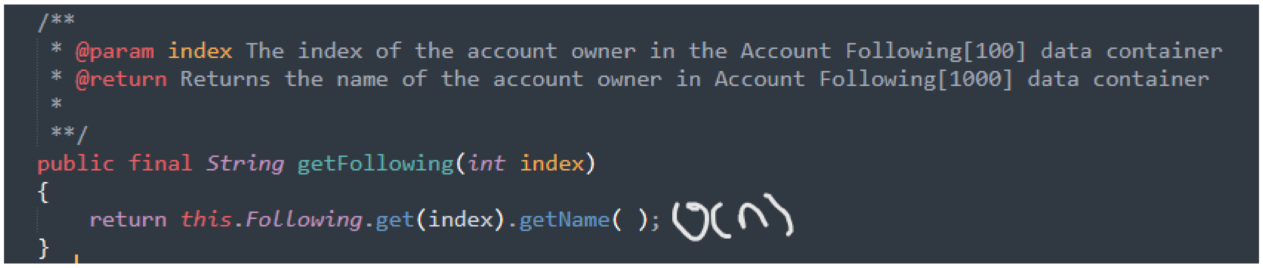
Account Object’s method time complexity analysis, LinkedList data structure.

1-)

String getFollowing(int index)



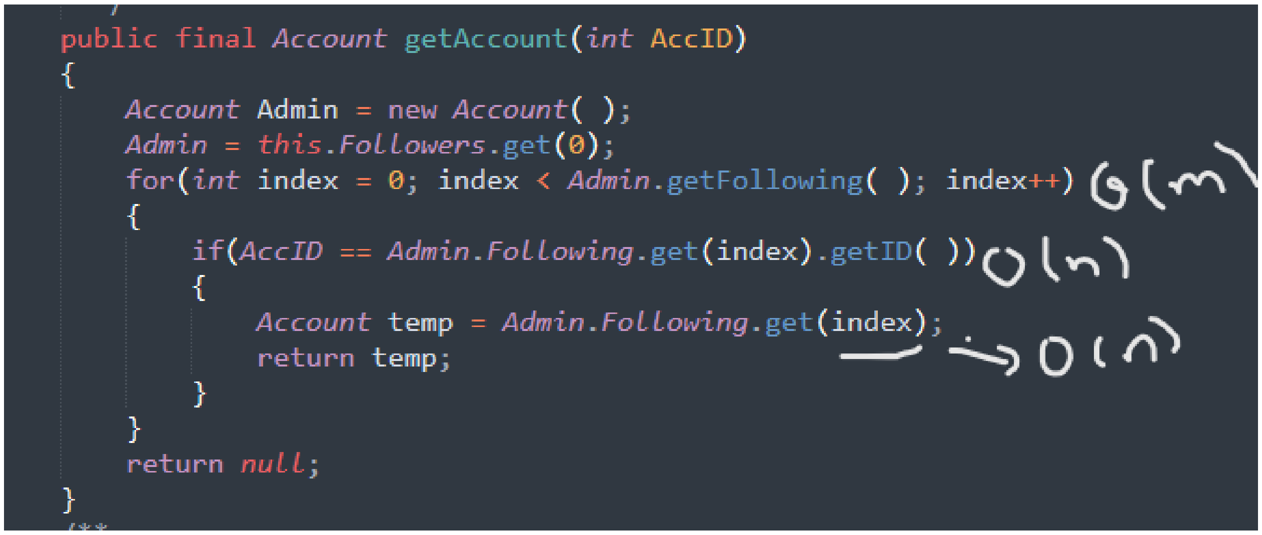
```
/**
 * @param index The index of the account owner in the Account Following[100] data container
 * @return Returns the name of the account owner in Account Following[1000] data container
 *
 **/
public final String getFollowing(int index)
{
    return this.Following.get(index).getName( ); O(n)
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(n)

2-)

Public final getAccount(int AccID)

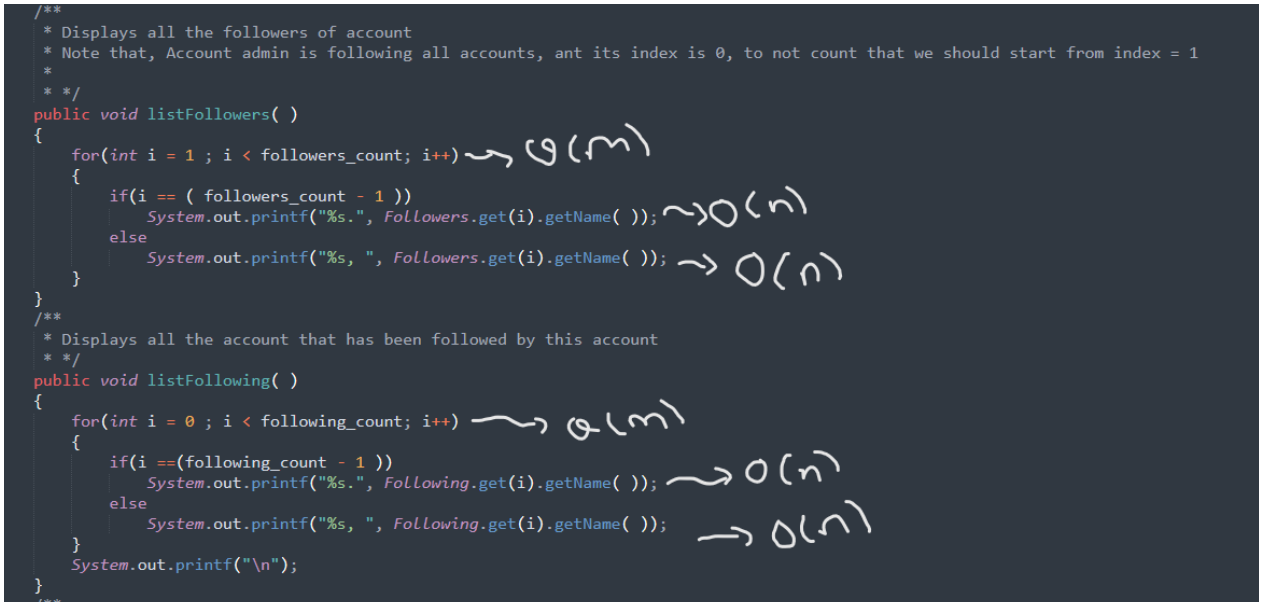


```
public final Account getAccount(int AccID)
{
    Account Admin = new Account( );
    Admin = this.Followers.get(0);
    for(int index = 0; index < Admin.getFollowing( ); index++) O(m)
    {
        if(AccID == Admin.Following.get(index).getID( ))O(n)
        {
            Account temp = Admin.Following.get(index);
            return temp;                                 → O(n)
        }
    }
    return null;
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n^2)

3-) void listFollowers( ) and listFollowing( )

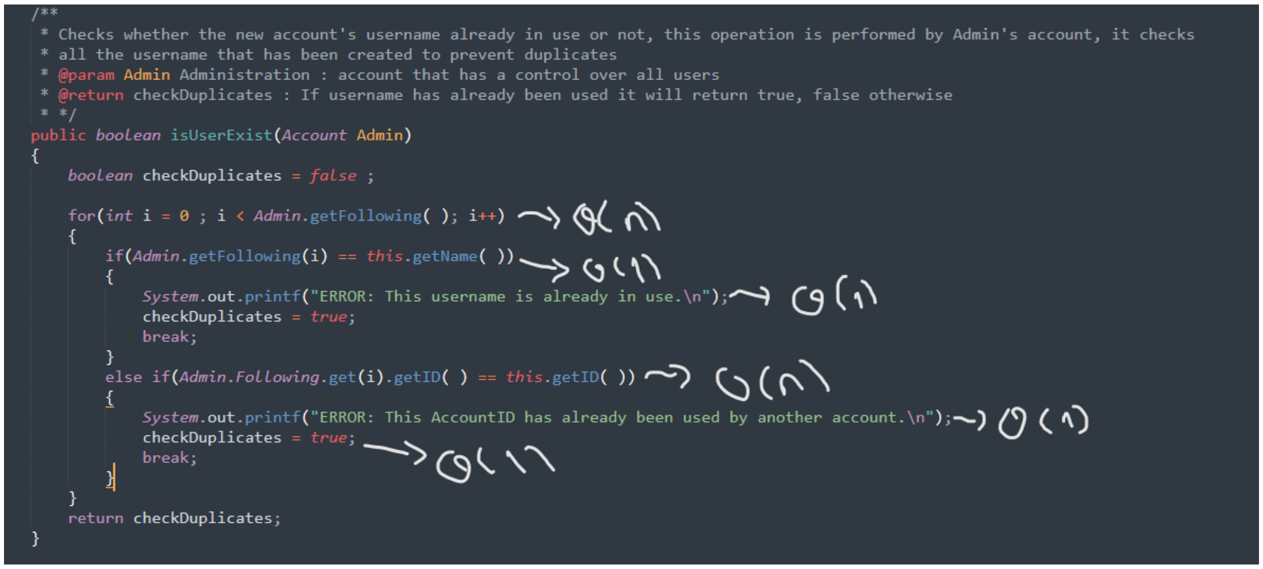


```
/**
 * Displays all the followers of account
 * Note that, Account admin is following all accounts, ant its index is 0, to not count that we should start from index = 1
 *
 * */
public void listFollowers( )
{
    for(int i = 1 ; i < followers_count; i++) → O(m)
    {
        if(i == ( followers_count - 1 ))
            System.out.printf("%s.", Followers.get(i).getName( )); → O(n)
        else
            System.out.printf("%s, ", Followers.get(i).getName( )); → O(n)
    }
}
/**
 * Displays all the account that has been followed by this account
 * */
public void listFollowing( )
{
    for(int i = 0 ; i < following_count; i++) → O(m)
    {
        if(i ==(following_count - 1 ))
            System.out.printf("%s.", Following.get(i).getName( )); → O(n)
        else
            System.out.printf("%s, ", Following.get(i).getName( )); → O(n)
    }
    System.out.printf("\n");
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n)

4-)boolean isUserExist (Account)

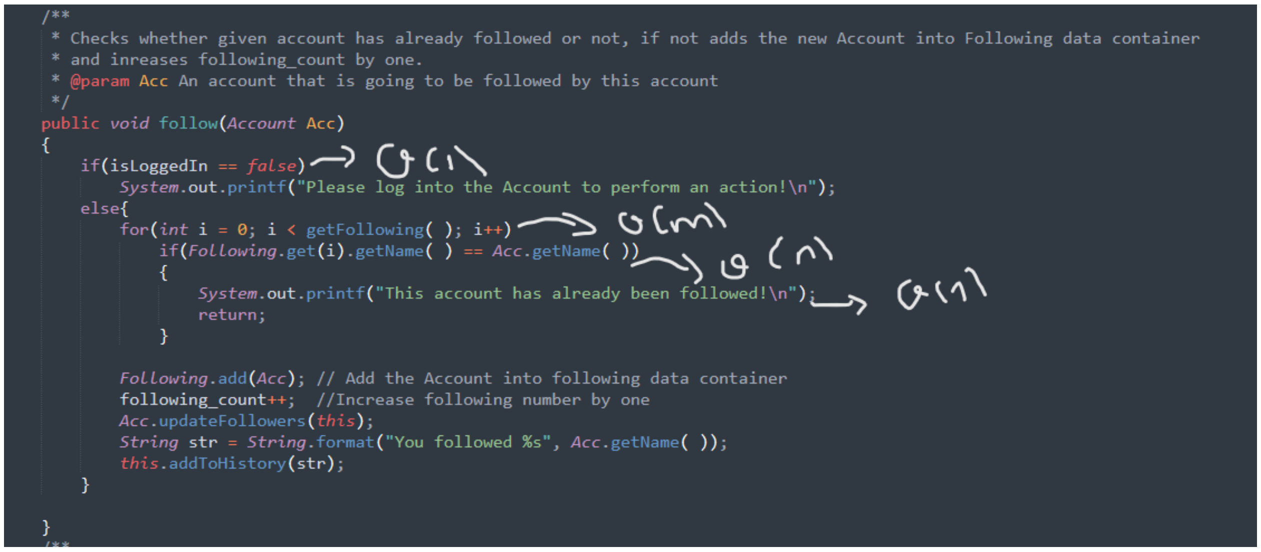


```
/**
 * Checks whether the new account's username already in use or not, this operation is performed by Admin's account, it checks
 * all the username that has been created to prevent duplicates
 * @param Admin Administration : account that has a control over all users
 * @return checkDuplicates : If username has already been used it will return true, false otherwise
 * */
public boolean isUserExist(Account Admin)
{
    boolean checkDuplicates = false ;

    for(int i = 0 ; i < Admin.getFollowing( ); i++) → O(n)
    {
        if(Admin.getFollowing(i) == this.getName( )) → O(1)
        {
            System.out.printf("ERROR: This username is already in use.\n"); → O(1)
            checkDuplicates = true;
            break;
        }
        else if(Admin.Following.get(i).getID( ) == this.getID( )) → O(n)
        {
            System.out.printf("ERROR: This AccountID has already been used by another account.\n"); → O(1)
            checkDuplicates = true; → O(1)
            break;
        }
    }
    return checkDuplicates;
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n)

5-) public void follow(Account Acc)

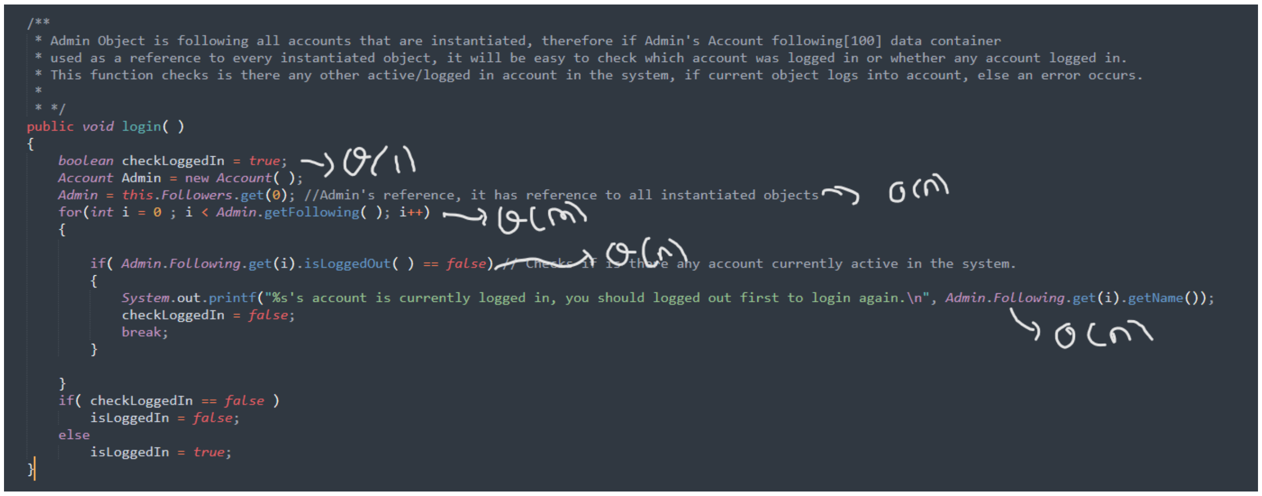


```java
/**
 * Checks whether given account has already followed or not, if not adds the new Account into Following data container
 * and inreases following_count by one.
 * @param Acc An account that is going to be followed by this account
 */
public void follow(Account Acc)
{
    if(isLoggedIn == false)
        System.out.printf("Please log into the Account to perform an action!\n");
    else{
        for(int i = 0; i < getFollowing( ); i++)
            if(Following.get(i).getName( ) == Acc.getName( ))
            {
                System.out.printf("This account has already been followed!\n");
                return;
            }

        Following.add(Acc); // Add the Account into following data container
        following_count++;  //Increase following number by one
        Acc.updateFollowers(this);
        String str = String.format("You followed %s", Acc.getName( ));
        this.addToHistory(str);
    }

}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n)

6-) public void login( )

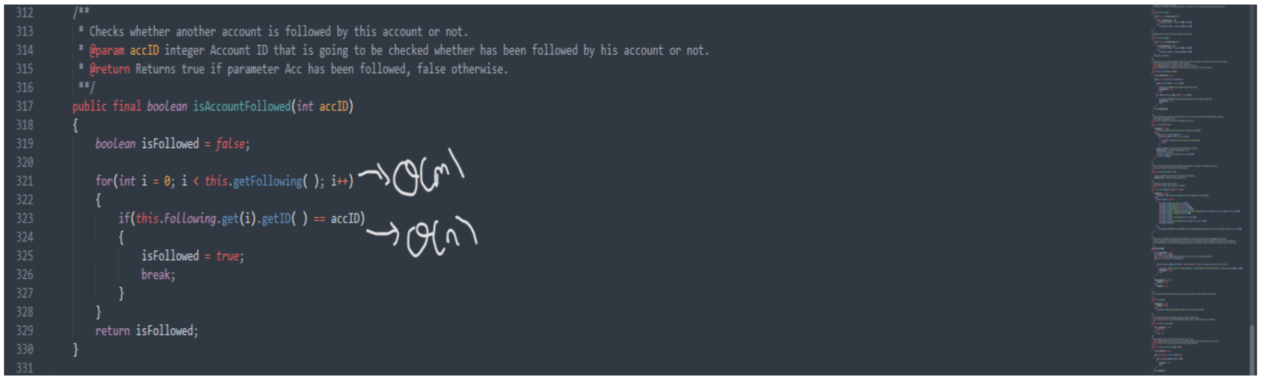


```java
/**
 * Admin Object is following all accounts that are instantiated, therefore if Admin's Account following[100] data container
 * used as a reference to every instantiated object, it will be easy to check which account was logged in or whether any account logged in.
 * This function checks is there any other active/logged in account in the system, if current object logs into account, else an error occurs.
 *
 * */
public void login( )
{
    boolean checkLoggedIn = true;
    Account Admin = new Account( );
    Admin = this.Followers.get(0); //Admin's reference, it has reference to all instantiated objects
    for(int i = 0 ; i < Admin.getFollowing( ); i++)
    {

        if( Admin.Following.get(i).isLoggedOut( ) == false) // Checks if is there any account currently active in the system.
        {
            System.out.printf("%s's account is currently logged in, you should logged out first to login again.\n", Admin.Following.get(i).getName());
            checkLoggedIn = false;
            break;
        }

    }
    if( checkLoggedIn == false )
        isLoggedIn = false;
    else
        isLoggedIn = true;
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n^2)

7-) public final boolean isAccountFollowed(int accID)

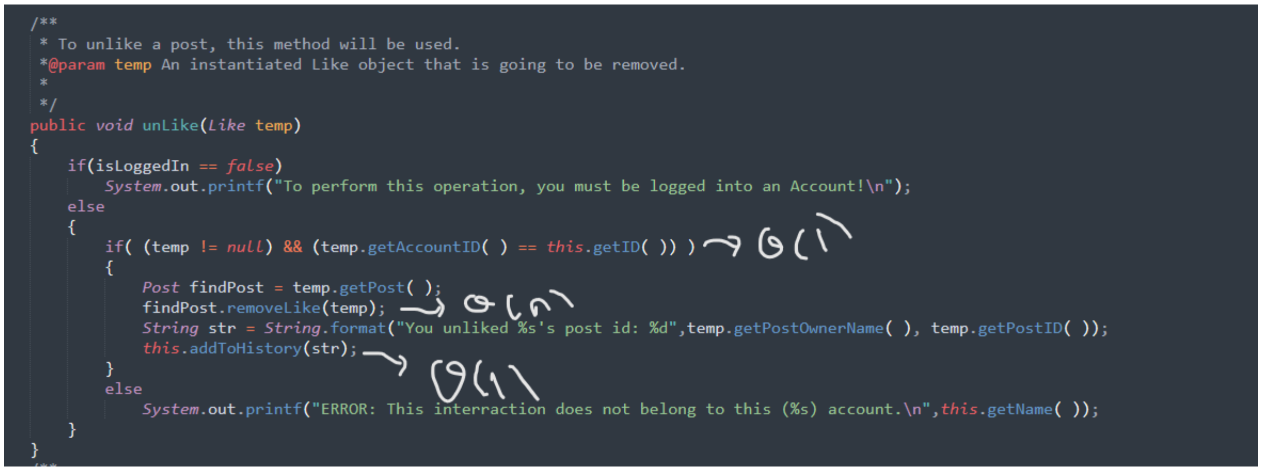


```java
/**
 * Checks whether another account is followed by this account or not.
 * @param accID integer Account ID that is going to be checked whether has been followed by his account or not.
 * @return Returns true if parameter Acc has been followed, false otherwise.
 **/
public final boolean isAccountFollowed(int accID)
{
    boolean isFollowed = false;

    for(int i = 0; i < this.getFollowing( ); i++)
    {
        if(this.Following.get(i).getID( ) == accID)
        {
            isFollowed = true;
            break;
        }
    }
    return isFollowed;
}
```

- get() method searching for an element takes *O(n)* time in LinkedList.

T(n) = O(m*n)

8-) void unLike(Like temp)



```java
/**
 * To unlike a post, this method will be used.
 *@param temp An instantiated Like object that is going to be removed.
 *
 */
public void unLike(Like temp)
{
    if(isLoggedIn == false)
        System.out.printf("To perform this operation, you must be logged into an Account!\n");
    else
    {
        if( (temp != null) && (temp.getAccountID( ) == this.getID( )) )
        {
            Post findPost = temp.getPost( );
            findPost.removeLike(temp);
            String str = String.format("You unliked %s's post id: %d",temp.getPostOwnerName( ), temp.getPostID( ));
            this.addToHistory(str);
        }
        else
            System.out.printf("ERROR: This interraction does not belong to this (%s) account.\n",this.getName( ));
    }
}
```

Time complexity of removeLike is O(n)

Time complexity of addToHistory is O(1)

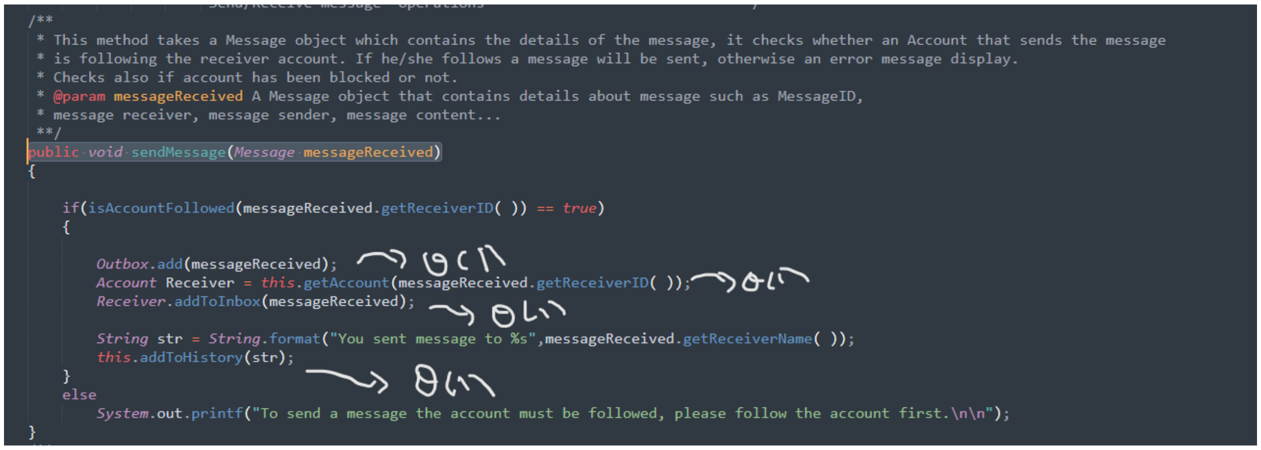
T(n) = O(n)

9-) void unComment(Comment temp)

This method has the same code structure, the only difference that is provoking the removeComment method which has a O(n) time complexity

T(n) = O(n)

9-) public void sendMessage(Message messageReceived)



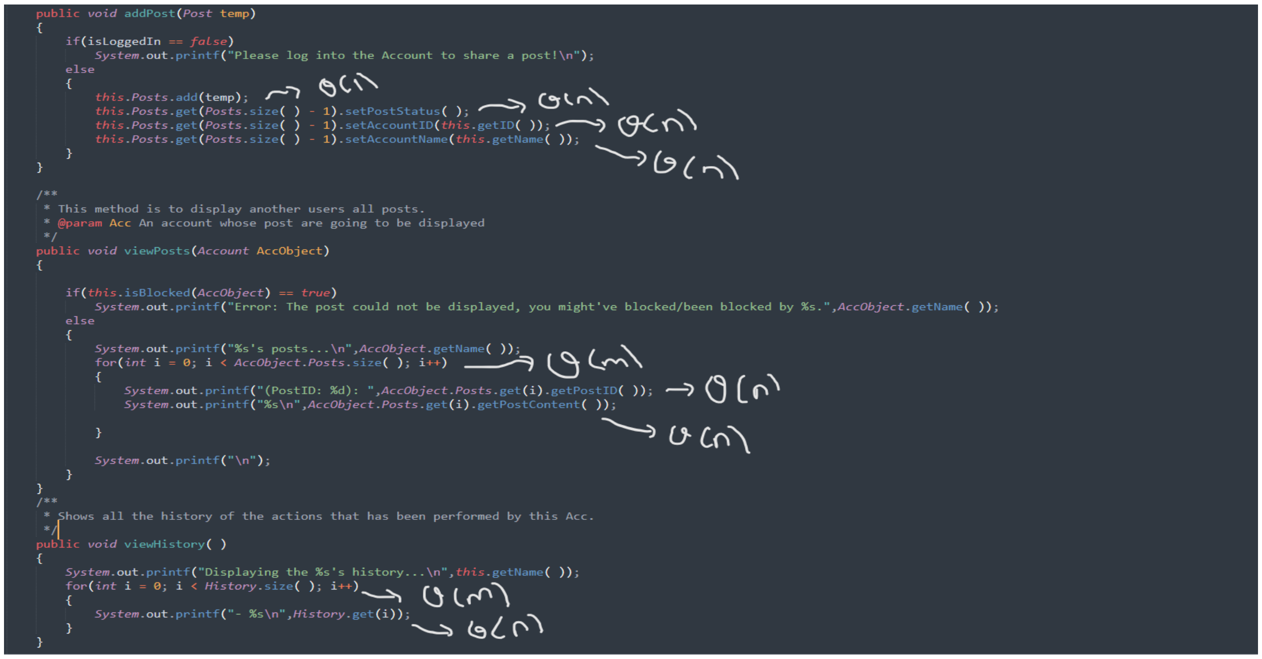
```java
/**
 * This method takes a Message object which contains the details of the message, it checks whether an Account that sends the message
 * is following the receiver account. If he/she follows a message will be sent, otherwise an error message display.
 * Checks also if account has been blocked or not.
 * @param messageReceived A Message object that contains details about message such as MessageID,
 * message receiver, message sender, message content...
 **/
public void sendMessage(Message messageReceived)
{

    if(isAccountFollowed(messageReceived.getReceiverID( )) == true)
    {

        Outbox.add(messageReceived);
        Account Receiver = this.getAccount(messageReceived.getReceiverID( ));
        Receiver.addToInbox(messageReceived);

        String str = String.format("You sent message to %s",messageReceived.getReceiverName( ));
        this.addToHistory(str);
    }
    else
        System.out.printf("To send a message the account must be followed, please follow the account first.\n\n");
}
```

Time complexity of addToInbox is O(1)

T(n) = O(1)

10-)

- public void addPost(Post temp)
- public void viewPosts(Account AccObject)
- public void viewHistory( )



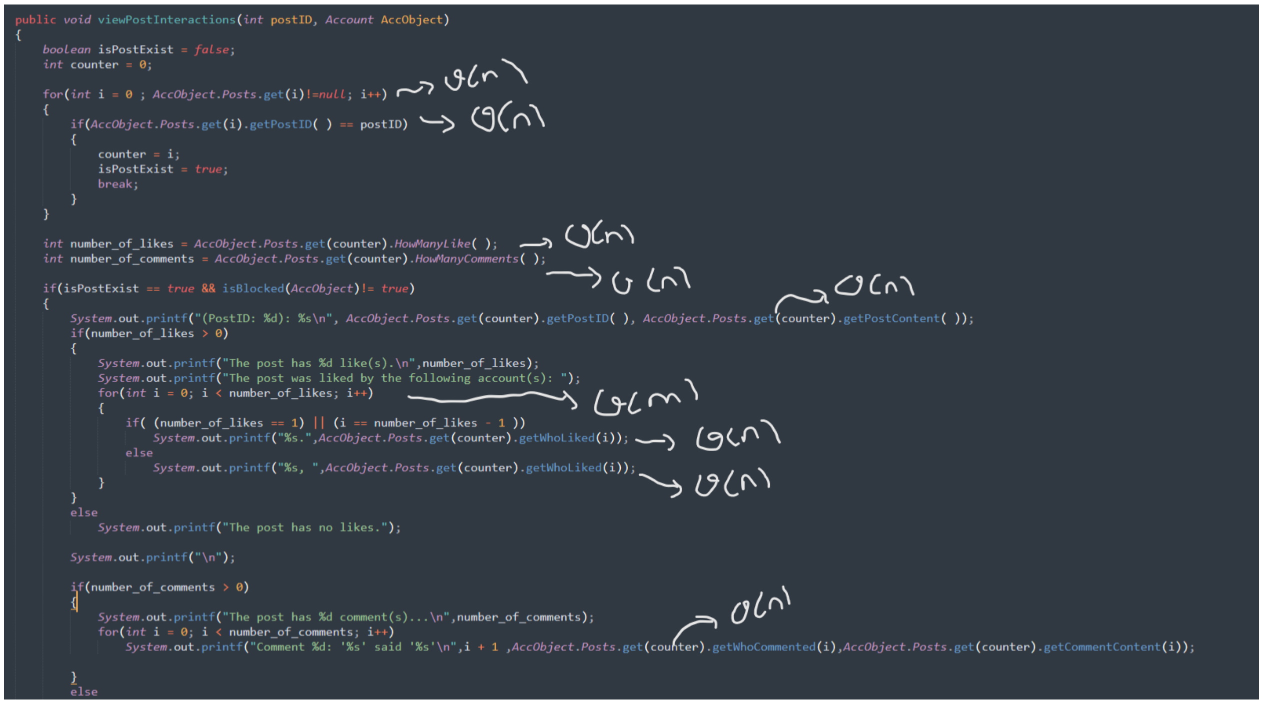
```java
public void addPost(Post temp)
{
    if(isLoggedIn == false)
        System.out.printf("Please log into the Account to share a post!\n");
    else
    {
        this.Posts.add(temp);
        this.Posts.get(Posts.size( ) - 1).setPostStatus( );
        this.Posts.get(Posts.size( ) - 1).setAccountID(this.getID( ));
        this.Posts.get(Posts.size( ) - 1).setAccountName(this.getName( ));
    }
}

/**
 * This method is to display another users all posts.
 * @param Acc An account whose post are going to be displayed
 */
public void viewPosts(Account AccObject)
{

    if(this.isBlocked(AccObject) == true)
        System.out.printf("Error: The post could not be displayed, you might've blocked/been blocked by %s.",AccObject.getName( ));
    else
    {
        System.out.printf("%s's posts...\n",AccObject.getName( ));
        for(int i = 0; i < AccObject.Posts.size( ); i++)
        {
            System.out.printf("(PostID: %d): ",AccObject.Posts.get(i).getPostID( ));
            System.out.printf("%s\n",AccObject.Posts.get(i).getPostContent( ));

        }

        System.out.printf("\n");
    }
}
/**
 * Shows all the history of the actions that has been performed by this Acc.
 */
public void viewHistory( )
{
    System.out.printf("Displaying the %s's history...\n",this.getName( ));
    for(int i = 0; i < History.size( ); i++)
    {
        System.out.printf("- %s\n",History.get(i));
    }
}
```

addPost = O(n)

viewPost = O(m*n)

viewHistory = O(m*n)

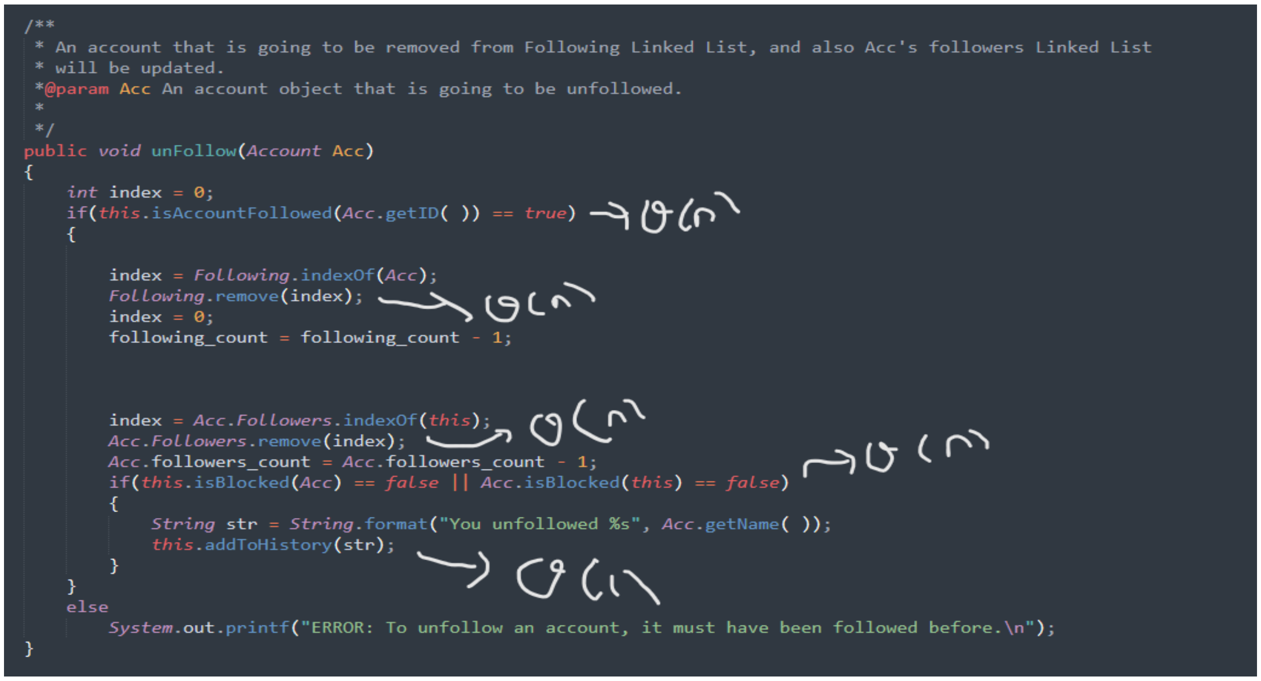
12-) public void viewPostInteractions(int postID, Account AccObject)



```java
public void viewPostInteractions(int postID, Account AccObject)
{
    boolean isPostExist = false;
    int counter = 0;

    for(int i = 0 ; AccObject.Posts.get(i)!=null; i++) → O(n)
    {
        if(AccObject.Posts.get(i).getPostID( ) == postID) → O(n)
        {
            counter = i;
            isPostExist = true;
            break;
        }
    }

    int number_of_likes = AccObject.Posts.get(counter).HowManyLike( ); → O(n)
    int number_of_comments = AccObject.Posts.get(counter).HowManyComments( ); → O(n)

    if(isPostExist == true && isBlocked(AccObject)!= true)
    {
        System.out.printf("(PostID: %d): %s\n", AccObject.Posts.get(counter).getPostID( ), AccObject.Posts.get(counter).getPostContent( )); → O(n)
        if(number_of_likes > 0)
        {
            System.out.printf("The post has %d like(s).\n",number_of_likes);
            System.out.printf("The post was liked by the following account(s): ");
            for(int i = 0; i < number_of_likes; i++) → O(m)
            {
                if( (number_of_likes == 1) || (i == number_of_likes - 1 ))
                    System.out.printf("%s.",AccObject.Posts.get(counter).getWhoLiked(i)); → O(n)
                else
                    System.out.printf("%s, ",AccObject.Posts.get(counter).getWhoLiked(i)); → O(n)
            }
        }
        else
            System.out.printf("The post has no likes.");

        System.out.printf("\n");

        if(number_of_comments > 0)
        {
            System.out.printf("The post has %d comment(s)...\n",number_of_comments);
            for(int i = 0; i < number_of_comments; i++)
                System.out.printf("Comment %d: '%s' said '%s'\n",i + 1 ,AccObject.Posts.get(counter).getWhoCommented(i),AccObject.Posts.get(counter).getCommentContent(i)); → O(n)

        }
        else
```

T(n) = O(n^2)

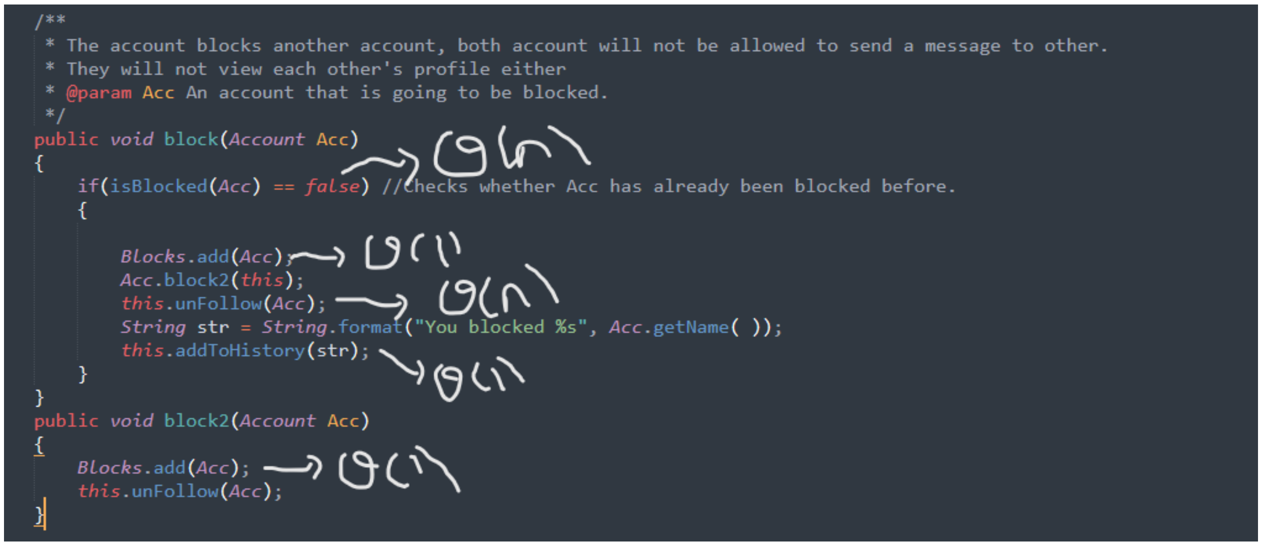
13-) public void unFollow(Account Acc)



```
/**
 * An account that is going to be removed from Following Linked List, and also Acc's followers Linked List
 * will be updated.
 *@param Acc An account object that is going to be unfollowed.
 *
 */
public void unFollow(Account Acc)
{
    int index = 0;
    if(this.isAccountFollowed(Acc.getID( )) == true) → O(n)
    {

        index = Following.indexOf(Acc);
        Following.remove(index); → O(n)
        index = 0;
        following_count = following_count - 1;


        index = Acc.Followers.indexOf(this); → O(n)
        Acc.Followers.remove(index);
        Acc.followers_count = Acc.followers_count - 1;
        if(this.isBlocked(Acc) == false || Acc.isBlocked(this) == false) → O(n)
        {
            String str = String.format("You unfollowed %s", Acc.getName( ));
            this.addToHistory(str); → O(1)
        }
    }
    else
        System.out.printf("ERROR: To unfollow an account, it must have been followed before.\n");
}
```

T(n) = O(n)

14-) public boolean block(Account Acc)



```
/**
 * The account blocks another account, both account will not be allowed to send a message to other.
 * They will not view each other's profile either
 * @param Acc An account that is going to be blocked.
 */
public void block(Account Acc)
{                                   → O(n)
    if(isBlocked(Acc) == false) //checks whether Acc has already been blocked before.
    {

        Blocks.add(Acc); → O(1)
        Acc.block2(this);
        this.unFollow(Acc); → O(n)
        String str = String.format("You blocked %s", Acc.getName( ));
        this.addToHistory(str); → O(1)
    }
}
public void block2(Account Acc)
{
    Blocks.add(Acc); → O(1)
    this.unFollow(Acc);
}
```

T(n) = O(n)