GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

Homework 2 Report

BERRU LAFCI

1901042681

Running Command and Results:

Basic Array, ArrayList, LinkedList have same outputs:

For testcase 4:

Accounts created

```
berry@DESKTOP-O92GAB6:/mnt/c/Users/lafci/Desktop/3. Sinif/3. sinif 2. dönem/Data Structure/HW/HW3/hw3_a1$ java hw3_a1.testclass4
An account with username gizemsungu has been created.

An account with username sibelgulmez has been created.

An account with username gokhankaya has been created.

An account with username berrylafci has been created.

An account with username nanaosaki has been created.

An account with username shinokazaki has been created.

An account with username ichigokurosaki has been created.

An account with username orihimeinoue has been created.

An account with username lelouch has been created.

An account with username sasukeuchiha has been created.
```

Unliking, uncommenting, unfollowing, unblocking will be same:

```
Liking a post of sibelgulmez...

Viewing sibelgulmez's posts' interactions...
--------------------
(PostId: 1): Tokyo is a great city!
The post was liked by the following account(s): gizemsungu,
gokhankaya,
The post has no comments.
(PostId: 2): Tokyo is capital of Japan!
The post has no likes.
The post has no comments.
(PostId: 3): I love Tokyo!
The post has no likes.
The post has no comments.
--------------------

Unliking a post of sibelgulmez...

Viewing sibelgulmez's posts' interactions...
--------------------
(PostId: 1): Tokyo is a great city!
The post was liked by the following account(s): gizemsungu,
The post has no comments.
(PostId: 2): Tokyo is capital of Japan!
The post has no likes.
The post has no comments.
(PostId: 3): I love Tokyo!
The post has no likes.
The post has no comments.
--------------------
```

History:

```
Showing berrylafci's history...

--------------------
You blocked sasukeuchiha.
You blocked nanaosaki.
You unblocked sasukeuchiha.
You unblocked nanaosaki.
--------------------
```

LDLinkList:

```
Commenting a post of ichigokurosaki...

Viewing ichigokurosaki's posts' interactions...
--------------------
(PostId: 4): Bleach new season is coming!
The post has no likes.
The post has 5 comment(s).
Comment 7: 'orihimeinoue' said 'I'm so excited!'.
size:5
Comment 9: 'lelouch' said 'Code Geass is better!'.
size:5
Comment 10: 'sibelgulmez' said 'aaaaa'.
size:5
Comment 11: 'berrylafci' said 'bbbbbb'.
size:5
Comment 10: 'sasukeuchiha' said 'Naruto is better!'.
size:5
(PostId: 5): I love Bleach!
The post has no likes.
The post has 1 comment(s).
Comment 8: 'orihimeinoue' said 'Me too!'.
size:1
--------------------
```

In here I printed to size to show the original size of the list. So the 5th one is lazily deleted in here.

```
Deleting a comment of ichigokurosaki...

Viewing ichigokurosaki's posts' interactions...
--------------------
(PostId: 4): Bleach new season is coming!
The post has no likes.
The post has 4 comment(s).
Comment 7: 'orihimeinoue' said 'I'm so excited!'.
size:5
Comment 9: 'lelouch' said 'Code Geass is better!'.
size:5
Comment 10: 'sibelgulmez' said 'aaaaa'.
size:5
Comment 11: 'berrylafci' said 'bbbbbb'.
size:5
(PostId: 5): I love Bleach!
The post has no likes.
The post has 1 comment(s).
Comment 8: 'orihimeinoue' said 'Me too!'.
size:1
--------------------
```

But when we delete one more, they will be delete physically so the size is 3 now.

```
Deleting a comment of ichigokurosaki...

Viewing ichigokurosaki's posts' interactions...
--------------------
(PostId: 4): Bleach new season is coming!
The post has no likes.
The post has 3 comment(s).
Comment 7: 'orihimeinoue' said 'I'm so excited!'.
size:3
Comment 10: 'sibelgulmez' said 'aaaaa'.
size:3
Comment 11: 'berrylafci' said 'bbbbbb'.
size:3
(PostId: 5): I love Bleach!
The post has no likes.
The post has 1 comment(s).
Comment 8: 'orihimeinoue' said 'Me too!'.
size:1
--------------------
```

Also in here it deletes lazily from blocked list. In addition, I showed the history.

Time Complexity Analysis

In ArrayList, methods like add, remove will be O(n). This is because the loop iterates through each element in the list, and the size of the list determines the number of iterations. Therefore, the time complexity is linear with respect to the number of elements in the list. Example loops:

```
following.add(account);
account.getFollowers().add(this);
```

```
following.remove(account);
account.getFollowers().remove(this);
```

```
public String showFollowings(Account account) {
    String followings = "";
    for (int i = 0; i < account.getFollowing().size(); i++) {
        followings += account.getFollowing().get(i).getUsername() + ", ";
    }
    return followings;
}
```

In linked list, the loop iterates over each element in the LinkedList, so for each iteration, the list needs to be traversed to get the element at that index. This results in a nested loop like structure where each iteration of the outer loop requires traversing the list from the beginning to the current index, resulting in a time complexity of O(n) for each iteration. As a result, the overall time complexity will be O(n^2).

That's why I chose to use iterator to some loops for better efficiency. Because, the loop begins by creating an iterator using the iterator method. The hasNext method of the iterator is then called to check if there are any more elements in the collection. If there are, the next method of the iterator is called to retrieve the next element in the collection. This process is repeated until all elements in the collection have been processed.

Since the loop iterates over each element in the collection exactly once, the time complexity of the loop is O(n).

```
Iterator<Post> iterator = account.getPosts().iterator();
while(iterator.hasNext()){
    Post post = iterator.next();
    System.out.println("(PostID: " + post.getPostId() + ") " + account.getUsername() + ": " + post.getContent() + "");
}
```

And methods like add, remove will be O(n) because traversing will take linear time.

In LDLinkedList, overridden get and add methods complexities will take O(n) time because of the same reason .

```java
@Override
public boolean add(E data) {
    // if list is empty, create a new node and make it head
    if (head == null) {
        head = new Node<E>(data);
        size++;
        return true;
    }

    // traverse till the last node
    Node<E> current = head;
    while (current.next != null) {
        current = current.next;
    }

    // create a new node and make it the next of the last node
    current.next = new Node<E>(data);
    size++;
    return true;
}
```

But for remove method, complexity will be O(n^2) because when it starts looping around for normal removing, it will go into a loop again if marked as deleted is more than 2.

```java
@Override
public boolean remove(Object data) {
    Node<E> current = head;
    //Node<E> previous = null;

    while (current != null) {
        // if the head is the node to be deleted, mark it as deleted
        if (current.data == data) {
            current.isDeleted = true;
            numDeleted++;
            // if there are more than 2 deleted nodes, remove them
            if (numDeleted >= 2) {
                deleteNode();
            }
            return true;
        }
        // if the head is not the node to be deleted, move to the next node
        else {
            //previous = current;
            current = current.next;
        }
    }
    // if the node is not found, return false
    return false;
}
```

And for other loops, I couldn't write Iterator so their complexity will be O(n^2) again because of nested-loop.

```java
for (int i = 0; i < account.getPosts().size(); i++) {
    System.out.println("(PostID: " + account.getPosts().get(i).getPostId() + ") " + account.getUsername() +
}
System.out.println("------------------------------\n");
```

So I calculated experimental running time like this:

Implementation Type	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Basic Array Structure (HW1)	O(n)	O(n)	O(n)	Not available
Array List Structure (a)	O(n)	O(n)	O(n)	O(n)
Linked List Structure (b)	O(n2)	O(n2)	O(n2)	O(n2)
LD Linked List Structure (c)	O(n2)	O(n2)	O(n2)	O(n2)

And I calculated theoretical running time like this:

Implementation Type	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Basic Array Structure (HW1)	0,116 sec	0119	0120	Not available
Array List Structure (a)	0,117	0119	0120	0,128
Linked List Structure (b)	0,116	0,120	0,121	0,129
LD Linked List Structure (c)	0,117	0119	0,123	0,137 sec

Problem solution approach for LDLinkedList:

When we first remove a data in LDLinkList, the isDeleted value on that node becomes true.

```
// if the head is the node to be
if (current.data == data) {
    current.isDeleted = true;
    numDeleted++;
```

I created a function called ldlist() so that lazy deleted ones doesn't appear in the list. This function prints all nodes except isDeleted marked ones. Thus, it is still not physically deleted.

```
public LDLinkedList<E> ldlist(){
    LDLinkedList<E> list = new LDLinkedList<E>();

    Node<E> current = head;
    while (current != null) {
        if (!current.isDeleted) {
            list.add(current.data);
        }
        current = current.next;
    }
    return list;
}
```

Whenever we do a 2nd remove in the list, it goes to the deleteNode() function and physically removes the nodes marked here from the list.

```
if (numDeleted >= 2) {
    deleteNode();
}
```

```
private void deleteNode() {
    Node<E> current = head;
    Node<E> previous = null;

    // traverse the list
    while (current != null) {
        // if the node is marked as
        if (current.isDeleted) {
```

If I would use iterators, printing lazily deleted list is would be more efficient.