Jasper Belenzo
301329283

Assignment 2
COMP 254 Data Structures & Algorithms

1. Exercise 1

A.) Give a big-Oh characterization, in terms of **n**, of the running time of the **example1** method from **Exercises.java** class in Lesson 4 examples.

```java
/**
 * Code for end-of-chapter exercises on asymptotics.
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwasser
 */
class Exercises {

  /** Returns the sum of the integers in given array. */
  public static int example1(int[] arr) {
    int n = arr.length, total = 0;
    for (int j=0; j < n; j++)       // loop from 0 to n-1
      total += arr[j];
    return total;
  }
```

Answer: Disregard constants. This would be **O(n)**.

B.) Give a big-Oh characterization, in terms of **n**, of the running time of the **example2** method from **Exercises.java** class in Lesson 4 examples.

```java

  /** Returns the sum of the integers with even index in given array. */
  public static int example2(int[] arr) {
    int n = arr.length, total = 0;
    for (int j=0; j < n; j += 2)    // note the increment of 2
      total += arr[j];
    return total;
  }
```

Answer: Disregard constants. This could be O(n/2), but it should be **O(n)**.
As O(n/2) would just have the same growth rate as **O(n)**, especially graphed. An n/2 would just be another n.

C.) Give a big-Oh characterization, in terms of **n**, of the running time of the **example3** method from **Exercises.java** class in Lesson 4 examples.

```java
/** Returns the sum of the prefix sums of given array. */
public static int example3(int[] arr) {
  int n = arr.length, total = 0;
  for (int j=0; j < n; j++)        // loop from 0 to n-1
    for (int k=0; k <= j; k++)     // loop from 0 to j
      total += arr[j];
  return total;
}
```

Answer: Disregard constants. This would be **O(n^2)**.
Inner loop **j** could run at worst at n times, multiplied by Outer loop **n**, so therefore "**n** multiplied **j** which is at worst is **n**" which makes it **O(n^2)**.

D.) Give a big-Oh characterization, in terms of **n**, of the running time of the **example4** method from **Exercises.java** class in Lesson 4 examples.

```java
/** Returns the sum of the prefix sums of given array. */
public static int example4(int[] arr) {
  int n = arr.length, prefix = 0, total = 0;
  for (int j=0; j < n; j++) {      // loop from 0 to n-1
    prefix += arr[j];
    total += prefix;
  }
  return total;
}
```

Answer: Disregard constants. This would be **O(n)**.

E.) Give a big-Oh characterization, in terms of **n**, of the running time of the **example5** method from **Exercises.java** class in Lesson 4 examples

```java
/** Returns the number of times second array stores sum of prefix sums from first. */
public static int example5(int[] first, int[] second) { // assume equal-length arrays
  int n = first.length, count = 0;
  for (int i=0; i < n; i++) {      // loop from 0 to n-1
    int total = 0;
    for (int j=0; j < n; j++)      // loop from 0 to n-1
      for (int k=0; k <= j; k++)   // loop from 0 to j
        total += first[k];
    if (second[i] == total) count++;
  }
  return count;
}
```

Answer: Disregard constants. This would be **O(n³)**.
Due to the nested inner inner loop **j** (which at worst can be **n**) and inner loop **n** and outer loop **n**, which makes it run "**j** multiplied by **n** multiplied **n**" in effect its **O(n³)**.

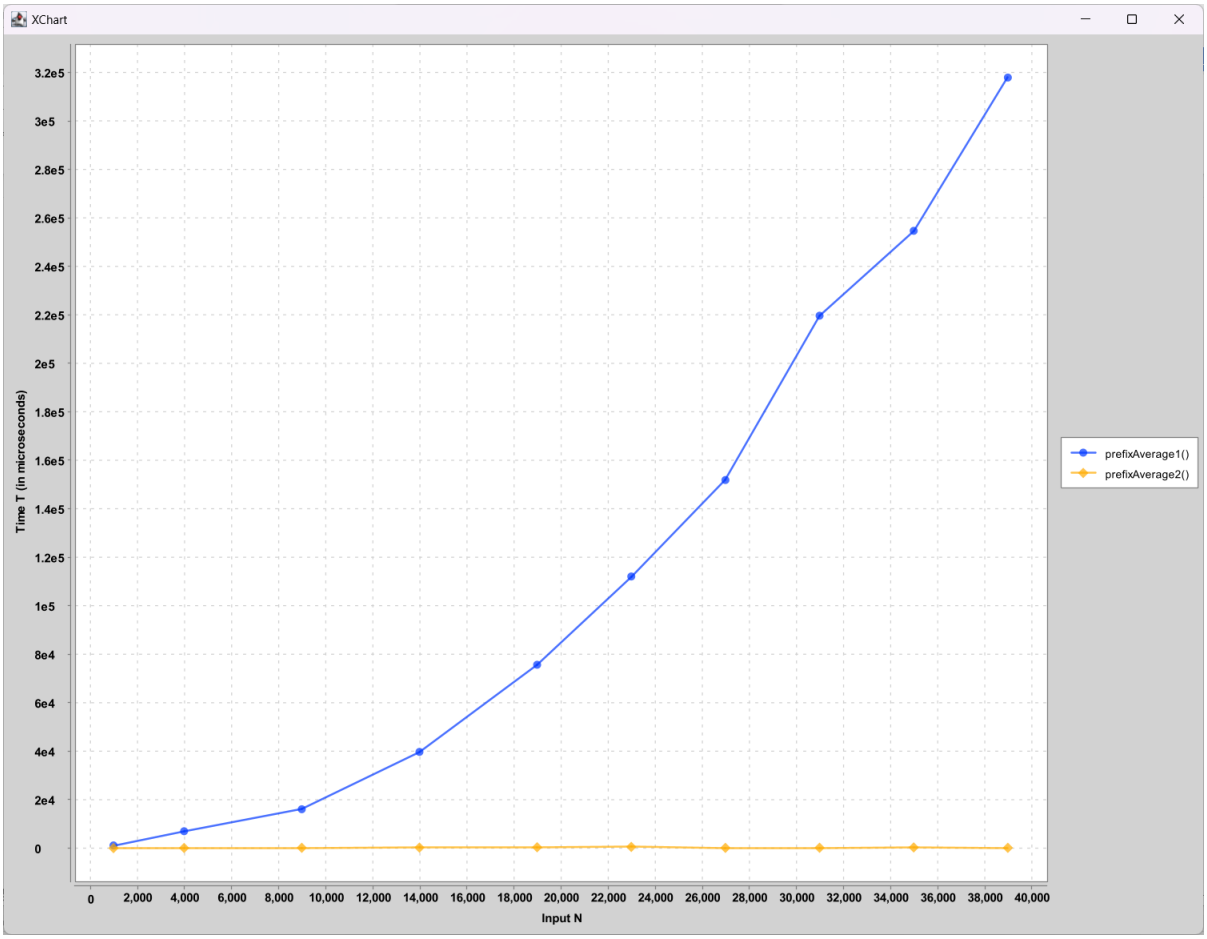
2. Exercise 2

Perform an **experimental analysis** of the two algorithms *prefixAverage1* and *prefixAverage2*, from lesson examples. Optionally, visualize their running times as a function of the input size with a **log-log chart**. Use either Java or Python graphical capabilities for visualization. **Hint**: Choose representative values of the input size **n**, similar to *StringExperiment.java* from class examples.

Answer: prefixAverage2() runs faster at **O(n)** while prefixAverage1() runs slower at **O(n²)**.

```
Problems  Javadoc  Declaration  Search  Console ×  Error Log  Checkstyle violations  Call Hierarchy
ExerciseTwoPrefixAverages [Java Application] C:\Users\Belenzo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.
Experimental Analysis on prefixAverage1()
PrefixAverage1() - Trial: 0, Input N Size: 1000.0, Elapsed Time: 1133.0
PrefixAverage1() - Trial: 1, Input N Size: 4000.0, Elapsed Time: 6989.0
PrefixAverage1() - Trial: 2, Input N Size: 9000.0, Elapsed Time: 16059.0
PrefixAverage1() - Trial: 3, Input N Size: 14000.0, Elapsed Time: 39714.0
PrefixAverage1() - Trial: 4, Input N Size: 19000.0, Elapsed Time: 75659.0
PrefixAverage1() - Trial: 5, Input N Size: 23000.0, Elapsed Time: 112076.0
PrefixAverage1() - Trial: 6, Input N Size: 27000.0, Elapsed Time: 151864.0
PrefixAverage1() - Trial: 7, Input N Size: 31000.0, Elapsed Time: 219657.0
PrefixAverage1() - Trial: 8, Input N Size: 35000.0, Elapsed Time: 254666.0
PrefixAverage1() - Trial: 9, Input N Size: 39000.0, Elapsed Time: 317899.0


Experimental Analysis on prefixAverage2()
PrefixAverage2() - Trial: 0, Input N Size: 1000.0, Elapsed Time: 22.0
PrefixAverage2() - Trial: 1, Input N Size: 4000.0, Elapsed Time: 77.0
PrefixAverage2() - Trial: 2, Input N Size: 9000.0, Elapsed Time: 152.0
PrefixAverage2() - Trial: 3, Input N Size: 14000.0, Elapsed Time: 226.0
PrefixAverage2() - Trial: 4, Input N Size: 19000.0, Elapsed Time: 373.0
PrefixAverage2() - Trial: 5, Input N Size: 23000.0, Elapsed Time: 515.0
PrefixAverage2() - Trial: 6, Input N Size: 27000.0, Elapsed Time: 122.0
PrefixAverage2() - Trial: 7, Input N Size: 31000.0, Elapsed Time: 170.0
PrefixAverage2() - Trial: 8, Input N Size: 35000.0, Elapsed Time: 294.0
PrefixAverage2() - Trial: 9, Input N Size: 39000.0, Elapsed Time: 139.0
```




```java
package assignment.lab.two;
import org.knowm.xchart.*;
/*
 * Jasper Belenzo
 * 301329283
 */
public class ExerciseTwoPrefixAverages {
        /** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. */
        public static double[] prefixAverage1(double[] x) {
                int n = x.length;
            double[] a = new double[n];    // filled with zeros by default
            for (int j=0; j < n; j++) {
                double total = 0;            // begin computing x[0] + ... + x[j]
                for (int i=0; i <= j; i++) {
                        total += x[i];
                }
                a[j] = total / (j+1);        // record the average
            }
            return a;
          } // prefixAverage1

        /** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. */
        public static double[] prefixAverage2(double[] x) {
                int n = x.length;
                double[] a = new double[n];    // filled with zeros by default
                double total = 0;              // compute prefix sum as x[0] + x[1] + ...
                for (int j=0; j < n; j++) {
                        total += x[j];               // update prefix sum to include x[j]
                        a[j] = total / (j+1);        // compute average based on current sum
                }
```

```
                return a;
        } // prefixAverage2

        public static void main(String[] args) {
                double[] array1000 = generateDoubleArray(1000); // 1
                double[] array4000 = generateDoubleArray(4000); // 2
                double[] array9000 = generateDoubleArray(9000); // 3
                double[] array14000 = generateDoubleArray(14000); // 4
                double[] array19000 = generateDoubleArray(19000); // 5
                double[] array23000 = generateDoubleArray(23000); // 6
                double[] array27000 = generateDoubleArray(27000); // 7
                double[] array31000 = generateDoubleArray(31000); // 8
                double[] array35000 = generateDoubleArray(35000); // 9
                double[] array39000 = generateDoubleArray(39000); // 10
                double[][] doubleArrays = { array1000, array4000, array9000, array14000, array19000,
array23000, array27000, array31000, array35000, array39000};
                int trials = 10; // Ten Trials for both prefixAverage1 PA1 and prefixAverage2 PA2
            System.out.println("Experimental Analysis on prefixAverage1()");
            double[] inputNpa1 = new double[10];
            double[] timeTpa1 = new double[10];
            for (int t1 = 0; t1 < trials; t1++) {
                long startTime = ((System.nanoTime()) / 1000);
                double[] resultValueDoubleArray = prefixAverage1(doubleArrays[t1]);
                long endTime = ((System.nanoTime()) / 1000);
                long elapsed = endTime - startTime;
                inputNpa1[t1] = doubleArrays[t1].length;
                timeTpa1[t1] = elapsed;
                System.out.println("PrefixAverage1() - Trial: " + t1 + ", Input N Size: " + inputNpa1[t1]
+ ", Elapsed Time: " + timeTpa1[t1]);
            }
            System.out.println("\n\nExperimental Analysis on prefixAverage2()");
            double[] inputNpa2 = new double[10];
            double[] timeTpa2 = new double[10];
            for (int t2 = 0; t2 < trials; t2++) {
                long startTime = ((System.nanoTime()) / 1000);
                double[] resultValueDoubleArray = prefixAverage2(doubleArrays[t2]);
                long endTime = ((System.nanoTime()) / 1000);
                long elapsed = endTime - startTime;
                inputNpa2[t2] = doubleArrays[t2].length;
                timeTpa2[t2] = elapsed;
                System.out.println("PrefixAverage2() - Trial: " + t2 + ", Input N Size: " + inputNpa2[t2]
+ ", Elapsed Time: " + timeTpa2[t2]);
            }
            XYChart chart = new XYChartBuilder().xAxisTitle("Input N").yAxisTitle("Time T (in
microseconds)").width(1200).height(900).build();
            XYSeries seriesPa1 = chart.addSeries("prefixAverage1()", inputNpa1, timeTpa1);
            XYSeries seriesPa2 = chart.addSeries("prefixAverage2()", inputNpa2, timeTpa2);
            new SwingWrapper<XYChart>(chart).displayChart();
        }

        public static double[] generateDoubleArray(int arraySize) {
                double[] doubleArray = new double[arraySize];
                for (int i = 0; i < doubleArray.length; i++) {
                        doubleArray[i] = arraySize - i;
                }
                return doubleArray;
        }
}
```

3. Exercise 3

For each of the algorithms *unique1* and *unique2* (**Uniqueness.java** class in Lesson 4 examples) which solve the element uniqueness problem, **perform an experimental analysis** to determine the largest value of **n** such that the given algorithm runs in one minute

or less. **Hint**: Do a type of “binary search” to determine the maximum effective value of **n** for each algorithm.

Answer: The unique1() method is slow at $O(n^2)$, it will need an input size **n** of ‘**940,000 to 980,000**’ in order for it to run around a minute (58,500 to 60,500 milliseconds). Screenshot below shows input size **n** of ‘**940,000**’ took **59,616** milliseconds (almost a minute)

While unique2() is fast at O(n) there is **no** INT input n that could make it run at around a minute, even the max value for INT (in Java) which is 2,147,483,647 (32-Bit INTeger) will **just run under a second**. Both unique1() and unique2() require array inputs of the INT type (32-Bit Integer).

```
Problems  Javadoc  Declaration  Search  Console ×  Error Log  Checkstyle violations  Call Hierarchy
<terminated> ExerciseThreeUniqueness [Java Application] C:\Users\Belenzo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.8.v20230831-1047\jre\bin\javaw.exe (
Experimental Analysis on unique1()
One (1) Minute is 60,000 Milliseconds
unique1() - Trial: 0, Input N Size: 250000, Elapsed Time: 3926, Is Unique: true
unique1() - Trial: 1, Input N Size: 500000, Elapsed Time: 16692, Is Unique: true
unique1() - Trial: 2, Input N Size: 750000, Elapsed Time: 37206, Is Unique: true
unique1() - Trial: 3, Input N Size: 850000, Elapsed Time: 48963, Is Unique: true
unique1() - Trial: 4, Input N Size: 940000, Elapsed Time: 59616, Is Unique: true
unique1() - Trial: 5, Input N Size: 950000, Elapsed Time: 62066, Is Unique: true
unique1() - Trial: 6, Input N Size: 960000, Elapsed Time: 62373, Is Unique: true
unique1() - Trial: 7, Input N Size: 970000, Elapsed Time: 63996, Is Unique: true
unique1() - Trial: 8, Input N Size: 980000, Elapsed Time: 63798, Is Unique: true
unique1() - Trial: 9, Input N Size: 990000, Elapsed Time: 66597, Is Unique: true
Using BINARY SEARCH: Ideal Input Size N for unique1() to run in One Minute 'more or else' is --> 940000, timed at 59616


Experimental Analysis on unique2()
One (1) Minute is 60,000 Milliseconds
unique2() - Trial: 0, Input N Size: 250000, Elapsed Time: 6, Is Unique: true
unique2() - Trial: 1, Input N Size: 500000, Elapsed Time: 14, Is Unique: true
unique2() - Trial: 2, Input N Size: 750000, Elapsed Time: 10, Is Unique: true
unique2() - Trial: 3, Input N Size: 850000, Elapsed Time: 13, Is Unique: true
unique2() - Trial: 4, Input N Size: 940000, Elapsed Time: 23, Is Unique: true
unique2() - Trial: 5, Input N Size: 950000, Elapsed Time: 33, Is Unique: true
unique2() - Trial: 6, Input N Size: 960000, Elapsed Time: 44, Is Unique: true
unique2() - Trial: 7, Input N Size: 970000, Elapsed Time: 107, Is Unique: true
unique2() - Trial: 8, Input N Size: 980000, Elapsed Time: 219, Is Unique: true
unique2() - Trial: 9, Input N Size: 990000, Elapsed Time: 441, Is Unique: true
Using BINARY SEARCH: Ideal Input Size N for unique2() to run in One Minute 'more or else' is --> 0, timed at 0
unique2() is too fast and INT data type which is just 32-bit (2,147,438,647) is NOT enough to generate input
that will make it run close to One (1) Minute
```

```java
package assignment.lab.two;

import java.util.Arrays;
import org.knowm.xchart.*;
/*
 * Jasper Belenzo
 * 301329283
 */

public class ExerciseThreeUniqueness {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int[] array1 = generateUniqueIntArray(250000); // 1
		int[] array2 = generateUniqueIntArray(500000); // 2
		int[] array3 = generateUniqueIntArray(750000); // 3
		int[] array4 = generateUniqueIntArray(850000); // 4
		int[] array5 = generateUniqueIntArray(940000); // 5
		int[] array6 = generateUniqueIntArray(950000); // 6
		int[] array7 = generateUniqueIntArray(960000); // 7
```

```
                int[] array8 = generateUniqueIntArray(970000); // 8
                int[] array9 = generateUniqueIntArray(980000); // 9
                int[] array10 = generateUniqueIntArray(990000); // 10
                int[][] intArrays1 = { array1, array2, array3, array4,
                                array5, array6, array7, array8, array9, array10 };

                InputNandTimeT[] inatt1 = new InputNandTimeT[10];
                // int trials = 10; // Ten Trials for both unique1 U1 and unique2 U2
                System.out.println("Experimental Analysis on unique1()");
                System.out.println("One (1) Minute is 60,000 Milliseconds");
            int[] inputNu1 = new int[10];
            long[] timeTu1 = new long[10];
            for (int t1 = 0; t1 < inatt1.length; t1++) {
                long startTime = System.currentTimeMillis();
                boolean isUnique = unique1(intArrays1[t1]);
                long endTime = System.currentTimeMillis();
                long elapsed = endTime - startTime;
                inputNu1[t1] = intArrays1[t1].length;
                timeTu1[t1] = elapsed;
                inatt1[t1] = new InputNandTimeT(inputNu1[t1], timeTu1[t1], isUnique);
                System.out.println("unique1() - Trial: " + t1 + ", Input N Size: " +
inatt1[t1].getInputN() + ", Elapsed Time: " + inatt1[t1].getTimeT() + ", Is Unique: " +
inatt1[t1].getResultValue());
            }
                System.out.println("Using BINARY SEARCH: Ideal Input Size N for unique1() to run in One
Minute 'more or else' is --> " + binarySearchIterative(inatt1).getInputN() + ", timed at " +
binarySearchIterative(inatt1).getTimeT());

                array1 = generateUniqueIntArray(410156); // 1
                array2 = generateUniqueIntArray(820312); // 2
                array3 = generateUniqueIntArray(1640625); // 3
                array4 = generateUniqueIntArray(3281250); // 4
                array5 = generateUniqueIntArray(6562500); // 5
                array6 = generateUniqueIntArray(13125000); // 6
                array7 = generateUniqueIntArray(26250000); // 7
                array8 = generateUniqueIntArray(52500000); // 8
                array9 = generateUniqueIntArray(105000000); // 9
                array10 = generateUniqueIntArray(210000000); // 10
                int[][] intArrays2 = { array1, array2, array3, array4,
                                array5, array6, array7, array8, array9, array10 };

                InputNandTimeT[] inatt2 = new InputNandTimeT[10];
                // int trials = 10; // Ten Trials for both unique1 U1 and unique2 U2
                System.out.println("\n\nExperimental Analysis on unique2()");
                System.out.println("One (1) Minute is 60,000 Milliseconds");
            int[] inputNu2 = new int[10];
            long[] timeTu2 = new long[10];
            for (int t2 = 0; t2 < inatt2.length; t2++) {
                long startTime = System.currentTimeMillis();
                boolean isUnique = unique2(intArrays2[t2]);
                long endTime = System.currentTimeMillis();
                long elapsed = endTime - startTime;
                inputNu2[t2] = intArrays1[t2].length;
                timeTu2[t2] = elapsed;
                inatt2[t2] = new InputNandTimeT(inputNu2[t2], timeTu2[t2], isUnique);
                System.out.println("unique2() - Trial: " + t2 + ", Input N Size: " +
inatt2[t2].getInputN() + ", Elapsed Time: " + inatt2[t2].getTimeT() + ", Is Unique: " +
inatt2[t2].getResultValue());
            }
            System.out.println("Using BINARY SEARCH: Ideal Input Size N for unique2() to run in One Minute
'more or else' is --> " + binarySearchIterative(inatt2).getInputN() + ", timed at " +
binarySearchIterative(inatt2).getTimeT());
            System.out.println("unique2() is too fast and INT data type which is just 32-bit
(2,147,438,647) is NOT enough to generate input \nthat will make it run close to One (1) Minute");
        }

        /** Returns true if the target value is found in the data array. */
        public static InputNandTimeT binarySearchIterative(InputNandTimeT[] data) {
                int low = 0;
```

```java
        int high = data.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if ((data[mid].getTimeT() > 58500) && (data[mid].getTimeT() < 60500))     // found a match
                    return data[mid];
            else if ( 60500 < data[mid].getTimeT()) {
                    high = mid - 1;          // only consider values left of mid
                    // System.out.println("mid: " + mid + ", high: " + high);
          } else {
                    low = mid + 1;           // only consider values right of mid
                    // System.out.println("mid: " + mid + ", low: " + low);
          }
        }
        return new InputNandTimeT(0, 0, null);               // loop ended without success
      }

    /** Returns true if there are no duplicate elements in the array. */
    public static boolean unique1(int[] data) {
            int n = data.length;
        for (int j=0; j < n-1; j++)
            for (int k=j+1; k < n; k++)
                    if (data[j] == data[k]) {
                            return false;    // found duplicate pair
                    }
        return true;// if we reach this, elements are unique
      }

    /** Returns true if there are no duplicate elements in the array. */
    public static boolean unique2(int[] data) {
            int n = data.length;
        int[] temp = Arrays.copyOf(data, n);   // make copy of data
        Arrays.sort(temp);                     // and sort the copy
        for (int j=0; j < n-1; j++) {
            if (temp[j] == temp[j+1]) {              // check neighboring entries
                    return false;    // found duplicate pair
            }
            }
        return true;         // if we reach this, elements are unique
    }

    public static int[] generateUniqueIntArray(int arraySize) {
            int[] intArray = new int[arraySize];
            for (int i = 0; i < intArray.length; i++) {
                    intArray[i] = arraySize - i;
            }
            return intArray;
    }

}
```