Jackson State University
Department of Computer Science
CSC 323 Algorithm Design and Analysis
Fall 2017
Instructor: Dr. Natarajan Meghanathan
Programming Project 2

Determining the Average Number of Comparisons in the Element Uniqueness Problem

Due: October 5, 2017, 11.30 AM **Maximum Points: 100**

Maximum Possible value (*m*) of the elements in your arrays:

Student Name	*m* values		Student Name	*m* values
Armon Clark	100, 1000		Taj Nelson	1000, 10000
Daniel Epps	200, 2000		Patricia Perry	1100, 11000
Allee Gammons	300, 3000		Daniel Powell	1200, 12000
Menlik Getachew	400, 4000		Aiyanna Price	1300, 13000
Taylor Hasty	500, 5000		Allaysia Roberts	1400, 14000
Derric Jackson	600, 6000		Dreshon Sanders	1500, 15000
Devario Lewis	700, 7000		Mircale Williams	1600, 16000
Jai-Michael McMillian	800, 8000		Michael Wilson	1700, 17000
Nahu Merawi	900, 9000			

Project Description

Consider the "Element Uniqueness" problem discussed in class. The problem is: Given an array, determine whether the elements in the array are distinctly unique or not. For example, the array [3, 5, 9, 2, 1] has unique elements. The array [4, 9, 4, 2, 3] does not have unique elements as element 4 is present more than once. Following is the algorithm to determine whether a given array A[0…*n*-1] of size *n* has unique elements or not.

```
ALGORITHM UniqueElements(A[0..n − 1])
    //Determines whether all the elements in a given array are distinct
    //Input: An array A[0..n − 1]
    //Output: Returns "true" if all the elements in A are distinct
    //          and "false" otherwise
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

The run-time of the algorithm is dependent on the number on the number of comparisons, which is the basic operation.

For a given array size *n*, the number of comparisons depends on the contents of the array, i.e., the values of the elements of the array.

- For arrays of any size > 1, the best-case number of comparisons is 1. This will happen when the first and second elements of the array are the same.

- For array of size n, the worst case number of comparisons is $\frac{n(n-1)}{2}$

The objective of this project is to determine the average-case number of comparisons required for array of size n.

It has been observed that given an array size n, the contents of which are randomly generated, the average number of comparisons depends on the maximum possible value of the elements in the array.

Let m be the maximum value of the elements in an array. Each student would do the project with two given values of m. The two values are written by the instructor in the first-page of the project description.

Implementation
Generate arrays of size n = 0.1m, 0.2m , 0.3m, 0.4m, 0.5m, 0.6m, 0.7m, 0.8m, 0.9m, m, 2m, 3m.
You need to use a random number generator to generate the elements of the array.

For each value of n and m, you would generate 100 sets of arrays. Make sure, you use the seed of the random number generator to be "System time", which is continuously changing.

Implement the "Element Uniqueness" algorithm, say with filename uniqueElements.java. For a given value of n and m, you need to run the code 100 times. For a given value of m, the array of size n (from the above list of different values) should be generated randomly in your uniqueElements.java code. You will determine the number of comparisons it takes for each run of the code and then average it over the 100 runs. The average represents the average number of comparisons for an n-element array, whose contents are generated randomly, and the maximum possible value for any element in the array is m.

To run the code 100 times for a given value of n and m, it is better to automate the code such that it runs a loop 100 times, during each run, it randomly generates an array of size n such that the maximum possible value for any element in the array is m.

Make sure to repeat your experiments with the values of n = 0.1m, 0.2m , 0.3m, 0.4m, 0.5m, 0.6m, 0.7m, 0.8m, 0.9m, m, 2m, 3m.

Results
Plot the following graph for each value of m:
Use the different values of n as the X-axis. In the Y-axis, you will plot the worst-case number of comparisons $\frac{n(n-1)}{2}$ and the average-case number of comparisons determined from your experiments.

So, you will have two curves in your graph. Are the two curves close enough to each other or far away from each other? What is the trend as n increases?

What to submit:

A project report that has the following:

- **Your Java code**
- **The result graphs** for the two values of m given to you

Your interpretation of the results. Relate the average number of comparisons observed for each of the values for m and the corresponding n values used. ***What can you conclude about the average case number of comparisons for a given n and m***?

The following code segment generates 100 different sets of arrays of size 10, such that the maximum possible value for any element in the array is 100. You could use this code segment and expand it by implementing the "Element Uniqueness" algorithm, determine the number of comparisons in each run and the average number of runs.

```
import java.util.*;

class uniqueElements{

  public static void main(String[] args){

    Scanner sc = new Scanner(System.in);

    // Get the input for the maximum value of an element and store it in variable name maxValue
    System.out.print("Enter the maximum value for an element: ");

    // Get the input for the number of elements in the array and store it in variable name numElements
    System.out.print("Enter the number of elements in the array: ");

    /* Test the two inputs maxValue and numElements for software safety. Display appropriate error messages and
       then terminate the program if one or both of the inputs are not appropriate as specified. */

    int totalRuns = 100;

    int A[] = new int[numElements];

  // initialize a variable, totalComp, to 0 and it keeps track of the total number of comparisons

  for (int runs = 1; runs <= totalRuns; runs++){ // begin each run

    Random rand = new Random(System.nanoTime( ));

    // initialize a variable, numComp, to 0
   // and it keeps track of the number of comparisons in a particular run

    for (int index = 0; index < numElements; index++){ // generate array for a particular run

      A[index] = rand.nextInt(maxValue);

    }

     /*
        Implement your "Element Uniqueness" algorithm here. Increment numComp appropriately.
        Note: Instead of using return statements as given in the pseudo code description,
              You should use break statement to come out of your loops, when the array is
              found to have two identical elements.
     */

     // add numComp to totalComp

  } // end each run

     // print the average number of comparisons (divide totalComp by 100)

  }
}
```