COMP 2080 – Data Structure and Algorithms in Java

Assignment 1

Due Date: Friday, February 26th, 2016 (4:00 pm)
Team Size: This is an individual assignment.

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

Part 1 - (30 marks)

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 first
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 each of the method, plot the time taken utilizing Microsoft excel for graphing. You should research how to plot data utilizing excel as this is used extensively 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.

- 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 nclude those times in your graph/report (since thy 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 or a separate word document – be clear and explain your conclusions referring to your calculated data) in **addition** to your driver (TestTiming.java) that tests the methods in accordance with the instructions provided from questions a) through g).

<u>Part 2 – (30 marks)</u>

Create a file named *StringRecursion.java* that implements the described methods below. Additionally create a separate driver class (*TestRecursion.java*) to test the methods described below.

Please note the following restrictions for Part 2:

- Your methods **<u>must</u>** be recursive.
- **No** credit/grade will be given for any methods that employ iteration.
- **No** global variables (variables declared outside of the method) are allowed.

<u>HINT:</u> You may find it helpful to employ *substring*(), *charAt*() and *length*() methods of the java *String* class as part of your solutions.

a. **public static void printWithSpaces(String str)**
This method should use recursion to print the individual characters in the string str separated by spaces.

For example, printWithSpaces(“space”) should prodiced the following output:

```
s p a c e
```

The method should not return a value. The method should not do any printing if the empty string (“”) or the value **null** is passed in the parameter.

b. **public static String weave(String str1, String str2)**
This method should use recursion to return the string that is formed by “weaving” togeather the characters in the str1 and str2 to create a single string.

For example:

- weave(“aaaa”, “bbbb”) should return the string “abababab”
- weave(“hello”,”world”) should return the string “hweolrllod”

If one of the strings is longer that the other, its “extra” characters (the ones with no counterparts in the shorter string) should appear immediately after the “woven” characters (if any) in the returned string.

For example:

- weave(“recurse”, ”NOW”) should return the string “rNeOcWurse”.

Once again this method should not do any printing; it should simply return the resulting string. If **null** is passed in either parameter, the method should throw a “*COMP2080AssignmentInputException*”.

If the empty string (“”) is passed in for either string, the method should return the other string.

For example:

- weave(“hello”,””) should return “hello”
- weave(“”, “”) should return “”

c. **public static int lastIndexOf(char ch, String str)**
This method should use recursion to find and return the index of the last occurrence of the characters ch in the string str, or -1 if ch does not occur in the str.

For example:

- lastIndexOf('r',"recurse") should return 4
- lastIndexOf('p',"recurse") should return -1

This method should not do any printing; it should simply return the resulting string. The method should return -1 if the empty string ("") or the value or **null** value is passed as the second parameter.

Problem 3 – (40 marks)

A palindrome is a phrase that reads the same forward as it does backward.

For example, the following below are examples of palindromes:

1. “Taco cat” and “don’t nod”
2. “a man, a plan, a canal, Panama”

Please note, when determining if a string is a palindrome, we ignore characters that are not letters and we additionally ignore the case.

For this problem you are to do the following:

a. Write a java program that uses a stack to check for palindromes in each line of a text file. Try your program on the text file provided (*palindrones.txt* posted on blackboard).

Your program should output the palindromes that it finds in the document (or any document) fed to it.

For example:

```
java FindPalindromes palindromes.txt
"a man, a plan, a canal, Panama" is a palindrome.
"Don’t nod" is a palindrome.
"Taco Cat!" is a palindrome.
...
```

HINT / NOTE: You must write your own stack class for this problem (do not use any built-in stack or api). Your **must** also implement your stack using LinkedLists (**not** arrays, and **not** ArrayLists).

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) via blackboard email.
- To be clear, your exported project (a compressed file) should contain all your java source files and your graph/plot analysis. You can simply create a folder within you project to house the graph for example.
- Failure to comply with the desired deliverable will result in a loss of marks, so be diligent in preparing your submission. Diligence is expected in industry when delivering your solutions.
- When emailing your assignment, please include your information in the body of your email:

  For example:

  COMP 2080 - Assignment 1

  Team Members: John Smith - 1234567

- When emailing, cc' a copy to yourself for backup and time verification (important).
- Name your project and exported project according to the following:

  COMP2080_ASSIGN1_***LASTNAME_FIRST***

- 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 students (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.