COMP 2080 – Data Structure and Algorithms in Java

Assignment 1

Due Date: Monday, March 26th, 2018 (8:00 am)
Team Size: This is a group assignment (2-6 members).

Assignment Objectives:
Java ADT, Linked Lists, Recursion and Big-O efficiency.

Part 1- Big O Notation Study

The source file T*iming.java* (available on blackboard) contains some very mysterious functions namely, *mickey(), minnie(), donald(), goofy(), pluto(), gyro()* and *daffy()* and also *fact()*. Your task is in this first part of your assignment is to develop a driver (name it *TestTiming.java*) that analyzes the performance of these functions experimentally.

Background - Timing Methods
The snippet below is an example of how to use *StopWatch* (available on blackboard) to time functions.

```
Stopwatch timer =  new Stopwatch();
//initialize arguments
timer.start();
//code to be timed
timer.stop();

System.out.println(timer);
```

a. For method *daffy()* develop a new method (called by your driver) to test and time this function testing for values of *n* from 30 through 44. Plot your results/analysis.

b. Repeat a) with method *donald*()

c. Develop a new method (called by your driver) to test *mickey*(). To initialize, make an array of size n with *randomarr*(n). The initialization of the array should **not** be inside the timing calls. Use values of n of 1000, 2000, 4000 ... through to 8192000. Plot your results/analysis.

d. Develop a new method (called by your driver) to test *minnie*(). To initialize, make an array of size *n* with *randomarr*(*n*). The initialization of the array should **not** be inside the timing calls. Use values of n of 1000, 2000, 4000 ... through 256000. Plot your results/analysis.

e. Repeat d) with methods *goofy*(), and *pluto*().

f. Develop a new method (called by your driver) to test *gyro*(). For *gyro()*, make an array using *randomarr*(n) and call *pluto* on it first (*pluto* should not be included in the timing). Use values of n of 1000, 2000, 4000 … through 256000. Plot your results/analysis.

g. Develop a new method (called by your driver) to test *fact*(BigInteger n). For *fact*(BigInterger n), test values n from 1000 through to 64000, doubling *n* each time.

Example of how you can make a BigInteger:

```
BigInteger bign = BigInteger.valueOf((long) 1000);
```

A *BigInteger bign* that is not too large can be converted to an int using *bign.intValue*(). There are operations *.add*(), *.subtract*(), *.multiply*() and *.divide*(), each with a *BigInteger* argument utilize if needed.

- For <u>each</u> method, plot the time taken utilizing Microsoft excel for graphing. You should research how to plot data utilizing excel as this is used regularly in industry. Ideallly you will be plotting time take vs *n*. You should use these values as a basis to deduce your conclusion and estimate the **Big-O** for the respective function accordingly. Please state your Big-O effriciency conclusion for each method clearly **<u>on</u>** the graph.

- Note that the timer provided will not be accurrate for very small times (less than a millisecond). If any recorded times of a functions are too small, do not include those times in your graph/report (since they simply act as noice to your calculations and estimates).

- The time of the StopWatch in seconds can be obtained (as a double) as *timer.time();*

- For small values of the input size, the times will be noisy and the ratio will not be meaningful, so you may need to ignore the first few timing calculation values, concentrating on the ratio for larger inputs.

- Your deliverables for this first part of the assignment is the excel spreadsheet containing your plotting data with your conclusions/extimates of the Big-O efficiencies for each of the mysterious methods (deduce your conclusions on the excel spreadsheet) in **<u>addition</u>** to your driver (TestTiming.java) that tests and executes the methods in accordance with the instructions provided from questions a) through g).

<u>Part 2 – Java Docs LinkedSequence Study</u>

To be java developers, you must become acquainted with java documentation (java docs). Javadoc are specifically generated for documenting api in HTML format (from java source code). The HTML format is specifically used to add the convenience of hyperlinks to related documentation. Typically, java docs are created on a class-by-class basis. To read a java doc, simply navigate to the **index** page of the provided documentation folder, utilizing a web browser of your choosing.

For part 2 of this assignment, please review the java documentation on **LinkedSequence** (uploaded to blackboard – **doc.zip**). You are required to implement this class in its entirety (methods etc..). Please review this documentation carefully as it explains all you need to know to perform the implementation precisely. Naturally, the class will require a suitable driver that clearly demonstrates that each method is functioning as per specification. Please name your driver accordingly and place within the same project (ex. COMP2080_ASSIGN1-XXXX).

Part 3 – Recursion

For this third part of the assignment, you will implement and test two short recursive methods. With the proper use of recursion, none of these method should require more than a dozen lines of code.

Create a new file in your assignment one project called **RecursionPartIII.java (**place it in a suitable package). This file should contain a class with the recursive methods described below. You will also need to create a driver that allows you to test the methods.

1. Pattern

```
public static void pattern(int x, int y)
  // Precondition: x is less than or equal to y
  // Postcondition: This method prints a pattern of 2*(y-x+1) lines
  // to the stdout (console). The first line contains x asterisks, the next
  // line contains x+1 asterisks, and so on up to a line with y asterisks.
  // Then the pattern is then reversed backwards, going y back down to x.

  /* Example output:
     pattern(3, 5) will output:
     ***
     ****
     *****
     *****
     ****
     ***
  */
```

2. Level

```
public static void level(String title, int levels)
// Precondition: none
// Postcondition: This method outputs a string (title) followed by numbers
// of the form 1.1, 1.2, 1.3 etc… The levels input parameter, determines how //
many levels the section numbers have. To clarify, an input of levels equal //
to 2, would have section number of the form x.y. Likewise, if levels is
// equal to 3, would have section number of the form x.y.z. Please note the //
digits permitted in each level are only 1 – 9.

/* Example output:
   level("TITLE", 2) will output:

   TITLE1.1.
   TITLE1.2.
   TITLE1.3.

  //and end by printing

   TITLE9.7.
   TITLE9.8.
   TITLE9.9.
*/
```

Notes on Deliverables:

- Create a single eclipse project to house all your solutions.
- Since you are submitting a single project. Your project will need to be organized, that is, your solution(s) should use appropriately named packages.
- You must email your exported project (containing your graph/plot) using your **George Brown Email** using a valid George Brown email account.
- You are required to cc'd each member on your team as part of the submission to the professor (only 1 email per group is necessary).
- When emailing your assignment, please include your information in the body of your email:

  For example:

  COMP 2080 - Assignment 1
  Group Name: The Hackers
  Team Members : John Smith - 1234567
  Jane Doe - 8948393
  Bilbo Baggins - 4798327

- Name your project and exported project according to the following:

  COMP2080_ASSIGN1_**THE_HACKERS**

- Each java file (.java) should include a header

  ```
  //*********************************************************************************************************
  * Project:          < project name … >
  * Assignment:       < assignment # >
  * Author(s):        < author name …>
  * Student Number: < student number … >
  * Date:
  * Description:      <describe the java file and its purpose briefly only – 1 or 2 lines>
  *********************************************************************************************************//
  ```

- Be cautious **DO NOT** share your application with others. Complete failures will be assigned if code is shared. All assignments will be reviewed and analyzed strictly within these regards.
  If two groups (or more) provide equivalent solutions and /or assignments are the same (or very much alike) they will all get 0 marks and be reported to the faculty, so again, this is your warning, be cautious not to share your application and solutions with others.
- Late assignments are assigned a penalty of 10% per day.