DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 352: Data Structure and Algorithms
Summer 2019
ASSIGNMENT 1
Due date and time: Sunday May 19th by 11:59 pm

Written Questions (50 marks):

Q1. What is the tightest bound big-Oh (O) time complexity for the following algorithm (shown in pseudocode) in terms of input size *n*? Show all necessary steps:

(a)
```
Algorithm MyAlgorithm (A,B)
   Input: Arrays A and B each storing n >= 1 integers.
   Output: What is the output? (Refer to part b below)
   Start:
   count = 0
   for i = 0 to n-1 do {
      sum = 0
      for j = 0 to n-1 do {
         sum = sum + A[0]
         for k = 1 to j do
           sum = sum + A[k]
      }
      if B[i] == sum then count = count + 1
   }
   return count
```

(b) Document a hand-run on `MyAlgorithm` for input arrays A = [1 2 5 9] and B = [2 29 40 57] and show the final output.

Q2. Consider the following code fragments (a), (b) and (c) where *n* is the variable specifying data size and *C* is a constant. What is the tight bound big-Oh time complexity in terms of *n* in each case? Show all necessary steps.

(a)

```
for (int i = 0; i < n; i = i + C)
   for (int j = 1; j < 1024; j = j*2)
            Sum[i] += j * Sum[i];
```

(b)

```
for (int i = 1; i < n; i = i*2)
   for (int j = 0; j < i; j = j+2)
            Sum[i] += j * Sum[i];
```

(c)

```
for (int i = 1; i < n; i = i * 2)
    for (int j = 1; j < i; j = j * 2)
        Sum[i] += j * Sum[i];
```

Q3. The number of operations executed by algorithms *A* and *B* are $12n^3 + 40n \log n$ and $5n^4 - 100n^2$ respectively. Plot the graphs of *A* and *B* to determine an *n0* such that *B* is greater than *A* for $n \geq n0$.

Q4. Answer the following questions:

a) Show that if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n) + e(n)$ is $O(f(n) + g(n))$.
b) $2^{n+1} + n^3$ is $O(2^n)$: Prove or disprove
c) Prove that 2^n is $O(n!)$

d) *log(n!)* is *O(n log n)*: Prove or disprove

Programming Questions (50 marks):

Referring to the slides from text book, Chapter 5, there are two versions of Fibonacci number calculators: *BinaryFib(n)* and *LinearFibonacci(n)*. The first algorithm has exponential time complexity, while the second one is linear.

Tetranacci numbers are a different version of Fibonacci numbers and start with four predetermined terms, each term afterwards being the sum of the preceding four terms. The first few Tetranacci numbers are: 0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, 208, 401, 773, 1490, …

Formally, the Tetranacci sequence above can be recursively defined as follows:
$T(0)=0, T(1)=0, T(2)=0, T(3)=1$
$T(n) = T(n-1) + T(n-2) + T(n-3) + T(n-4)$, for $n \geq 4$.

In this programming assignment, you will design in pseudo code and implement in Java two versions of Tetranacci calculators and experimentally compare their runtime performances.

a) In the first version, you will design the pseudocode and implement in Java an **exponential** version of Tetranacci calculator by directly implementing the previous recursive definition. You may call it *ExponentialTetranacci*(n). You will calculate ExponentialTetranacci(5), ExponentialTetranacci (10), etc. in increments of 5 up to ExponentialTetranacci(50) (or lower value if necessary for your timing measurement) and measure the corresponding run times. You need to use Java's built-in time function for this purpose. You should redirect the output to an *out.txt* file. You should write about your observations on timing measurements in a separate text or pdf file. You are required to submit the fully commented Java source files, the compiled executable, and the text/pdf files.
b) In the second version, you will design the pseudocode and implement in Java a **tail recursive** version of Tetranacci calculator. You may call it *TRTetranacci(n)*. Run your timing measurement as in a) above, starting with TRTetranacci(5), and incrementing in steps of 5 up to TRTetranacci(100) (or higher value if necessary). Provide the same deliverables as in a).
c) What is the tight bound time complexity (in terms of big-Oh) of the *TRTetranacci(n)* that you designed in part b) above? Explain your answer formally or intuitively.
d) Why *ExponentialTetranacci(n)* has an exponential time complexity? Explain intuitively.

Submit all your answers to written questions in PDF format only. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into two separate .zip files (one for Theory and one for Programming) and submitted via EAS (Refer to the course outline for more details on submission guidelines).