CS2020: Data Structures and Algorithms (Accelerated)

Problems 1–3

Due: January 20th, before discussion groups

Overview. Your first job this week is to set up your environment for writing programs in CS2020. By the end of this problem set, you should be able to write simple Java programs, and measure their performance. Your second job this week is to familiarize yourself with the code provided for solving the "Document Distance" problem discussed in lecture.

Collaboration Policy. As always, you are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 1. (Getting up and running)

In CS2020, we will be writing programs in Java. We will be using *Eclipse* as our basic development environment. Eclipse is an open source IDE (integrated development environment) consisting of editing, debugging, and performance measurement tools.

In particular, we will be using the "Test and Performance Tools Platform" (TPTP) version of Eclipse which allows you to easily profile the performance of your programs. It can run on almost any platform, but we encourage you to use either Windows or Linux for ease of installation. You can find more information on Eclipse with TPTP at:

`http://www.eclipse.org/tptp/home/downloads/?ver=4.7.1`

You can download the all-in-one installation package for Eclipse with TPTP for Windows at:

`http://tinyurl.com/2cr3f6e`

You can download the all-in-one installation packages for Eclipse with TPTP for Linux at:

`http://tinyurl.com/35dvtak`

Even if you have Eclipse already installed, it may be easier to simply install a complete version with the TPTP package.

Problem 1.a. Install Eclipse with TPTP on your machine.

Problem 1.b. Start Eclipse, and specify a workspace where Eclipse can store your projects. Create a new Java project (File→New→Java Project) called CS2020-PS1. Within the new project, create a new package (File→New→Package) called sg.edu.nus.cs2020.

Problem 1.c. Within the cs2020 package, create a new class (File→New→Class) called PSOne. Within this class, create the following method:

```
static int MysteryFunction(int argA, int argB)
{
     int c = 1;
     int d = argA;
     int e = argB;
     while (e > 0)
     {
          if (2*(e/2) !=e)
          {
               c = c*d;
          }
          d = d*d;
          e = e/2;
     }
     return c;
}
```

Also, create the following main function:

```
public static void main(String args[])
{
     int output = MysteryFunction(5, 5);
     System.out.printf("The answer is: " + output + ".");
}
```

Run your program. What is the answer output in your solution? Submit PSOne.java, along with the output.

Problem 1.d. Use the profiling feature to determine how long your program runs for. Under (Run→Profile), select "Execution Time Analysis", and click Profile. Under "Execution Time Analysis," what is the Cumulative Time (in seconds) of MysteryFunction and main?

Problem 1.e. (Optional and very hard.) What is MysteryFunction calculating? If you set ($argA = 2$) and try a few examples, you might be able to guess. Notice how hard it is to determine with opaque variable names and no comments!

Note: For the remainder of the problems, we will assume that you can create appropriate projects and classes to contain your files. Hence, we will no longer specify in detail filename, class names, etc.

Problem 2. (Performance Experiments)

Problem 2.a. Write a Java program to calculate the following function:

$$\begin{aligned} f(n) &= n * (n-1) + f(n-1), \text{ if } n > 1 \\ f(n) &= n * n, \text{ if } n \leq 1 \end{aligned}$$

Specifically, your function should take as input an integer n, and return the value determined by the above recurrence . In addition, write a *main* function to test your function.

Problem 2.b. What is the asymptotic performance of your algorithm? Give your answer in big-O notation.

Problem 2.c. We now want to measure the real performance of your algorithm, in order to understand how its performance changes as n grows. Choose 6 different values of n, and profile your algorithm for each value, measuring how long it takes to calculate the function.

Notice that your program may be very fast, so it is hard to measure a single execution of the function. Instead, run the function many times (e.g., 1000 times) for each test value. For example:

```
int testValue = 16;
int output = 0;
for (int j=0; j<1000; j++)
{
     output = myfunction(testValue);
}
```

By looking at the cumulative running time of the function, you will get a good idea of its cost. Submit a table containing the 6 data points, along with a graph plotting n on the x-axis and time on the y-axis.

Problem 2.d. Using the graph from the previous part, determine (approximately) the constant hidden in the big-O notation from part (b). That is, use the experimental data from part (c) to estimate the function $T(n)$, i.e., the time it takes to calculate the function on n. (For example, in class we talked about a sorting algorithm that was $O(n^2)$, and when implemented in C++ on a particular machine, ran in time $T(n) = 0.01n^2$.) What is $T(n)$ for your algorithm? (You do not need to perform a real regression analysis; simply estimate the constant from the data available.)

Problem 3. (Document Distance Experiments)

In this question, you are going to determine whether a given mystery text was written by Shakespeare or by Kafka. Attached to the problem set, you will find three files:

- *hamlet.txt*, a play by William Shakespeare
- *metamorphosis.txt*, a short story by Franz Kafka
- *mystery.txt*, an exerpt from an unknown text

Using the provided VectorTextFile.java class, determine the author of the mystery text. Include in your submission your program to determine the identity of the mystery text, as well as the *document distance* between the mystery text and each of the two other given texts. (Recall that the bigger the angle between the two files, the more different they are.)