**Lab Assignment #2 – Algorithm Analysis**

Due Date: Friday, Week 5

Purpose: The purpose of this Lab assignment is to:

1. Perform experimental analysis of algorithms
2. Explain and proof the running time of algorithms in terms of big-Oh notation
3. Design various algorithm that achieve a specified running time

References: Read the course’s text chapter 4 and the lecture slides. This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:

* This assignment must be completed individually by all the students.
* You will have to provide a **demonstration video for your solution** and upload the video together with the solution on **eCentennial** through the assignment link. See the **video recording instructions** at the end of this document.x

**Exercise 1:**

**If your first name starts with a letter from A-J inclusively:**

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.

For each of the above questions, use comments in the code to provide the results and a brief explanation.

**If your first name starts with a letter from K-Z inclusively:**

1. . Give a big-Oh characterization, in terms of **n**, of the running time of the **example1** method from **Exercises2.java** class in Lesson 4 examples.
2. Give a big-Oh characterization, in terms of **n**, of the running time of the **example2** method from **Exercises2.java** class in Lesson 4 examples.
3. Give a big-Oh characterization, in terms of **n**, of the running time of the **example3** method from **Exercises.2java** class in Lesson 4 examples.
4. Give a big-Oh characterization, in terms of **n**, of the running time of the **example4** method from **Exercises2.java** class in Lesson 4 examples.
5. Give a big-Oh characterization, in terms of **n**, of the running time of the **example5** method from **Exercises2.java** class in Lesson 4 examples.

(3 marks)

**Exercise 2:**

**If your first name starts with a letter from A-J inclusively:**

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.

**If your first name starts with a letter from K-Z inclusively:**

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.

(4 marks)

**Exercise 3:**

**If your first name starts with a letter from A-J inclusively:**

An array A contains n−1 unique integers in the range [0, n−1], that is, there is one number from this range that is not in A. Design an O(n)-time algorithm for finding that number. You are only allowed to use O(1) additional space besides the array A itself. Write the java method that implements this algorithm and a main method to test it.

**Hint:** Numbers in [0, n-1] form an arithmetic progression whose sum is known.

**If your first name starts with a letter from K-Z inclusively:**

Given an array A of n arbitrary integers, design an O(n)-time algorithm for finding an integer that cannot be formed as the sum of two integers in A. Write the java method that implements this algorithm and a main method to test it.

**Hint** The sum of every two integers in A is always less or equal to twice the maximum element.

(3 marks)

**Evaluation:**

|  |  |
| --- | --- |
| **Functionality**   * Correct result of running time * Correct experimental analyses code and visualization * Code demonstration and brief explanation of proof or analyses in a short video | 20%  40%  10% |
| **Object Oriented Design**   * Correct design of classes and methods similarly to chapter 3 examples. * Correct use of generics * Correct use of naming guidelines for classes, variables, methods, packages. | 15%  5% |
| **Friendly graphical display** | 10% |
| **Total** | 100% |

**Naming and Submission Rules:**

You must **name your Eclipse project** according to the following rule:

**YourFullname\_COMP254Labnumber**. Example: **JohnSmith\_COMP254Lab2**

You must name package names **com.exercisenumber.yourfirstname.yourlastname**, for example: **com.exercise1.john.smith**

Provide your **student number and full name as a comment** at the top of main method for each exercise.

**Archive your project in a zip file** named according to the following rule:

**YourFullname\_COMP254Labnumber.zip**

Example: **JohnSmith\_COMP254Lab2.zip**

Upload the zip file on eCentennial using the Assignment link.