CSC 323 Algorithm Design and Analysis, Fall 2018
Instructor: Dr. Natarajan Meghanathan
Project 1: Brute Force Algorithm for the Element Uniqueness Problem

Due by: Sept. 20th, 11.30 AM

In this project, you will implement the brute force algorithm discussed in Module 1 for the "Element Uniqueness Problem." Each of you have been assigned two 'm' values that correspond to the maximum value for an element in the array. The two 'm' values are independent of each other and should be considered separately.

For a particular 'm' value, the values for the array size 'n' are: 0.1m, 0.2m, 0.3m, 0.4m, 0.5m, 0.6m, 0.7m, 0.8m, 0.9m, m. For example, if m = 100, the values of the array size 'n' are: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

As part of your code, you should generate an array of size 'n' whose values are generated randomly in the range [1...m]. Your algorithm should keep track of the number of comparisons needed to determine whether the array of random elements (generated as above) is unique or not.

You should run your algorithm/code several times (say, 10000 times using an automated loop) for each (n, m) pair and determine the average number of comparisons.

For each of the two 'm' values (with 'n' varying from 0.1m to m as described above), plot the values for 'n' vs. the average number of comparisons for the n value.

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

Student Name	*m* values		Student Name	*m* values
Clark, Lavaskie	100, 1000		Manuel, Jackie	1000, 10000
Epps, Justin	200, 2000		McIntosh, Blair	1100, 11000
Harris, James	300, 3000		Sheffey, Varlin	1200, 12000
Hester, Larriel	400, 4000		Simmons, Jetnya	1300, 13000
Hopson, Shanice	500, 5000		Thomas, Eriana	1400, 14000
Jackson, Martice	600, 6000		Walker, Brandon	1500, 15000
Jones, Demarius	700, 7000		Wynn, Marcus	1600, 16000
Kang, Ning	800, 8000		Zimmerman, Taba	1700, 17000
Kirk, Damon	900, 9000			

Note (For C++): For each iteration, if you create the array using dynamic memory allocation (for example, shown below), then delete the allocated memory at the end of the iteration. This will prevent you from running out of memory while running the 10000 iterations, especially with larger array size.

```
int *array = new int[numElements];  // at the beginning of an iteration
.............
...........
delete[ ] array; // at the end of an iteration
```

Submission (through Canvas): Upload the following together as ONE PDF file.
Your code (Java/C++/Python)
Excel plots (as required above for each 'm' value) and your explanation interpreting the results.

Your explanation should address the following:
(1) For a given maximum value 'm' and with increase in the number of elements 'n', does the average number of comparisons increase, decrease or remain about the same? Give an explanation of why you see the pattern you observe.
(2) For a given value of 'n', does the average number of comparison depend on the value of 'm' or not? Infer from the results/figures and explain.