University of New Brunswick

CS2383

Michael Drost
3738984

Assignment 1

1A)

Extending code

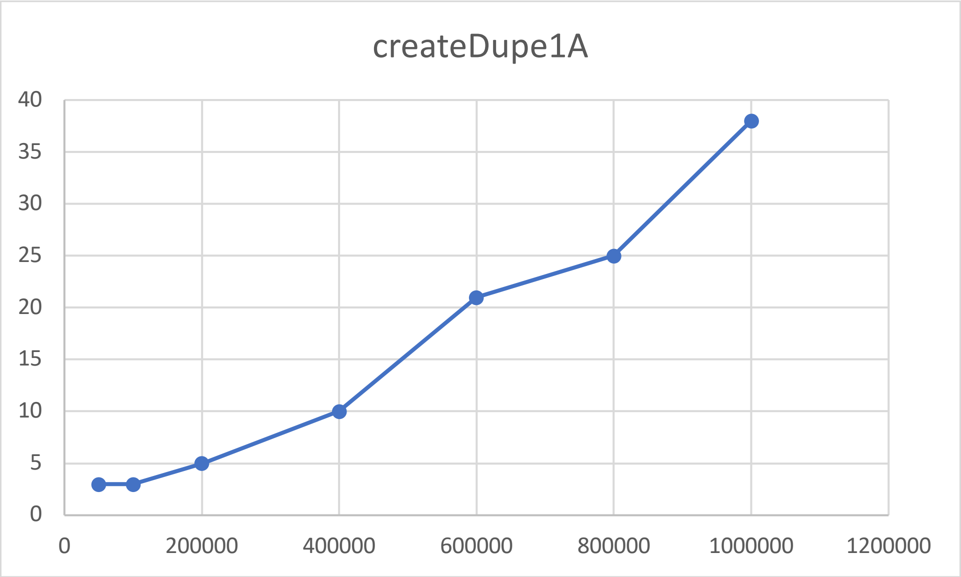
```
package As1;

public class createDupe1A extends Duplication{

    public createDupe1A(int arraySizeIn, int maxValueIn){
        super(arraySizeIn , maxValueIn);
    }

    public boolean containsDuplicate() {
        int length = array.length;
        for (int i =0;i<length;i++){
            for (int j = i+1; j<length;j++){
                if (array[i] == array[j]){
                    return true;
                }
            }
        }
        return false;

    }

}
```

Driver Code

```
package As1;

public class createDupeDriver {
    public static void main(String[] args){
        long start;
        long end;
        long time;

        start = System.currentTimeMillis();
        createDupe1A test =new createDupe1A(1000000, 8000000);
        System.out.println(test.containsDuplicate());
        end = System.currentTimeMillis();
        time = end - start;
        System.out.println(time);

    }
}
```

1A

ArraySize	time(ms)
50000	3
100000	3
200000	5
400000	10
600000	21
800000	25
1000000	38



When I first made this code I had used the break; statement and the graph showed N^2 very well, however when I removed the break statement and replaced it with just return true; the results showed less of an obvious N^2 correlation. However there is still a small relation showing the parabola in the graph.

1B)

Extended Code

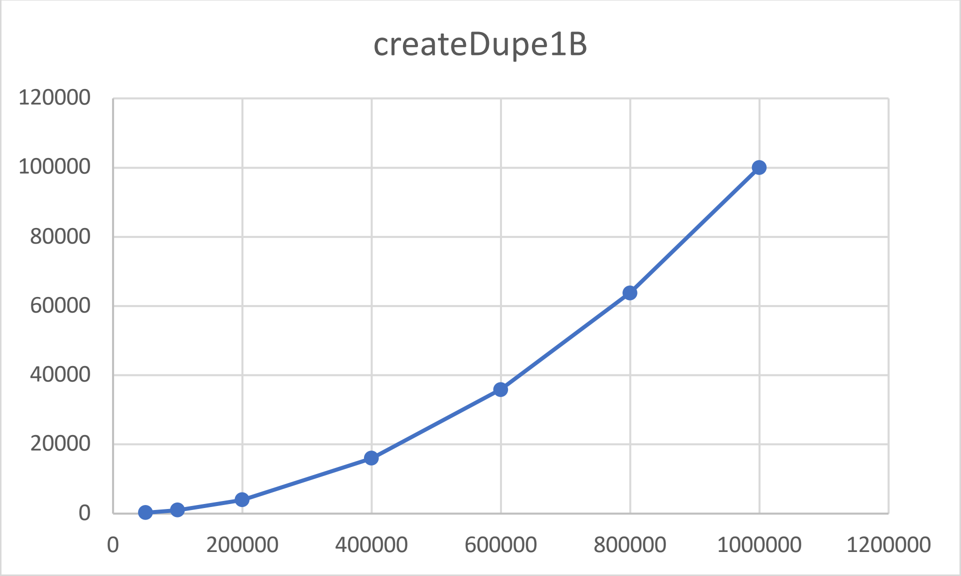
```
package As1;

public class createDupe1B extends Duplication {

    public createDupe1B(int arraySizeIn, int maxValueIn) {
        super(arraySizeIn, maxValueIn);
    }

    public boolean containsDuplicate() {
       boolean dupeFound = false;
       int length = array.length;
       for (int i =0;i<length;i++){
        for (int j = i+1; j<length;j++){
            if (array[i] == array[j]){
                dupeFound = true;
            }
        }
    }
       return dupeFound;
    }


}
```

Driver Code

```
package As1;

public class createDupeDriver {
    public static void main(String[] args){
        long start;
        long end;
        long time;

        start = System.currentTimeMillis();
        createDupe1B test =new createDupe1B(1000000, 8000000);
        System.out.println(test.containsDuplicate());
        end = System.currentTimeMillis();
        time = end - start;
        System.out.println(time);

    }


}
```

1B

ArraySize	time(ms)
50000	259
100000	1046
200000	4004
400000	16037
600000	35890
800000	63806
1000000	100031



By checking all the other values the graph followed N^2 a lot more and shows what I was looking for almost perfectly.

2)

Extended code

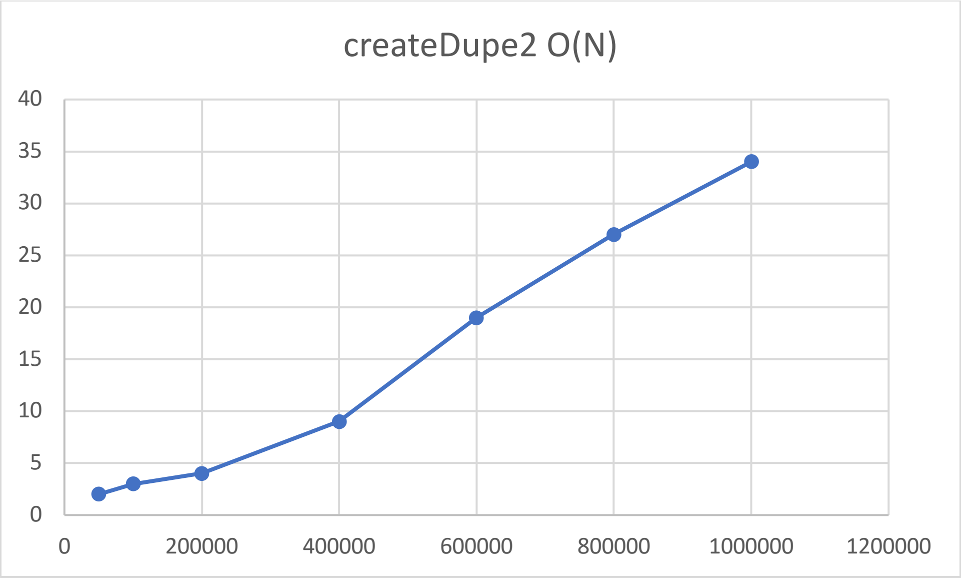
```
package As1;

public class createDupe2 extends Duplication {

    public createDupe2(int arraySizeIn, int maxValueIn) {
        super(arraySizeIn, maxValueIn);
    }


    public boolean containsDuplicate() {
       boolean dupeFound = false;

       int[] lookUP = new int[super.maxValue +1];
       for (int i =0;i<super.arraySize;i++){
        if (lookUP[super.array[i]] == 1){
            dupeFound = true;
            break;
        }else{
            lookUP[super.array[i]] = 1;
        }
       }
       return dupeFound;
    }


}
```

Driver code

```
package As1;

public class createDupeDriver {
    public static void main(String[] args){
        long start;
        long end;
        long time;

        start = System.currentTimeMillis();
        createDupe2 test =new createDupe2(1000000, 8000000);
        System.out.println(test.containsDuplicate());
        end = System.currentTimeMillis();
        time = end - start;
        System.out.println(time);

    }


}
```

O(N)

ArraySize	time(ms)
50000	2
100000	3
200000	4
400000	9
600000	19
800000	27
1000000	34

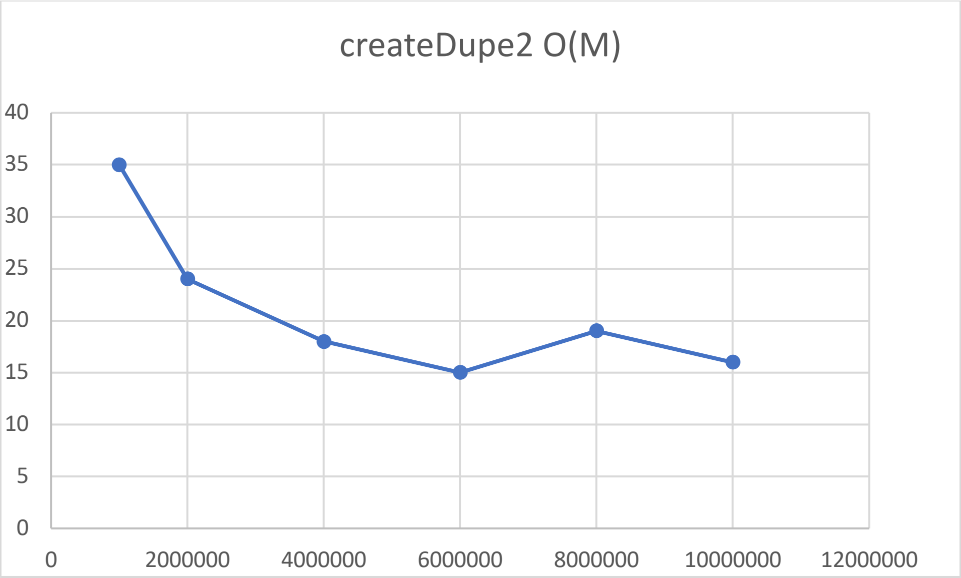


This graph shows that when N (Array size) grows the time grows too in a fairly accurate Order N relation.

O(M)

MaxValue	time
1000000	35
2000000	24
4000000	18
6000000	15
8000000	19
10000000	16

Array size
is 1000000



For this test I kept the array size at 1 million and increased the Max value, It shows that the more values in the array and the larger the lookup array is the faster the code Is run which confused me. It seems like a Log N but flipped upside down.

3)

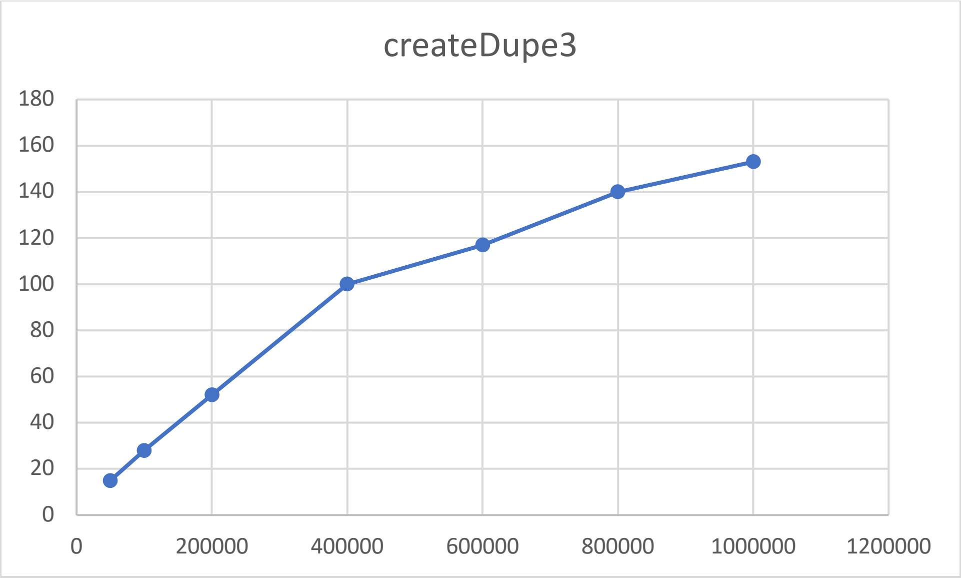
Extended code

```
package As1;

import java.util.Arrays;

public class createDupe3 extends Duplication {

    public createDupe3(int arraySizeIn, int maxValueIn) {
        super(arraySizeIn, maxValueIn);
    }

    public boolean containsDuplicate() {
       boolean dupeFound = false;
       int[] newArr = Arrays.copyOf(array, array.length);
       Arrays.sort(newArr);
       for (int i=0;i+1<newArr.length;i++){
        if (newArr[i] == newArr[i+1]){
            dupeFound =true;
            break;
        }
       }
       return dupeFound;
    }
}
```

Driver code

```
package As1;

public class createDupeDriver {
    public static void main(String[] args){
        long start;
        long end;
        long time;

        start = System.currentTimeMillis();
        createDupe3 test =new createDupe3(1000000, 8000000);
        System.out.println(test.containsDuplicate());
        end = System.currentTimeMillis();
        time = end - start;
        System.out.println(time);

    }


}
```

ArraySize	time(ms)
50000	15
100000	28
200000	52
400000	100
600000	117
800000	140
1000000	153



This graph shows the beginning of log N forming and with even more data would clearly show a Log N algorithm just as I was expecting.

4)

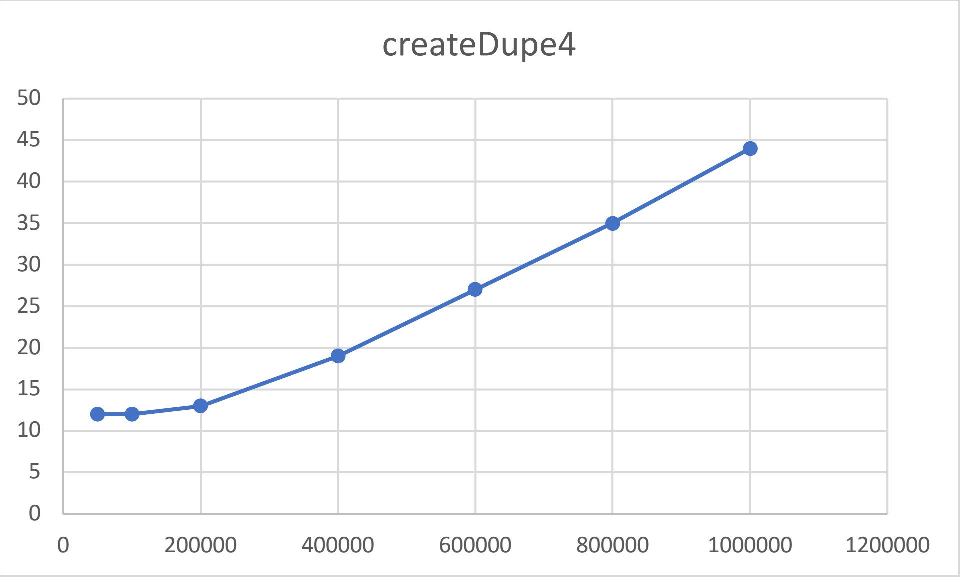
Extended code

```
package As1;

public class createDupe4 extends Duplication {

    public createDupe4(int arraySizeIn, int maxValueIn) {
        super(arraySizeIn, maxValueIn);
    }

    public boolean containsDuplicate() {
       boolean dupeFound = false;
       BinaryTree tree = new BinaryTree();
       for (int value :array)
        try{
            tree.insert(value);
        }catch(IllegalArgumentException e){
            dupeFound= true;
            return dupeFound;
        }
       return dupeFound;
    }
}
```

Driver Code.

```
package As1;

public class createDupeDriver {
    public static void main(String[] args){
        long start;
        long end;
        long time;

        start = System.currentTimeMillis();
        createDupe4 test =new createDupe4(1000000, 8000000);
        System.out.println(test.containsDuplicate());
        end = System.currentTimeMillis();
        time = end - start;
        System.out.println(time);

    }


}
```

ArraySize	time(ms)
50000	12
100000	12
200000	13
400000	19
600000	27
800000	35
1000000	44



For the 4th test we saw a scenario closer to the worst case for a binary search tree closer to N^2 however it was not as Potent as 1B was showing. With many more tests I believe the graph would have looked more like N Log (N).

Overall, The results seem fairly accurate and where I expected them to be.