Even Semester (2023)

Assignment Cover Letter

(Group Assignment)

Student Names: Brigitte Sharon Alexander Ivandito Rakaputra Tirza Gabriella	**Student IDs:** 2602119190 2602119303 2602109870
Course Code: COMP6048001	**Course Name:** Data Structures
Class: L2AC	**Name of Lecturer:** Nunung Nurul Qomariyah, S.Kom., M.T.I., Ph.D.
Major: Computer Science	
Title of Assignment: Password Management System	
Type of Assignment: Final Project	
Due Date: 13 - 06 - 2023	

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signatures of Students:

Brigitte Sharon Alexander
Ivandito Rakaputra
Tirza Gabriella

Table of Contents

I. Introduction

A. Problem Description

From learning the different types of data structures to utilizing them in code, we have gained so much knowledge throughout the course of this semester regarding the role of data structures in building a program. Therefore, in this project, we as a team decided that we wanted to work on a password management system to test the efficiency of 4 different data structures. With this program, we are able to enhance our skills in terms of problem solving and case analysis.

B. Objective

Our foremost aim is to test the efficiency (using time complexity) of the 4 data structures that we picked and see which one best suits the distinct datasets.

Features of our program include:

a. Update Password

Allows the admin to either manually or automatically (using the program) update the initial/given password.

b. Delete Password

Enables the admin to delete their password.

c. (View) Saved Passwords

Facilitates the admin to access the stored passwords.

II. Specifications

A. Data Structures Used

1. Doubly Linked List
2. Hash Map
3. Array List
4. Binary Search Tree (BST)

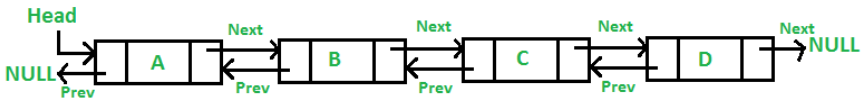
B. Software/Library Used

1. IntelliJ IDE
2. nanoTime( )

III. Solution Implementation

A. Doubly Linked List

Doubly Linked Lists contain nodes that are linked to the previous and next data field. What makes a singly linked list and a doubly linked list different is that the singly linked lists don't have a link to the previous data field, therefore traversals can't be done to the previous node.



B. Hashmap

Hashmap stores data in key-value pairs and you can access them by an index of another type (e.g. Integer).

C. Array List

Array List is a resizable array and unlike an array, elements can be added and removed easily as you need to make a new array to conduct the mentioned functions. But, data needs to be shifted in order to update the elements.

D. Binary Search Tree

A Binary Search Tree is a special type of data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree. There must be no duplicate nodes.

E. Proof of Working Application

1. Doubly Linked List

Update Password

```
--------->  Main Menu  <-------
=====[Doubly Linked List]=====
[1]- Update Password
[2]- Delete Password
[3]- Saved Passwords
=========[Array List]=========
[4]- Update Password
[5]- Delete Password
[6]- Saved Passwords
==========[Hash Map]==========
[7]- Update Password
[8]- Delete Password
[9]- Saved Passwords
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
============================
[0]- Exit System
----------------------------
Choice: 1
Enter Service Name:
test1
Enter Username:
test_1
```

```
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 1

Enter Password: h3lloth3r3*


Press any key to return

/
```

(View) Saved Passwords

```
----------------------------
Choice: 3
| Service  | Username | Password |
--------------------------------
| test1    | test_1   | h3lloth3r3* |
| test2    | test_2   | O#Qk7Bb0y! |
| test3    | test_3   | M7A9p*OeT$ |
| test4    | test_4   | Xv5g!Jd#U% |
| test5    | test_5   | #G1lU9z&Nw |
| test6    | test_6   | Y4H6n&Gx*J |
| test7    | test_7   | C@2b1J9Fw%D |
| test8    | test_8   | *Rw3o7Mh&J |
| test9    | test_9   | H1z6&Vw#O$ |
| test10   | test_10  | D9u@1jPx!K |
| test11   | test_11  | S$5fL1j#Ku |
| test12   | test_12  | L6Q7j&Dx!Z |
| test13   | test_13  | *Y8h!Z1q%P |
| test14   | test_14  | F6P&3zKd!J |
| test15   | test_15  | @T2fU1b$Op |
```

Delete Password

```
------------------------------
Choice: 2
==> DELETE <==
Enter Service Name:
test1
Enter Username:
test_1
Password successfully deleted!

Press any key to return

/
```

```
------------------------------
Choice: 3
| Service  | Username | Password |
---------------------------------
| test2    | test_2   | O#Qk7Bb0y! |
| test3    | test_3   | M7A9p*OeT$ |
| test4    | test_4   | Xv5g!Jd#U% |
| test5    | test_5   | #G1lU9z&Nw |
| test6    | test_6   | Y4H6n&Gx*J |
| test7    | test_7   | C@2b1J9Fw%D |
| test8    | test_8   | *Rw3o7Mh&J |
```

2. Hash Map

Update Password

```
------------------------------
Choice: 7
Enter Service Name:
test2
Enter Username:
test_2
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 2

Enter Length of you Password
8

Press any key to return

/
```

(View) Saved Passwords

```
----------------------------
Choice: 3
| Service  | Username | Password |
---------------------------------
| test183  | test_183 | K6I@1yXn#D |
| test2    | test_2   | 0Ds8UNW) |
| test152  | test_152 | C9J@2dHf#N |
| test57   | test_57  | F5V#8cMh!D |
| test74   | test_74  | Z7J#9tLn&D |
| test179  | test_179 | G3O@9cJk&N |
| test66   | test_66  | L8I#1tXj!K |
| test148  | test_148 | W3S@2xPn&H |
| test121  | test_121 | Z2S#1tQk&N |
| test158  | test_158 | E2U#4lGn!D |
| test127  | test_127 | O3D#8pXn&J |
| test100  | test_100 | I3M#7lJt&H |
```

Delete Password

```
==========[Hash Map]==========
[7]- Update Password
[8]- Delete Password
[9]- Saved Passwords
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
============================
[0]- Exit System
----------------------------
Choice: 9
| test183  | test_183 | K6I@1yXn#D |
| test152  | test_152 | C9J@2dHf#N |
| test57   | test_57  | F5V#8cMh!D |
| test74   | test_74  | Z7J#9tLn&D |
| test179  | test_179 | G3O@9cJk&N |
| test66   | test_66  | L8I#1tXj!K |
| test148  | test_148 | W3S@2xPn&H |
| test121  | test_121 | Z2S#1tQk&N |
| test158  | test_158 | E2U#4lGn!D |
```

3. Array List

Update Password

```
------------------------------
Choice: 4
Enter Service Name:
test3
Enter Username:
test_3
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 1

Enter Password: b1ngb0ng123


Press any key to return

/
```

(View) Saved Passwords

```
=========[Array List]=========
[4]- Update Password
[5]- Delete Password
[6]- Saved Passwords
==========[Hash Map]==========
[7]- Update Password
[8]- Delete Password
[9]- Saved Passwords
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
============================
[0]- Exit System
----------------------------
Choice: 6
```

```
| test40   | test_40  | T1D#3xJm&N |
| test22   | test_22  | E1Q5&Gj#S$ |
| test88   | test_88  | E5Y#6vDn!K |
| test104  | test_104 | J3C#8yKn&D |
| test43   | test_43  | F6X#9cGp&J |
| test27   | test_27  | R5D#6nWb&Y |
| test32   | test_32  | T4E#2nXd!K |
| test3    | test_3   | b1ngb0ng123 |
| test103  | test_103 | R9Y@5gBh#N |
| test98   | test_98  | S6G@1tXn#B |
| test61   | test_61  | K5I@1zQw#J |
| test53   | test_53  | A8C#9zNk&G |
| test35   | test_35  | Q7B!N9j#Gd |
```

Delete Password

```
------------------------------
Choice: 5
==> DELETE <==
Enter Service Name:
test3
Enter Username:
test_3
Password successfully deleted!

Press any key to return

/
```

4. Binary Search Tree (BST)

Update Password

```
-----------------------------
Choice: 10
Enter Service Name:
test4
Enter Username:
test_4
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 2

Enter Length of you Password
8

Press any key to return

/
```

(View) Saved Passwords

```
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
============================
[0]- Exit System
----------------------------
Choice: 12
```

```
| test47   | test_47  | J9L#1hBx&N |
| test48   | test_48  | Y5W!6bXn#Q |
| test49   | test_49  | E7U#9mZd!H |
| test4    | test_4   | 3SV8m+rC |
| test50   | test_50  | D9V@3zOn#J |
| test51   | test_51  | L1B#6jNk&D |
| test52   | test_52  | P5H!2tBx#N |
| test53   | test_53  | A8C#9zNk&G |
| test54   | test_54  | S4W!6kQn&H |
| test55   | test_55  | C9X#1dGp!K |
| test56   | test_56  | N6Z@4qKp!H |
```

Delete Password

```
Choice: 11
==> DELETE <==
Enter Service Name:
test4
Enter Username:
test_4
Password successfully deleted!

Press any key to return
```

IV. Appendix & Conclusion

A. Program Manual

```
--------->  Main Menu  <-------
=====[Doubly Linked List]=====
[1]- Update Password
[2]- Delete Password
[3]- Saved Passwords
=========[Array List]=========
[4]- Update Password
[5]- Delete Password
[6]- Saved Passwords
==========[Hash Map]==========
[7]- Update Password
[8]- Delete Password
[9]- Saved Passwords
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
==============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
==============================
[0]- Exit System
------------------------------
Choice: 1
```

When you run the program, the user needs to input the password in order to get through the program. If the password is accurate, the main menu will be printed in the command line and it'll display all of the choices that the user can choose from. 1 to 3 is for the doubly linked list, 4 to 6 is for the arraylist, 7 to 9 is for the hash map, 10 to 12 is for the binary search tree. For the last set of choices, it's used for the benchmarking.

```
Choice: 1
Enter Service Name:
test1
Enter Username:
test_1
```

For the update feature, let's say we choose choice 1 in this case, the program will print out 'enter service name' and 'enter username.' This way, the program can retrieve this data to search it through our file storage system in 'passwordDB.txt.'

```
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 1

Enter Password: h3lloth3r3*


Press any key to return

/
```

```
Enter Service Name:
test4
Enter Username:
test_4
Data Found
[1] To change password manually
[2] To change generate new password
Choice: 2

Enter Length of you Password
8

Press any key to return
```

When data matches with the one in the ‘txt’ file, the program will print out two choices whether to change the password manually or randomly using the program. If the user chooses 1, the user is able to enter a password to their liking. But if the user chooses 2, the program prints out ‘enter length of your password’ and the user is able to set a length for their password.

```
-----------------------------
Choice: 5
==> DELETE <==
Enter Service Name:
test3
Enter Username:
test_3
Password successfully deleted!

Press any key to return

/
```

For the delete feature, the program prints ‘enter service name’ and ‘enter username’ again for the user input. If data is found, then the password will be deleted and removed from the file storage system. If no match is found, the program will not go through.

```
=====[Binary Search Tree]=====
[10]- Update Password
[11]- Delete Password
[12]- Saved Passwords
============================
[13]- Compare Update Password Data types
[14]- Compare Delete Password data types
[15]- Compare View Saved Passwords
============================
[0]- Exit System
----------------------------
Choice: 12
```

```
| test47   | test_47  | J9L#1hBx&N |
| test48   | test_48  | Y5W!6bXn#Q |
| test49   | test_49  | E7U#9mZd!H |
| test4    | test_4   | 3SV8m+rC |
| test50   | test_50  | D9V@3zOn#J |
| test51   | test_51  | L1B#6jNk&D |
| test52   | test_52  | P5H!2tBx#N |
| test53   | test_53  | A8C#9zNk&G |
| test54   | test_54  | S4W!6kQn&H |
| test55   | test_55  | C9X#1dGp!K |
| test56   | test_56  | N6Z@4qKp!H |
```

For the view saved passwords feature, simply input the choice from the main menu and the entire file system will be iterated and printed out in the command line in a somewhat table-like form.

B. Measurement of Time Complexity

1. Update Password

Data: 100	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	25	16	13	11	15	14	13	10	7	16	14
Hash Map	10	10	9	7	9	8	8	7	4	9	8.1
Array List	11	9	8	8	9	8	9	8	7	9	8.6
Binary Search Tree	2	1	0	0	1	0	0	0	0	0	0.4

Data: 150	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	25	21	19	13	17	17	17	21	18	20	18.8
Hash Map	14	11	12	10	13	11	11	13	13	12	12
Array List	15	10	12	10	12	11	12	13	13	10	11.8
Binary Search Tree	2	1	1	0	0	0	0	0	0	0	0.4

Data: 200	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	36	25	21	23	24	23	22	22	23	33	25.2
Hash Map	18	16	14	14	14	16	13	14	13	22	15.4
Array List	20	16	15	15	15	15	13	14	11	19	15.3
Binary Search Tree	2	1	1	1	0	0	0	0	0	9	0.9

2. Delete Password

Data: 100	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	27	17	14	12	15	14	13	14	13	13	15.2
Hash Map	10	9	8	7	9	8	9	9	9	7	8.5
Array List	11	10	8	7	8	7	9	8	8	8	8.4
Binary Search Tree	2	1	0	0	0	0	0	0	0	0	0.3

Data: 150	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	33	22	17	20	16	19	20	16	19	17	19.9
Hash Map	15	12	10	13	11	13	11	12	11	10	11.8
Array List	16	11	11	12	11	12	12	12	11	10	11.8
Binary Search Tree	1	0	1	0	0	0	0	0	0	0	0.2

Data: 200	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	35	26	26	14	26	19	23	23	24	21	23.7
Hash Map	18	15	15	13	15	13	13	13	15	12	14.2
Array List	20	15	15	14	15	13	13	14	13	10	14.2
Binary Search Tree	2	1	1	0	1	0	0	0	1	0	0.6

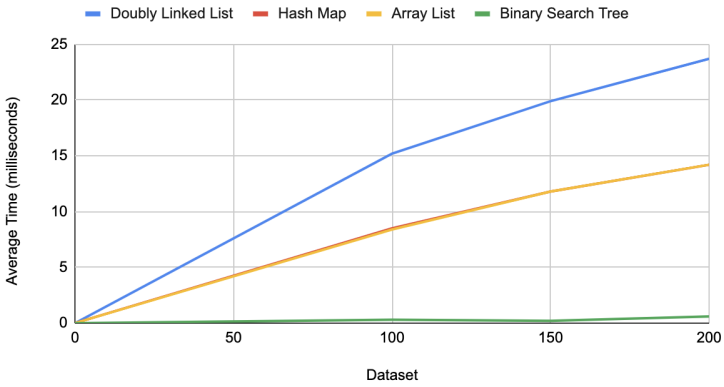
3. Saved Passwords

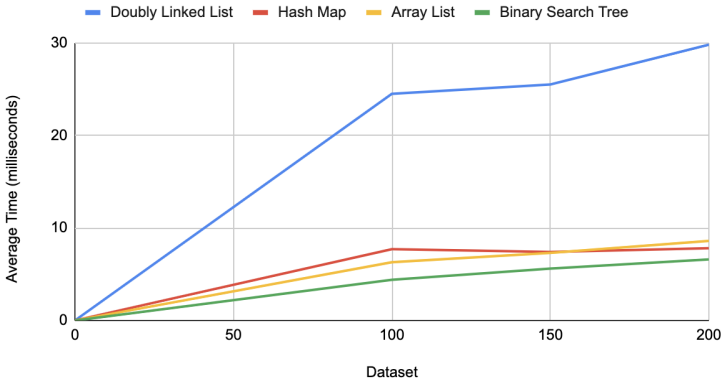
Data: 100	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	26	25	23	25	24	24	25	25	27	24	24.5
Hash Map	7	8	8	7	7	9	9	7	8	7	7.7
Array List	6	6	7	6	6	7	6	6	7	6	6.3
Binary Search Tree	5	4	5	4	5	5	4	4	4	4	4.4

Data: 150	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	25	28	27	24	24	25	26	28	25	23	25.5
Hash Map	7	8	8	7	7	7	7	7	9	7	7.4
Array List	8	7	8	8	7	7	7	7	7	7	7.3
Binary Search Tree	5	6	6	8	5	5	5	5	6	5	5.6

Data: 200	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	32	30	29	31	29	29	28	33	28	29	29.8
Hash Map	8	7	8	7	10	7	9	8	7	7	7.8
Array List	9	8	8	9	9	9	8	10	8	8	8.6
Binary Search Tree	7	7	6	6	6	7	6	6	8	7	6.6

In the line graphs below, we compiled the average time in milliseconds for each of the data structures and features.





The trend of the efficiency of our code lies in the line graphs above; as the datasets grow larger, all data structures collectively take longer to complete the task at hand. However, if we were to view these data structures individually, the doubly linked list seems to take the longest time amongst the data structures as well as across the three features, and the binary search tree is the most reliable in this scenario.

C. How we Calculated our Findings

In order to discover which data structure is most efficient and least efficient for each scenario of our password management system, we used the nanoTime() method. In order to gain a precise and accurate time, we use the nanoTime() method, which returns the current time with the most precise system timer available, in nanoseconds. Let's take a look at the example using one of the features in our password management system, "View Saved Password."

```
public void compareDataType(){
    long timeListStart=System.nanoTime();
    displayList();
    long timeListExce=(System.nanoTime()-timeListStart)/1000000;
    long timeArrayStart=System.nanoTime();
    displayFromArraylist();
    long timeArrayExce=(System.nanoTime()-timeArrayStart)/1000000;
    long timeHashmapStart=System.nanoTime();
    displayFromHashmap();
    long timeHashmapExce=(System.nanoTime()-timeHashmapStart)/1000000;
    long timeBSTStart=System.nanoTime();
    displayFromBST();
    long timeBSTExce=(System.nanoTime()-timeBSTStart)/1000000;

    System.out.println("Double Linked List View: "+ timeListExce);
    System.out.println("Array List View: "+timeArrayExce);
    System.out.println("Hashmap View:  "+ timeHashmapExce);
    System.out.println("BST View:  "+ timeBSTExce);
```

The code above shows different data structures that appear to be comparing each other's execution times. The method is about measuring the time it takes to show the elements from a doubly linked list, an array list, a hashmap, and a binary search tree.

Started by calling the displayList() method and recording the start time of timeListStart using System.nanoTime(), this method is starting to measure the execution time for displaying elements from doubly linked lists until the binary search tree. It calculates the execution time by deducting the start time from the current time using System.nanoTime() after displaying the elements from the doubly linked list. Once more, nanoTime() divides the nanosecond value by 1,000,000 to obtain the millisecond value, and TimeListExce will keep the outcome.

The same procedures are repeated in order to display items from an array list, hash map, and binary search tree, respectively. Each data structure's execution times are saved in its corresponding variable.

That is an illustration of how one of our features, “View Saved Password," can be used to obtain time complexity. Other capabilities,such as the Update and Delete password features, differ slightly from the View Saved Password feature in that they require input. This can impact the search time for time-complexity. The user's speed when typing data, large or small data volumes, and hardware efficiency can all affect how time-consuming this feature is. Why is that? Here's a little sample of our code (full code can be seen on GitHub):

```
public void compareDelete(){
    //Input Data
    System.out.println("==> DELETE <==");

    Scanner input = new Scanner(System.in);
    String ServiceName, UserName;

    System.out.println("Enter Service Name:");
    ServiceName = input.nextLine();

    System.out.println("Enter Username:");
    UserName = input.nextLine();
```

The messages "Enter Service Name:" and "Enter Username:" in the code prompt the user to enter the appropriate name according to the specifications. The user also has to wait before entering values into the application and pressing the Enter key.

Furthermore, this can affect time complexity because, after the user presses the key, the programs need to check if the entered service name and username are on the list. If the data written by the user is in the list, then the "deletion" action runs to delete the password. If there is no specific data on the list, the program recalls the first menu screen. We can see this code at a glance.

```
//Double Linked List
long searchDoubleLinkedTimeStart=System.nanoTime();
if (first == null) {
    System.out.println("The list is empty");
    return;
}

if (ServiceName.equals(first.serviceName) && UserName.equals(first.userName)) {
    first = first.next;
} else if (ServiceName.equals(last.serviceName) && UserName.equals(last.userName)) {
    last.previous.next = null;
    last = last.previous;
} else {
    NodePassword current = first.next;
    while (current != null) {
        if (ServiceName.equals(current.serviceName) && UserName.equals(current.userName)) {
            current.previous.next = current.next;
            current.next.previous = current.previous;
            break;
        }
        current = current.next;
    }
}
long searchDoubleLinkedDuration=System.nanoTime()-searchDoubleLinkedTimeStart;
```

```
long deleteDoubleLinkedStart=System.nanoTime();
PrintWriter printWriter = null;
try {
    printWriter = new PrintWriter( fileName: "passwordDB.txt");
    printWriter.print("");
    printWriter.close();
} catch (FileNotFoundException e) {
    System.out.println("File not found");
    e.printStackTrace();
}

NodePassword current = first;
while (current != null) {
    String data = current.serviceName + "<->" + current.userName + "<->" + current.password + "\n";

    File file = new File( pathname: "passwordDB.txt");
    FileWriter fileWriter = null;
    try {
        fileWriter = new FileWriter(file,  append: true);
        fileWriter.write(data);
        fileWriter.close();
    } catch (IOException e) {
        System.out.println("Error");
        throw new RuntimeException(e);
    }
    current = current.next;
}
System.out.println("Password successfully deleted!");
long deleteDoubleLinkedDuration=(System.nanoTime()-deleteDoubleLinkedStart)/1000000;
```

Following that, the same methodology is used for additional data structures; however, the implementation varies depending on the method of each data structure. The method for determining time complexity is the same; the only difference is that there are units for milliseconds as well as nanoseconds (depending on whether the time is too quick or too slow). This also occurs in the Update Password function, which likewise includes an input mechanism for use.

In theory, we can conclude that there is a best and worst time complexity, in which each data structure has different results. The theory can be seen as follows:

- Doubly Linked List:

  Best case: O(1), Worst case: O(n)

- Array List:

  Best case: O(1), Worst case: O(1)

- Hash Map:

  Best case: O(1), Worst case: O(n)

- Binary Search Tree:

  Best case: O(log n), Worst case: O(n)

The relationship between the code and the theory mentioned earlier shows that different execution times of data structures can give you a characteristic indicator of time complexity in general. The theoretical expectations given by the best and worst time complexities mentioned above may differ from the performance complemented above, depending on implementation specifications, data volume, and hardware efficiency.

D. What We Learnt

After comparing all four data structures in this research, it turns out that the fastest algorithm known to perform better in our case is the binary search tree. Following Binary Search Tree are Hashmap and Arraylist (both are almost similar in all aspects of the test, and both are using Java libraries), and at the last place is Doubly Linked List.

E. Conclusion

In some cases, in other research, binary trees might come in second when compared to other algorithms like hash maps, but in our case, it turns out that the fastest algorithm is the binary search tree compared to arraylist, doubly linked list, and hashmap. This is caused by some factors and characteristics in our data that favor the binary search tree. The first one is the nature of the data that is ordered when it is first inserted into the binary search tree. Binary Search Tree data structures function better when the datasets are ordered and the performed actions are order-related. Another factor that favors binary search trees is that they are good at dynamic operations, namely delete operations and insert operations (in our case, the deletion operation). If we take hash map as an example for comparison, in hash map we still need to take care of collisions, which can lead to increased overhead and potential rehashing operations. BSTs, on the other hand, provide good performance for dynamic operations without requiring additional steps to handle collisions.