Data Structures:
Final Project

Password Management System

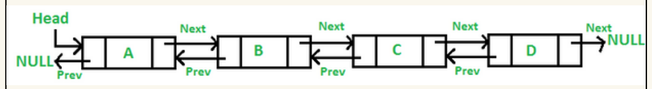
Made by Brigitte, Ivan, and Tirza of L2AC

Introduction

- What is a Password Management System?
- What problem are we solving?
  - To implement a Java password management system that lets the admin securely store, change, delete, and see passwords. The system should restrict password access to the admin, making password management secure and dependable.
- What features does it provide?
  - Update Password
  - Delete Passwords
  - View Saved Passwords

Types of Data Structures

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.

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.

Hash Map

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

Binary Search Tree (BST)

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.

#1
Doubly Linked List

NodePassword Class

```java
// class for 1 node, to be used in doubly linked list data structure
17 usages
public class NodePassword {
    15 usages
    public String serviceName;
    15 usages
    public String userName;
    10 usages
    public String password;
    23 usages
    public NodePassword next;
    12 usages
    public NodePassword previous;

    1 usage
    public void displayNode() {

        System.out.printf("| %-8s | %-8s | %-8s |\n", serviceName, userName, password);
    }

}
```

```java
NodePassword current = first;

while (current != null && flag == 0) {
    if (sn.equals(current.serviceName) && un.equals((current.userName))) {
        System.out.println("Data Found");


        System.out.println("[1] To change password manually");
        System.out.println("[2] To change generate new password");
        String pass = "";
        short choice;
        do {
            System.out.print("Choice: ");
            choice = input.nextShort();
            switch (choice) {
                case 1:
                    System.out.println();
                    input = new Scanner(System.in); // refresh scanner to avoid errors
                    System.out.print("Enter Password: ");
                    pass = input.nextLine();
                    current.password = pass;
                    System.out.println();
                    break;
                case 2:
                    int l;
                    do {
                        System.out.println();
                        input = new Scanner(System.in); // refresh scanner to avoid errors
                        System.out.println("Enter Length of you Password");
                        l = input.nextInt();
                        if (l < 8) {
                            System.out.println("Minimum Password length = 8");
                        }
                    } while (l < 8);
                     pass = passwordGenerator.generatePassword(l);
                    current.password = pass;
                    break;
                default:
                    System.out.println("ERROR: Choice not valid");
            }
        } while (choice < 1 || choice > 2);
        flag = 1;
    }
    current = current.next;
}
```

```java
for(int i = 0; i < PasswordArrayList.size(); i++) {
    if (sn.equals(PasswordArrayList.get(i).getServiceName()) && un.equals((PasswordArrayList.get(i).getUserName()))) {
        PasswordObj newObj = PasswordArrayList.get(i);
        System.out.println("Data Found");
        System.out.println("[1] To change password manually");
        System.out.println("[2] To change generate new password");
        String pass = "";
        short choice;
        do {
            System.out.print("Choice: ");
            choice = input.nextShort();
            switch (choice) {
                case 1:
                    System.out.println();
                    input = new Scanner(System.in); // refresh scanner to avoid errors
                    System.out.print("Enter Password: ");
                    pass = input.nextLine();
                    newObj.setPassword(pass);
                    PasswordArrayList.set(i, newObj) ;
                    System.out.println();
                    break;
                case 2:
                    int l;
                    do {
                        System.out.println();
                        input = new Scanner(System.in); // refresh scanner to avoid errors
                        System.out.println("Enter Length of you Password");
                        l = input.nextInt();
                        if (l < 8) {
                            System.out.println("Minimum Password length = 8");
                        }
                    } while (l < 8);
                    pass = passwordGenerator.generatePassword(l);
                    newObj.setPassword(pass);
                    PasswordArrayList.set(i, newObj) ;
                    break;
                default:
                    System.out.println("ERROR: Choice not valid");
            }
        } while (choice < 1 || choice > 2);
        flag = 1;
        break;
    }
```

#2
Array List

#3
Hash Map

```java
if(PasswordHashmap.containsKey(sn + "<->" + un)) {
    System.out.println("Data Found");
    System.out.println("[1] To change password manually");
    System.out.println("[2] To change generate new password");
    String pass = "";
    short choice;
    do {
        System.out.print("Choice: ");
        choice = input.nextShort();
        switch (choice) {
            case 1:
                System.out.println();
                input = new Scanner(System.in); // refresh scanner to avoid errors
                System.out.print("Enter Password: ");
                pass = input.nextLine();
                PasswordHashmap.replace(sn + "<->" + un, pass);
                System.out.println();
                break;
            case 2:
                int l;
                do {
                    System.out.println();
                    input = new Scanner(System.in); // refresh scanner to avoid errors
                    System.out.println("Enter Length of you Password");
                    l = input.nextInt();
                    if (l < 8) {
                        System.out.println("Minimum Password length = 8");
                    }
                } while (l < 8);
                pass = passwordGenerator.generatePassword(l);
                PasswordHashmap.replace(sn + "<->" + un, pass);
                break;
            default:
                System.out.println("ERROR: Choice not valid");
        }
    } while (choice < 1 || choice > 2);
    flag = 1;
}
```

#4
Binary Search Tree (BST)

BSTNodePassword Class

```java
public class BSTNodePassword {
    11 usages
    String key;
    8 usages
    String value;
    13 usages
    BSTNodePassword left;
    14 usages
    BSTNodePassword right;

    1 usage
    public BSTNodePassword(String key, String value) {
        this.key = key;
        this.value = value;
        left = null;
        right = null;
    }
}
```

```java
4 usages
public void insert(String key, String value) {
    root = insertNode(root, key, value);
}

3 usages
private BSTNodePassword insertNode(BSTNodePassword root, String key, String value) {
    if (root == null) {
        return new BSTNodePassword(key, value);
    }

    int compareResult = key.compareTo(root.key);
    if (compareResult < 0) {
        root.left = insertNode(root.left, key, value);
    } else if (compareResult > 0) {
        root.right = insertNode(root.right, key, value);
    } else {
        // Update value if the key already exists
        root.value = value;
    }

    return root;
}
```

```java
4 usages
public String search(String key) {
    BSTNodePassword node = searchNode(root, key);
    return (node != null) ? node.value : null;
}

3 usages
private BSTNodePassword searchNode(BSTNodePassword root, String key) {
    if (root == null || root.key.equals(key)) {
        return root;
    }
    int compareResult = key.compareTo(root.key);
    if (compareResult < 0) {
        return searchNode(root.left, key);
    } else {
        return searchNode(root.right, key);
    }
}
```

```java
if(PasswordBST.search( key: sn + "<->" + un) != null) {
    System.out.println("Data Found");
    System.out.println("[1] To change password manually");
    System.out.println("[2] To change generate new password");
    String pass = "";
    short choice;
    do {
        System.out.print("Choice: ");
        choice = input.nextShort();
        switch (choice) {
            case 1:
                System.out.println();
                input = new Scanner(System.in); // refresh scanner to avoid errors
                System.out.print("Enter Password: ");
                pass = input.nextLine();
                PasswordBST.insert(sn + "<->" + un, pass);
                System.out.println();
                break;
            case 2:
                int l;
                do {
                    System.out.println();
                    input = new Scanner(System.in); // refresh scanner to avoid errors
                    System.out.println("Enter Length of you Password");
                    l = input.nextInt();
                    if (l < 8) {
                        System.out.println("Minimum Password length = 8");
                    }
                } while (l < 8);
                pass = passwordGenerator.generatePassword(l);
                PasswordBST.insert(sn + "<->" + un, pass);
                break;
            default:
                System.out.println("ERROR: Choice not valid");
        }
    } while (choice < 1 || choice > 2);
    flag = 1;
}
```

#1 Time Complexity using the Save section (no input)

```java
// Time Complexity to compare each data structures wit save feature.

1 usage
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);

}
```

Time Complexity

- To compare whose data structures are the fastest of the four that we use.
- 3 ways how to compare them.

#2 Time Complexity using the Delete section Including Searching (input)

```java
// Time Complexity to compare each data structures in delete section
1 usage
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();
```

#DoublyLinkedList

```java
//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;
```

Time Complexity

```
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;
```

Time Complexity

#ArrayList

```
    //ArrayList

    long searchArrayListTimeStart=System.nanoTime();
    if (PasswordArrayList.size() < 0 || PasswordArrayList.size() == 0) {
        System.out.println("The list is empty");
        return;
    }

    long searchArrayListDuration=System.nanoTime()-searchArrayListTimeStart;
    // delete
    long deletionArrayStart=System.nanoTime();
    for(int i = 0; i < PasswordArrayList.size(); i++) {
        if (ServiceName.equals(PasswordArrayList.get(i).getServiceName()) && UserName.equals((PasswordArrayList.get(i).getUserName()))) {
            PasswordArrayList.remove(i);
            break;
        }
    }
```

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

for(int i = 0; i < PasswordArrayList.size(); i++) {
    String data = PasswordArrayList.get(i).getServiceName() + "<->" + PasswordArrayList.get(i).getUserName() + "<->" + PasswordArrayList.get(i).getPassword() + "\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);
    }
}
System.out.println("Password successfully deleted!");
long deletionArrayDuration=(System.nanoTime()-deletionArrayStart)/1000000;
```

Time Complexity

#3 Time Complexity using the Update section Including Searching (input)

#HashMap

```
//HashMap
sc = new Scanner(System.in); // Clearing Input Buffer
long hashMapUpdateStart=System.nanoTime();
flag = 0;

if(PasswordHashmap.containsKey(sn + "<->" + un)) {
    System.out.println("Data Found");
    int m=16;
    String pass = "";
    pass = passwordGenerator.generatePassword(m);
    PasswordHashmap.replace(sn + "<->" + un, pass);
    flag = 1;
}

if (flag == 0) {
    System.out.println("Data does not exist!");
} else {
    PrintWriter writer = null;
    try {
        writer = new PrintWriter( fileName: "passwordDB.txt");
        writer.print("");
        writer.close();
    } catch (FileNotFoundException e) {
        System.out.println("File not found");
        e.printStackTrace();
    }


  String[] keyArr;
  for (Map.Entry<String, String> set : PasswordHashmap.entrySet()) {
      // split key
      keyArr = set.getKey().split( regex: "<->");
      String data = keyArr[0] + "<->" + keyArr[1] + "<->" + set.getValue() + "\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);
      }
  }
}
long hashMapUpdateDuration=(System.nanoTime()-hashMapUpdateStart)/1000000;
```

Time Complexity

#BinarySearchTree

```
///BST
sc = new Scanner(System.in); // Clearing Input Buffer
long bSTUpdateStart=System.nanoTime();
flag = 0;

if(PasswordBST.search( key: sn + "<->" + un) != null) {
    System.out.println("Data Found");
    int n=16;

    String pass = "";
    pass = passwordGenerator.generatePassword(n);
    PasswordBST.insert(sn + "<->" + un, pass);
    flag = 1;
}

if (flag == 0) {
    System.out.println("Data does not exist!");
} else {
    PrintWriter writer = null;
    try {
        writer = new PrintWriter( fileName: "passwordDB.txt");
        writer.print("");
        writer.close();
    } catch (FileNotFoundException e) {
        System.out.println("File not found");
        e.printStackTrace();
    }

    // save to textFile
    PasswordBST.saveTree();
}
long bSTUpdateDuration=(System.nanoTime()-bSTUpdateStart)/1000000;
```

Time Complexity

Results

Update Passwords

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

Results

Delete Passwords

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

Results

(View) Saved Passwords

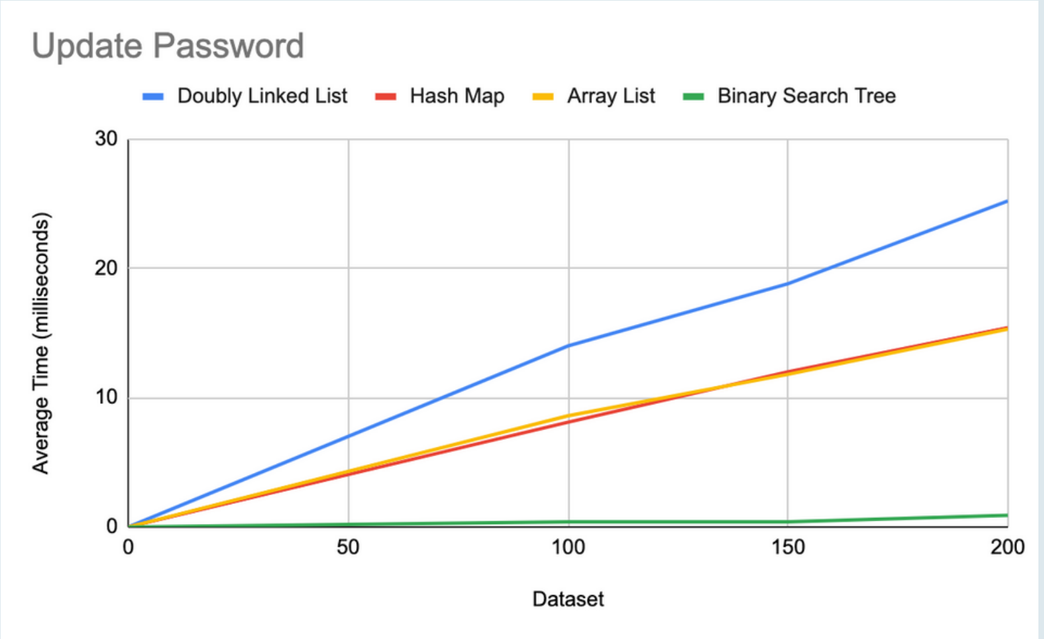
3. Saved Passwords

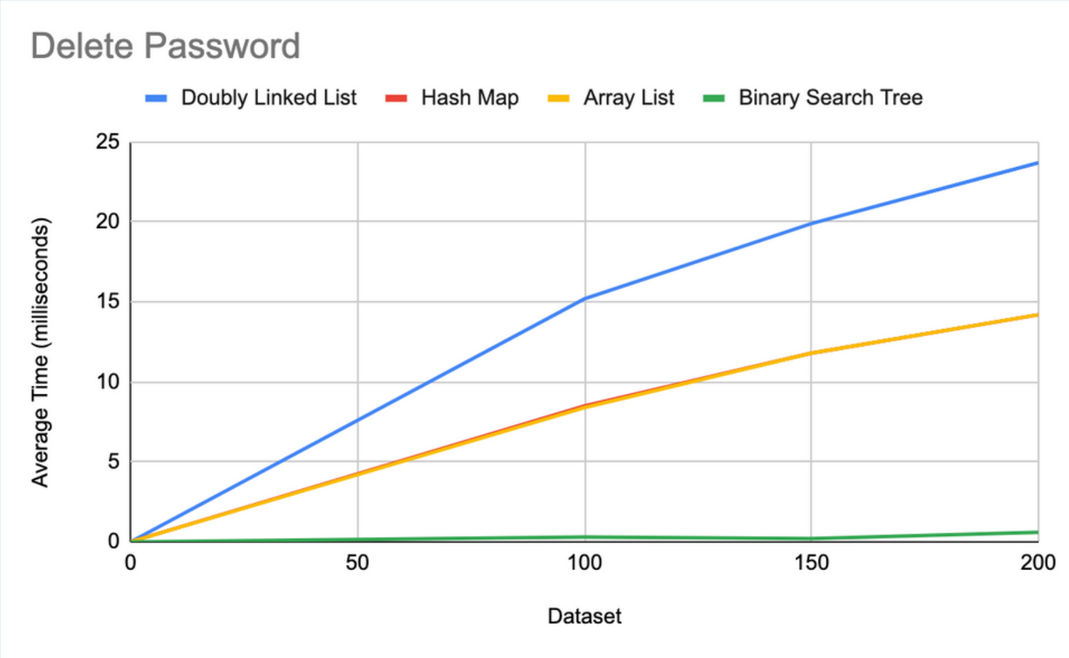
Data: 100	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	26	25	23	25	24	24	25	25	27	24	24.5
Hash Map	7	8	8	7	7	9	9	7	8	7	7.7
Array List	6	6	7	6	6	7	6	6	7	6	6.3
Binary Search Tree	5	4	5	4	5	5	4	4	4	4	4.4

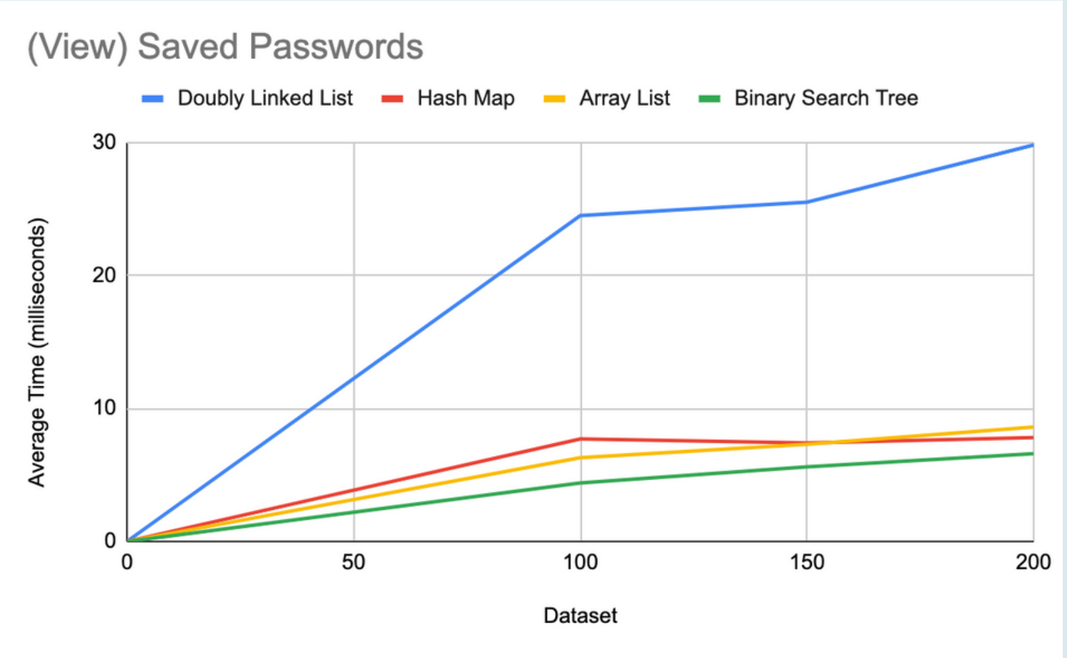
Data: 150	Time (milliseconds)										Avg
Attempt Count	1	2	3	4	5	6	7	8	9	10	
Doubly Linked List	25	28	27	24	24	25	26	28	25	23	25.5
Hash Map	7	8	8	7	7	7	7	7	9	7	7.4
Array List	8	7	8	8	7	7	7	7	7	7	7.3
Binary Search Tree	5	6	6	8	5	5	5	5	6	5	5.6

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

Results
Analysis: Graph







Conclusion

According to our research, we discovered that the most efficient data structure for our project is the **binary search tree (BST)** and the least efficient is the **doubly linked list.** In both cases where datasets are considered small and large in our analysis, BST is constantly the most reliable and doubly linked list is the least efficient.

References

https://www.prepbytes.com/blog/linked-list/difference-between-a-singly-linked-list-and-a-doubly-linked-list/#:~:text=A%20Singly%20Linked%20has%20nodes,link%20of%20the%20next%20node.

https://www.w3schools.com/java/java_hashmap.asp

https://www.w3schools.com/java/java_arraylist.asp

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/