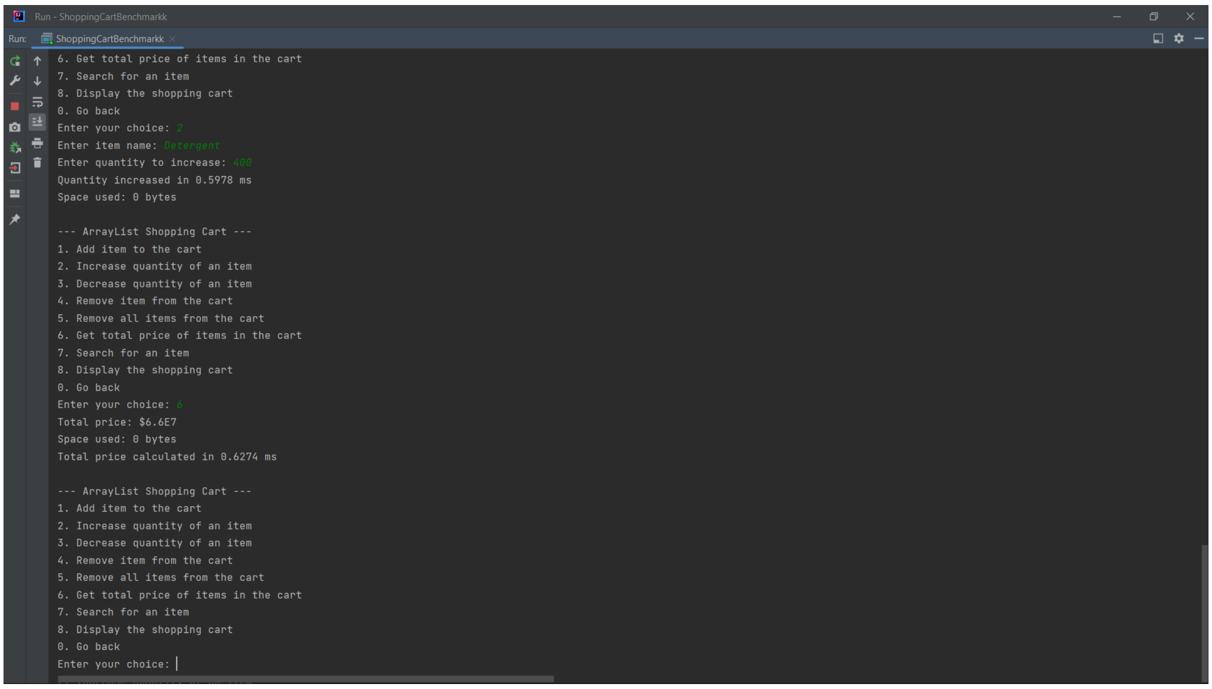
```
Detergent - $30000.0
Detergent - $30000.0
Detergent - $30000.0
Detergent - $30000.0
Detergent - $30000.0
Detergent - $30000.0
Total Items: 3100
Space used: 1171296 bytes
Cart displayed in 41.5446 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 4
Enter item name: Soap
Item removed from the cart in 3.2393 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 
```

Option 5 (Clear Cart): After you enter '5', all of the items in the cart will be cleared.



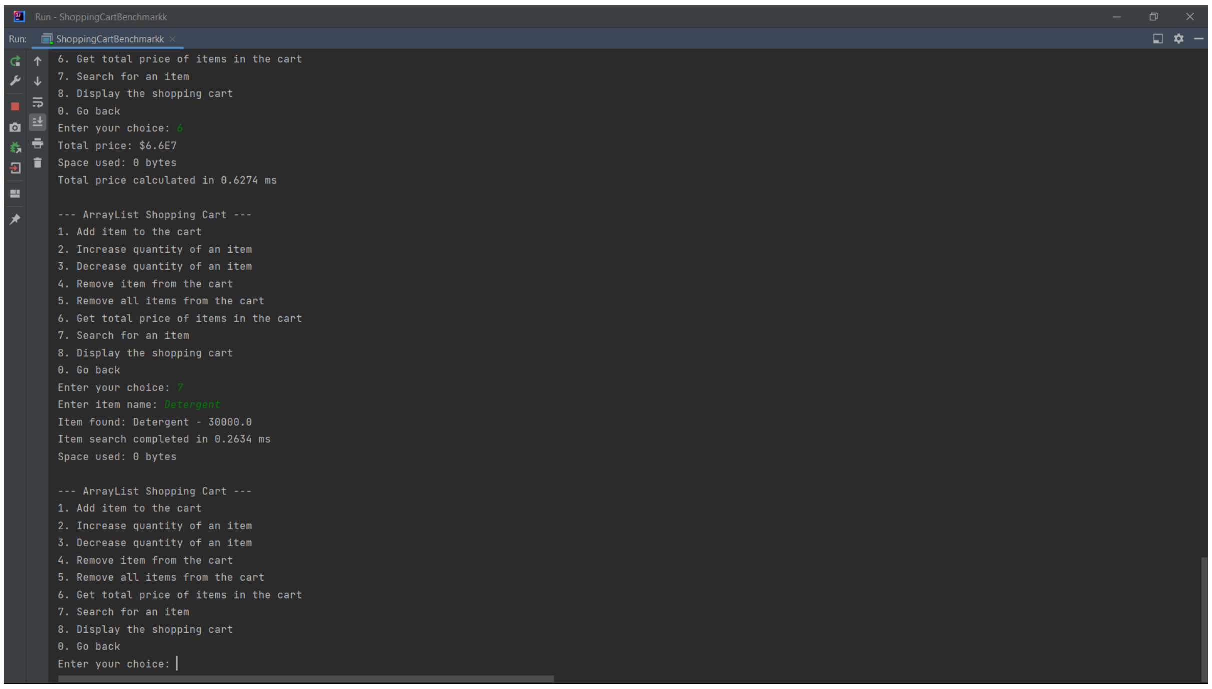
```
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 4
Enter item name: Soap
Item removed from the cart in 3.2393 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 5
All items removed from the cart in 0.0767 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice:
```

Option 6 (Total Price): After typing in '6', the prices of all the items in the cart will be quantified and displayed in the terminal.



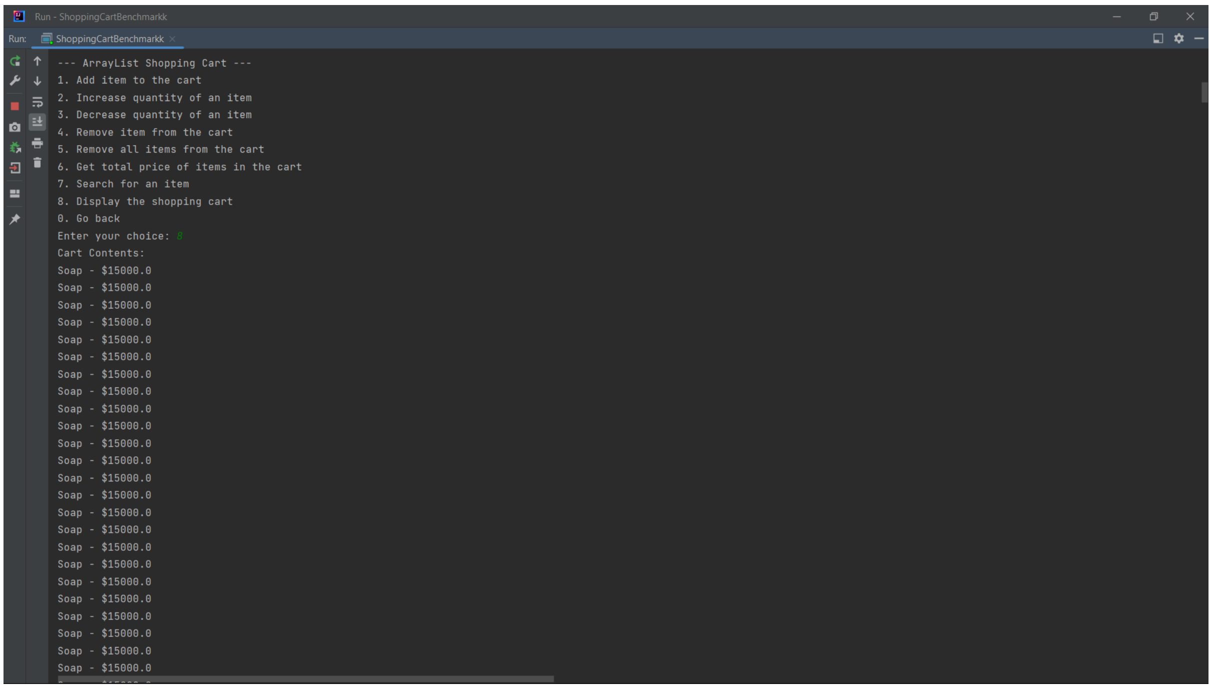
```
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 2
Enter item name: Detergent
Enter quantity to increase: 400
Quantity increased in 0.5978 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 6
Total price: $6.6E7
Space used: 0 bytes
Total price calculated in 0.6274 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice:
```

Option 7 (Search Item): After typing in '7', enter the name of the item that you want to look for. The program will then iterate through the list and search for the first instance of the item that matches the description entered. If the item exist in the cart, the full item description will be displayed in the terminal.



```
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 6
Total price: $6.6E7
Space used: 0 bytes
Total price calculated in 0.6274 ms

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 7
Enter item name: Detergent
Item found: Detergent - 30000.0
Item search completed in 0.2634 ms
Space used: 0 bytes

--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 
```

Option 8 (Display Cart): After typing in ‘8’, every item in the cart will be displayed one by one until all of items have been iterated in the terminal.



```
--- ArrayList Shopping Cart ---
1. Add item to the cart
2. Increase quantity of an item
3. Decrease quantity of an item
4. Remove item from the cart
5. Remove all items from the cart
6. Get total price of items in the cart
7. Search for an item
8. Display the shopping cart
0. Go back
Enter your choice: 8
Cart Contents:
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
Soap - $15000.0
```