CPSC 319 – Winter 2024	**Released:**	January 12 (Friday)
Programming Assignment 1 (PA-1) **Comparing Algorithms**	**Due:**	January 27 (Saturday) @ 11:59 PM D2L Dropbox

GOAL

The goal of this assignment is to implement and compare different algorithms that solve the same problem.
In this case, the problem is to compute the n^{th} Fibonacci number.

Fibonacci numbers are defined by the Fibonacci numbers – denoted F_n – form a sequence where each number is the sum of the two preceding ones, starting from 0 and 1. Fibonacci numbers are defined as follows:

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ F_{n-2} + F_{n-1}, & n > 1 \end{cases}$$

which generates the Fibonacci series 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, … The algorithms range from one based on the recursive definition above to a more sophisticated algorithm that runs much faster for large values of *n*.

THE ALGORITHMS

Here are descriptions of the three algorithms.

Algorithm 1: Compute F_n using the recursive definition given above.

Algorithm 2: Initialize variables to represent the first two numbers in the Fibonacci sequence. Then, in a loop, compute each member of the sequence from the previous two. Stop when the n^{th} number is reached.

Algorithm 3: Using matrix exponentiation:

Global Variable:

✓ Define FM as a 2x2 matrix of integers

Function **Fibonacci**
Input: n (integer);
Output: F_n (integer);

Begin

- If (n = 0) **return** 0;
- Initialize $FM = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$
- Call **MatrixPower** (n - 1);
- **Return** the element that is in the first column of the first row of FM;

End

Procedure **MatrixPower**
Input: n (integer);
Output: *none*

```
Begin
   If (n > 1)
   Begin
      Call MatrixPower (n / 2);
      Update FM = FM * FM  // matrix multiplication

      If (n is odd)
         Update FM = FM * [1 1; 1 0]  // matrix multiplication
   End
End
```

INSTRUCTIONS

(1) Write a Java program by completing the TODO parts in the skeleton code provided for this PA-1 that implements each of the three algorithms as a function that computes the n^{th} Fibonacci number. It is recommended that you get your program to display a table that allows you to verify that each of the algorithms is coded correctly.

(2) Modify your program so that it can measure the execution time for each of the algorithms. Make sure to measure the performance of your algorithms for different ranges of Fibonacci sequences F_n. For instance, from F_0 to F_{100}, carefully observing the performance of each of the three algorithms for different intervals – e.g., for n < about 20, for 20 < n <= 50 (approximately), and for all n larger than that. Your program should display out data in the following format:

0 time to compute F_0 with alg. 1
1 time to compute F_1 with alg. 1
⋮
blank line
2 time to compute F_0 with alg. 2
3 time to compute F_1 with alg. 2
⋮
blank line
4 time to compute F_0 with alg. 3
5 time to compute F_1 with alg. 3
⋮

(3) Explore different ranges of Fibonacci sequences to provide flexibility for you to decide the specific intervals and ranges for your experiments. Make sure to analyze and interpret your results accordingly.

(4) Use the accompanying **Excel** file provided for PA-1 ("PA-1-plotting-results.xls") to produce plots showing execution time for each algorithm versus n. Note that to get illustrative plots useful, you may have to alter the range of n used for each algorithm, the number of values of n that are tried, and so on, as described above in item (2).

NOTE:	The functions **System.currentTimeMillis()** and **System.nanoTime()** are handy for timing execution. Make sure you experiment with both functions. See Section 4.1, Code Fragment 4.1 in the textbook for examples using System.currentTimeMillis() – also refer to the lecture slide set "02 - Algorithm Analysis - (2) Experimental Studies", slides numbers 11 – 17.
	Your functions may run faster than the resolution of the CPU clock. In that case, you must execute the function several times and divide the time by the number of executions to get a good measurement of the time.

QUESTIONS

(1) Concerning algorithm 1:
- **(a)** Does it perform redundant calculations (yes or no)?
- **(b)** Draw its recursion tree for the computation of F_6
- **(c)** How many times are F_2, F_3, … evaluated in the computation of F_6?

(2) What is the big-O running time for algorithm 2? Why?

(3) What is the big-O running time for algorithm 3? Why?

(4) Under what situations would you use each of the algorithms? Answer your question according to the experimental studies results (i.e., after computing the timing execution).

HAND-IN

(1) Cover sheet with your name and ID number only (file "PA-1 – Grading Summary.pdf" in D2L).
(2) A written report (PDF format) with your name and ID number, including the answers to the questions and your plots. Keep your answers short and to the point. The maximum length for a report is 2 pages plus a maximum of three plots.
(3) Your source code file that generates the data you used for your plots (in the specified format.
(4) Include all the above items in a zipped folder (standard ZIP) titled (asgmt-1-your LAST NAME-ID).zip before submission to the **D2L dropbox**.

LATE ASSIGNMENTS WILL NOT BE ACCEPTED

MARKING

- Assignment grades will be base equally on the two submitted components as described on the cover/grading sheet (first page of this document)
- Source code that does not compile or produces run-time errors will receive a grade of 0%.
- Code that produces incorrect output will receive a maximum grade of 1 (poor)

IMPORTANT

1. **Individual Work:** This programming assignment is to be completed **individually**. All solutions must be expressed through your own code and your own words.
2. **Originality Requirement:** Every submission should consist of your authentic work, excluding instances where code is directly borrowed from textbooks, lecture materials (slides, notes), or online resources, and provided by your TA.
3. **Acknowledgment of External Code:** When incorporating code from external sources, it is imperative to explicitly acknowledge the origin. This involves citing the sources in a scientific manner, providing details such as author(s), title, page numbers, URLs, etc.
4. **Documentation for Original and Borrowed Code:** For both your original code and any code obtained from textbooks, lecture materials, or the web, it is mandatory to include descriptive and technical comments. These comments should be comprehensive, providing insights into the purpose and functionality of each line or block of code.
5. **Copying another student's work constitutes academic misconduct**, a very serious offense that will be dealt with rigorously in all cases. Please read the sections of the University Calendar under the heading "Student Misconduct". If you are in doubt whether a certain form of aid is allowed, ask your instructor!

- The above points will be strictly enforced.
- Contact your TA if you have problems getting your code to work.
- Note that your code will be checked thoroughly for plagiarism by computer.

END OF THE PROGRAMMING ASSIGNMENT #1