CMP-5014Y Coursework Assignment 2

100108964 (`kzy14tcu`)

Wed, 20 Mar 2019 14:32

PDF prepared using PASS version 1.15 running on `Windows 10 10.0` (`amd64`).

☑ I agree that by submitting a PDF generated by PASS I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

DSAver2.pdf

 (Included PDF starts on next page.)

CMP – 5014Y Data Structures and Algorithm

Coursework Assignment 2

100108964

Exercise 1: Algorithm Design and Implementation

1. Algorithm for $O(n^2)$

Design an algorithm for solving SVD problem whose worst case run-time complexity is $O(n^2)$.

Informal algorithm

Given an Array ***a*** of ***n*** integers. Scan through array ***a***. If a value occurs more than n/2 times in this value dominates ***a***. Return this element, return false otherwise.

Formal Algorithm:

SVD_Onn(int a[],int n) scan array arrSVD[] and return majority value if exists.

```
1.  maxCount := 0
2.  index := -1
3.  for i := 0 i < n i++
4.     count:=0
5.     for j := 0 j < n j++
6.        if a[i] == a[j]
7.        count++
8.      if count > maxCount
9.         maxCount := count
10.        index := i
11. if maxCount > n/2
12.    print(a[index])
13. else
14.    print("No")
```

Analysis runtime complexity:

a. **Determine fundamental operation(s):**

maxCount = count;
if (maxCount > n/2)

b. **Determine worst case:**

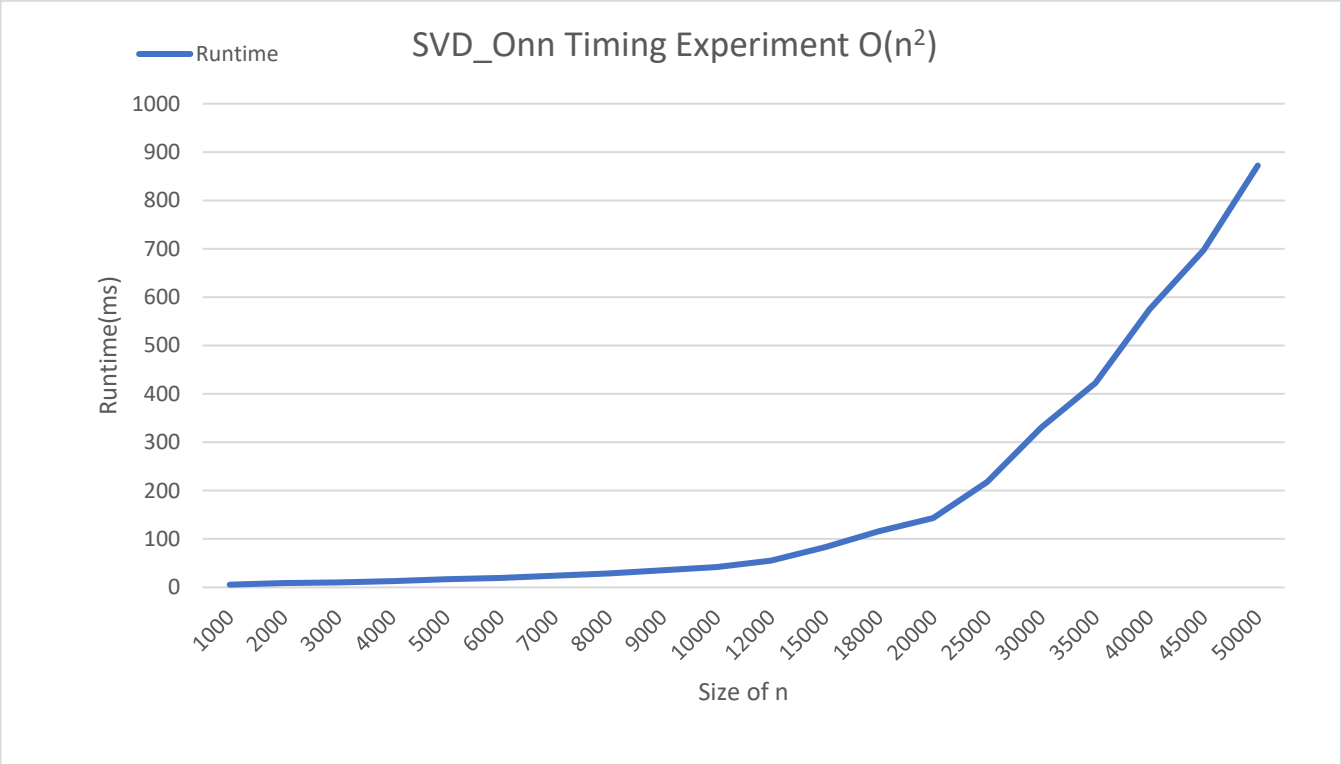
No majority element

c. **Form runtime complexity function:** Σ

d. **Characterise runtime complexity:**

$O(n^2)$

Graph of timing experiment



Results of Timing experiment:

Size of n	Runtime(ms)
1000	5.720052
2000	9.184367
3000	10.46479
4000	12.82098
5000	16.94992
6000	19.80746

7000	23.97185
8000	28.65998
9000	35.68875
10000	42.12879
12000	55.37087
15000	83.31864

18000	116.2774
20000	143.5882
25000	218.2397
30000	330.585
35000	422.897
40000	574.5478
45000	697.1299
50000	872.0925

2. Algorithm for O(nlog(n))

Informal Description:

Given an array of numbers ***a*** , scan through a to find majority element exists. If element in ***a*** exists more than n/2, return element, otherwise return 0

SVD_Onlogn(int a[]) Scan array a[] and **return** majority value if exists.

```
1.  Arrays.sort(a)
2.  count := 1
3.  x := a[0]
4.  for i := 1 i < arr.length i++
5.      if x == a[i]
6.          count++
7.          if count > a.length/2
8.              print(x)
9.      else
10.         x = a[i]
11.         count = 1
```

a. Determine fundamental operation(s)

Arrays.sort(arr)

If x == arr[i]

b. Determine worse case:

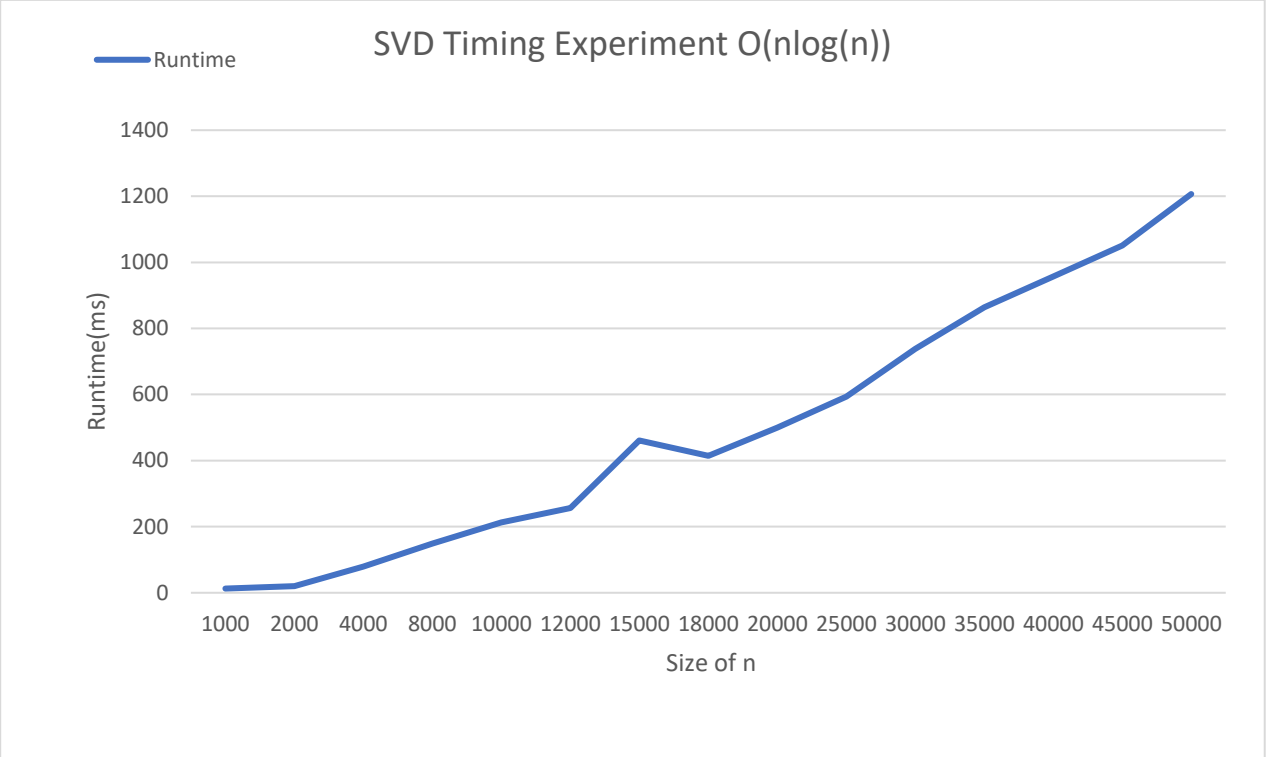
No majority value

c. **Form runtime complexity function:**

d. Characterise runtime complexity:

O(nlog(n))

Graph of timing experiment



Runtime(ms)	Size of n
1000	12.73825
2000	19.75427
4000	79.41238
8000	149.1288
10000	213.2128

12000	256.3459
15000	460.6095
18000	414.005
20000	499.9647
25000	593.4094

30000	738.4379
35000	864.4171
40000	956.9516
45000	1050.909
50000	1206.633

3. **Algorithm for O(n)**

Informal Description:

Given an array ***a*** of integers. Scan through array ***a*** in Hashmap to find majority element in ***a***. If element occurs more than n/2 than return found return element. If not return nothing.

Formal Algorithm:

SVD_On(int a[]) scan array a[] and return majority value if exists.

```
1.  Hashmap<Integer, Integer> map
2.  for i := 0 i < arr.length i++
3.      if map.containsKey(arr[i])
4.          count := map.get(arr[i]) + 1
5.          if count > arr.length / 2
6.            print(arr[i])
7.            return
8.          else
9.             map.put(arr[i], count)
10.     else
11.         map.put(arr[i],1)
```

a. **Determine fundamental operation(s)**

```
if map.containsKey(arr[i])
return
```

b. **Determine worst case:**

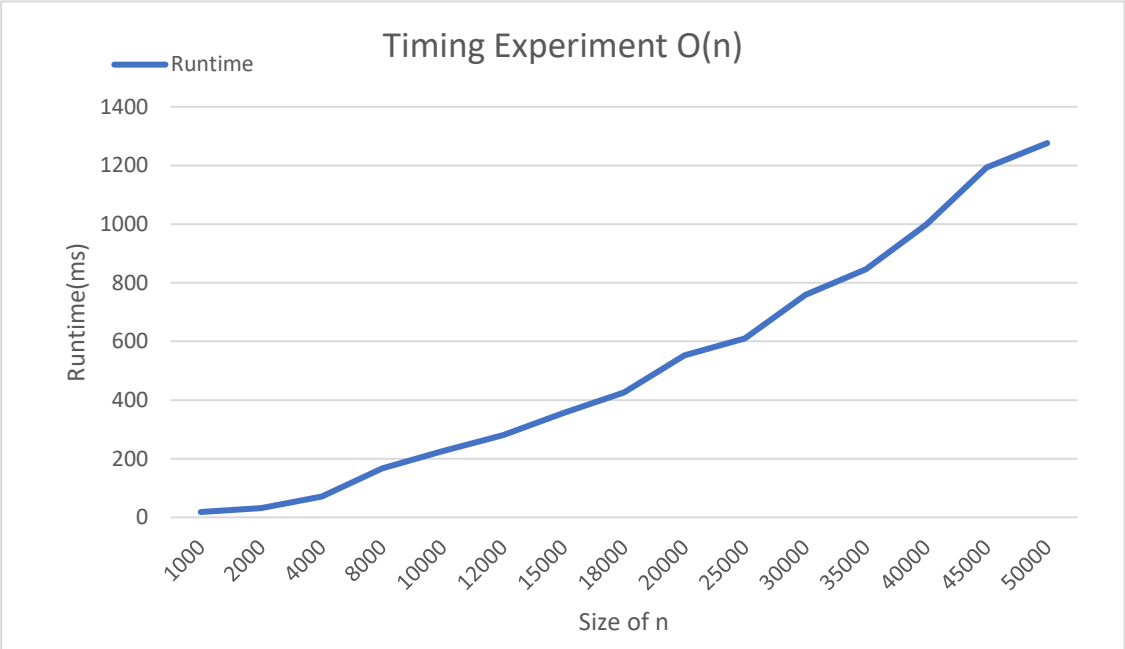
if no majority value exists

c. **Form runtime complexity function**

d. **Characterise runtime complexity:**

O(n)

Graph of timing experiments:



Graph shows increase in runtime as n increases. Presents a steady growth.

Size of n	Runtime(ms)
1000	18.2972
2000	31.22735
4000	70.56173
8000	166.9371

10000	226.1859
12000	280.4244
15000	356.1401
18000	425.7104
20000	552.4438

25000	610.365
30000	759.0297
35000	845.7561
40000	998.4944
45000	1192.766
50000	1276.237

Exercise 2: Data Structure Design and Implementation

1. **Implement a class ArrayHashTable and basic structure and constructor.**

ArrayHashTable.java

```
Class ArrayHashTable()

Object[][] table
Int chainSize := 5
Int[] counts

ArrayHashTable()
        table := new Object[capacity][]
        counts := new int[capacity]
        setTable()
        setCount()

Void settable()
        For i := 0 i < capacity i++
                Table[i] := null
Void setCount()
        For i:=0 I < capacity i++
                Counts[i] := 0
```

2. **Implement method add**

```
Boolean add(Object obj)
        Int hash := obj.hashcode() % this.capacity ←

        If table[hash] == null
                Object[] chain := new Object[chainSize]
                table[hash] := chain

        if !contains(obj)
                if table[hash][table[hash].length – 1] != null
                  Object[] tempArr := new Object[this.table[hash].length*2]
                  arraycopy(this.table[hash],0,tempArr,0,this.table[hash].length

                  this.table[hash] := tempArr

                boolean addObj := false
                int I := 0
                while !addObj
                        if table[hash][i] == null
                                table[hash][i] := obj
                                counts[hash]++
                                size++addObj := true
```

```
                    i++
                return true
            return false
```

3. **Implement method contains**

```
boolean contains(Object obj)
        int hash := obj.hashCode() % this.capacity

        for I := 0 i < counts[hash] i++
                if table[hash][i] == obj
                        return true
        return false
```

4. **Implement method remove**

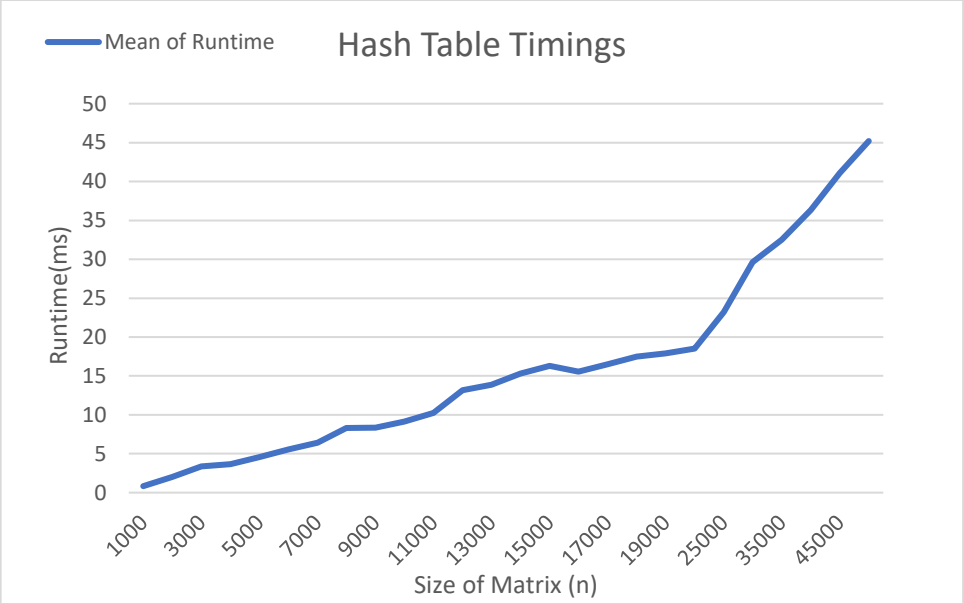
```
Boolean remove(Object obj)
        Int hash = obj.hashCode() % this.capacity

        For I := 0 I < counts[hash] i++
                If table[hash][i] == obj
                        Table[hash][i] == null

                        For j := I + 1 j < counts[hash] j++
                                Table[hash][j-1] := table[hash][j]
                        Counts[hash] - -
                        Size- -

                        Return true
        Return false
```

5. Timing experiment



Graph shows that as number of size increases the runtime of hashing varies –as round 15000 there is a noticeable decrease than as the size is increased to 25000 there is a large spike in time it takes to complete. As the random numbers are being added to the hash table there time it takes to complete can vary. Due to the hash table using 2d array for chaining, there had to be a way to identify numbers using if and Boolean statements instead of an arrayList which have functions to allow manipulation of adding and deleting. The add() and remove() methods have to scan through the Object array each time to identify the value. The remove() function has to move all items back once on is removed as arrays cannot be dynamically altered. Add() function has to create a bigger array if the max sized is reached and copy it to a temp array – this would not have to be done in a List as they can be alter much easier.

Results of timing experiments below:

Size of Matrix (n)	Mean	Standard Deviation
1000	0.83545984	8.31272
2000	2.026261	20.16105
3000	3.355281	33.38463
4000	3.666951	36.4857
5000	4.570236	45.47327
6000	5.57048	55.42557
7000	6.419198	63.87021
8000	8.316491	82.74804
9000	8.337468	82.95676
15000	16.28129	161.9968
20000	18.52929	184.3641
25000	23.22291	231.065
30000	29.63768	294.8912
35000	32.50584	323.429
40000	36.34326	361.6109
45000	41.07506	408.6917
50000	45.1952	449.6866

SVD.java

```
1  package question1;
2
3  import java.util.Arrays;
4  import java.util.HashMap;
5  import java.util.Random;
6
7  /**
8   *
9   * @author Nahim
10  */
11 public class SVD {
12
13     //Algorithm to find SVD whose run time complexity id O(n*n)
14     public static void SVD_Onn(int arr[], int n){
15         int maxCount = 0;
16         int index = -1;
17         long t1 = System.nanoTime();
18         for(int i = 0; i < n; i++){
19             int count = 0;
20             for(int j = 0; j < n; j++)
21             {
22                 if(arr[i] == arr[j])
23                     count++;
24             }
25
26
27             if(count > maxCount)
28             {
29                 maxCount = count;
30                 index = i;
31         }
32     }
33         if(maxCount > n/2){
34             System.out.println(arr[index]);
35
36         }
37         else
38             System.out.print("no value found");
39
40
41
42     }
43
44     // Algorithm to find SVD whose worst runtime complexity is O(nlog(n))
45     public static void SVD_Onlogn(int[] arr){
46
47     Arrays.sort(arr);
48     int count = 1;
49     int x = arr[0];
50
51
52     for (int i = 1; i <arr.length ; i++) {
53         if(x==arr[i]){
54             count++;
55             if(count>arr.length/2) {
56                 System.out.println(x);
57             }
58         }else{
59             x = arr[i];
60             count = 1;
61         }
```

```
63      }

65

        }
67

        // Algorithm for finding SVD whose space complexity and worst case runtime is
            O(n) using a hashmap
69      public static void SVD_On(int[] arr){

71           long t1 = System.nanoTime();
             long t2 = 0;
73

            HashMap<Integer,Integer> map = new HashMap<Integer, Integer>();
75

            for(int i = 0; i < arr.length; i++){
77              if(map.containsKey(arr[i])){
                    int count = map.get(arr[i]) + 1;
79                  if(count > arr.length / 2){
                        System.out.println(arr[i]);
81

83                  }else
                        map.put(arr[i], count);
85                  t2 = System.nanoTime();

87              }
                else
89                  map.put(arr[i], 1);
            }
91              long end = t2 - t1;
            System.out.println(end);
93

95      }

97      public static void main(String[] args){

99          Random rand = new Random();
            int randomNum = rand.nextInt();
101         int n = 50000;
            int[] a = new int [n];
103         int[] arr = {7,7,9,3,2,7,7};

105         for(int i = 0; i < a.length; i++){
                a[i] = randomNum;
107         }

109         //testing

111         //O(n*n)
            //SVD_Onn(a,n);
113

            //O(nlog(n))
115         //SVD_Onlogn(a);

117         //O(n)
            SVD_On(a);
119     }

121 }
```

ArrayHashTable.java

```
1
package question2;
3
import java.io.FileWriter;
5 import java.io.IOException;
import java.io.PrintWriter;
7 import java.util.Random;
import java.util.logging.Level;
9 import java.util.logging.Logger;

11 /**
 *
13 * @author Nahim
 */
15 public class ArrayHashTable extends HashTable {

17

    Object[][] table; //Create Object variable
19     int chainSize = 5; //Initial size of chain
    int[] counts; //Array to store counts
21


23

    public ArrayHashTable(){
25         table = new Object[capacity][]; //initialise 2D Object
        counts = new int[capacity];
27

29         setTable(); //set table to null;

31

        setCount(); //set count to 0
33

35     }

37     public void setTable(){
        //initialise all values in table to null
39         for(int i = 0; i<capacity; i++){
            table[i] = null;
41         }
    }
43

    public void setCount(){
45         //initialise all counts to 0
        for(int i = 0; i < capacity; i++){
47             counts[i] = 0;
        }
49     }


51

53     /**
     * Method to add Object to HashTable
55     * @param obj
     * @return
57     */
    @Override
59     boolean add(Object obj) {

61         // generate hash code
```

```
        int hash = obj.hashCode() % this.capacity;

        //check if chain arrays exists and make new one if it doesn't
        if(table[hash] == null){
            Object[] chain = new Object[chainSize];
            table[hash] = chain;
        }

        //check if object is already in hash table
        if(!contains(obj)){
            //Double chain capcity if max number
            if(table[hash][table[hash].length - 1] != null){

                // copy chain into temporary array
                Object[] tempArr = new Object[this.table[hash].length*2];
                System.arraycopy(this.table[hash], 0, tempArr, 0, this.table[hash
                    ].length);

                // Copy chains back doubled in size
                this.table[hash] = tempArr;
        }
            boolean addObj = false;
            int i = 0;
            while(!addObj){
                //checks if space is free.
                if(table[hash][i] == null){
                    table[hash][i] = obj;
                    counts[hash]++;
                    size++;
                    addObj = true;
                    System.out.println("Added " + table[hash][i] + " hash "
                    + hash + " to hash table. ");
                }
                i++;
            }

            return true;
    }
        return false;

    }

    /**
     * Method to check if a Object is in the Hashtable
     * @param obj
     * @return
     */
    @Override
    boolean contains(Object obj) {
        int hash = obj.hashCode() % this.capacity;

        // search through chain for Object
        for(int i = 0; i < counts[hash]; i++){
            if(table[hash][i] == obj){
                System.out.println(table[hash][i] + " found at position " +
                        hash + "," + i + ".");
                return true;
            }
        }

        return false;
    }

```

```
    /**
125  * Method to remove object from Hashtable
     * @param obj
127  * @return
     */
129 @Override
    boolean remove(Object obj) {
131     int hash = obj.hashCode() % this.capacity;

133     for(int i = 0; i < counts[hash]; i++){
            if(table[hash][i] == obj){
135             System.out.println("Removed " + table[hash][i] + " " + "from hash
                    table");
                table[hash][i] = null;
137
                //Move all items back one
139             for(int j = i + 1; j < counts[hash]; j++){
                    table[hash][j-1] = table[hash][j];
141             }

143             counts[hash]--;
                size--;
145
                return true;
147         }
        }
149     return false;
    }
151
    /**
153  * Method to test run time experiment
     */
155 public static void TimingExperiment(){


157
        Random r = new Random();
159     ArrayHashTable hTable = new ArrayHashTable();
        int a = 100;
161     int n = 1000; // matrix size

163     while(n <= 50000) {

165         int[] numbers = new int[n];

167         for(int j = 0; j < n; j++){
                numbers[j] = Math.abs(r.nextInt());
169         }

171         // mean and std deviation

173         double sum = 0;
            double sumSquared = 0;
175


177             long t1 = System.nanoTime();

179             for(int j = 0; j < n; j++){
                    hTable.add(numbers[j]);
181             }

183             for(int j = 0; j < n; j++){
                    hTable.remove(numbers[j]);
185             }
```

```
187                 long t2 = System.nanoTime() - t1;

189                 // Recording it in milli seconds to make it more interpretable
                    sum += (double)t2 / 1000000.0;
191                 sumSquared += (t2/1000000.0) * (t2/1000000.0);

193                 double mean = sum / a;
                    double variance = sumSquared / a - (mean * mean);
195                 double stdDev = Math.sqrt(variance);

197                 if(n < 20000){
                        n += 1000;
199                 }

201                 else if(n < 50001){
                        n += 5000;
203                 }
            }
205
        }
207
       /**
209     * Main method for testing
        * @param args
211     */
        public static void main(String[] args) {
213
            //testing
215         int t = 6;
            ArrayHashTable table = new ArrayHashTable();
217
            System.out.println("Testing");
219         //add values to hash table
            for(int i = 0; i < t; i++ ){
221             table.add(i);
            }
223
            //check values were added correctly
225         for(int i = 0; i < t; i++){
                table.contains(i);
227         }

229         //remove all values
            for(int i = 0; i < t; i++){
231             table.remove(i);
            }
233


235        TimingExperiment();

237     }


239
}
```

HashTable.java

```
/*
Hash Table interface for the DS&A labs, week 1, semester 2. Note the emphasis here
is to get you to implement the algorithms, not write fancy code (although feel free to
do so! There are exercises for Programming 2 to engineer it more.
 */
package question2;

/**
 *
 * @author ajb
 */
public abstract class HashTable {
    int capacity=100;
    int size=0;

    public int size(){ return size;}

/**
 * Adds the specified element to this hash table if it is not already present
 *
 * @param obj
 * @return true if the element is successfully added
 */
   abstract boolean add(Object obj);

/**
 *
 * @param obj
 * @return  true if this hash table contains the specified element
 */
   abstract boolean contains(Object obj);
/**
 *
 * @param obj
 * @return Removes the specified element from this set if it is present
 */
  abstract  boolean remove(Object obj);

}
```