Joe Del Rocco
jdelrocco [at] stetson [dot] edu
Assistant Professor of Practice, Computer Science
Stetson University
421 N Woodland Blvd, DeLand, FL, 32723
www.stetson.edu

CSCI 142
(all sections)
Spring 2022
Assignment

Assignment 4

Due: Monday 3/28/2022 11:59pm

Contents

Program

This program will give you some experience with empirical algorithm analysis. You will implement 4 different functions and then run some timings to see how they perform. You will then plot those timings in order to compare their growth (complexity). See the Example Output for examples of results from a fully implemented program and appropriate graphs.

Here is the link to the GitHub Classroom assignment:
https://classroom.github.com/a/qhdu3JLh

Sequencer

Do not make any changes to the provided `Main.java` file. You will create a class called `Sequencer` in a file called `Sequencer.java`. This class will contain 4 static methods. You can see the names of the 4 static methods called from `Main.java`. The names of the 4 static methods are: `fibRecursive`, `fibIterative`, `sumIterative`, and `sumGauss`. All four methods must take a single argument of type `int` and return a `long`. No other methods, fields or even a constructor need to be added to your `Sequencer` class.

fibRecursive(s)

This method should implement the standard elegant recursive algorithm for computing a specific Fibonacci number. The method takes a single argument of type `int` which represents which Fibonacci number to compute and return. We discussed this standard Fibonacci algorithm in CSCI141 and we will review it in class, but you can also find plenty of information about it online. There are also tables of the Fibonacci sequence online to help you verify your answers.

fibIterative(s)

This implementation of generating a specific Fibonacci number must not be recursive. It should be iterative. In other words, you should probably use a for loop. If you think about it, you only need 3 variables to compute any Fibonacci number: the previous Fibonacci number (which we'll call `prev1`), the previous previous Fibonacci number (which we'll call `prev2`), and a variable to hold the result of adding those 2 together. The first iteration of your for loop could compute the second Fibonacci number by adding `prev1` and `prev2`, which are 1 and 0 respectively. Then you could shift the values down, in the sense that the current result becomes `prev1` and `prev1` becomes `prev2`. The next iteration of your loop will compute the next Fibonacci adding those two previous numbers, and so on.

sumIterative(s)

This method is the standard naive solution of summing a sequence of integers from 1 to `n`. In other words, use a for loop to iterate from 1 to `n` and sum all the values up. Return the result.

sumGauss(s)

This method should use Gauss's implementation, often called Gauss's Formula.

$$sum = \frac{n(n+1)}{2} \tag{1}$$

Plot Your Timings

Once you have implemented your methods (correctly), you should see the timings in the right most column growing as the input size (`n`) grows. You now have all the makings of an empirical growth function: the input size and the amount of computing time. Create 2 graphs (plots) of input size vs. time (ms).

One of the graphs plots the timings of the 2 Fibonacci function implementations, and the other graph has the timings of the 2 Summation implementations. The graphs provide a visual way to "prove" and compare the growth of the implementations.

You can generate your graphs w/ any way you want, including: Google Sheets, Microsoft Excel, Apple Numbers, Desmos, Python and matplotlib, MATLAB, or even drawing them by hand. If you have never used any plotting software before, search the Internet for very basic tutorials, such as this.

Submission

You will commit and push your changes to your specific GitHub Classroom repository for this assignment. You are encouraged to use an IDE for development, but we will compile and run your program using the shell/terminal during grading, so it isn't a bad idea to test it in that environment to make sure it works. Please follow the directions in this assignment, make the requested code changes, and commit and push your changes any time before the due date. Please see the advice below; it is important for grading purposes. **Failure to follow these directions will result in a loss of points.**

Always make sure to:

- Keep all source files in the folder called `src`, which is one directory in from the root of your repo
- Do not commit multiple copies of the same named source file; modify the ones provided to you. In other words, do not make an old and new version of the same file
- The main starting source file should always be called `Main`
- When loading resources, do not use absolute paths to files on your drive; use relative paths
- Do not have the keyword `package` at the top of any files. Some IDEs add your files to a custom package by default. Please remove this, as it complicates grading.

Rubric

Task	Percentage
Assignment files not pushed to GitHub	Grade is 0%
General attempt at solving this assignment	50%
Correct implementations of algorithms	30%
Fibonacci algorithms graph w/ correct growth functions	10%
Summation algorithms graph w/ correct growth functions	10%
Total	100%

Example Output

Expected output (w/o timings) of a run of `fibIterative` and `sumGauss`:

```
-----------------------------
Method: fibIterative
                Input                Result              Timing
                   31               1346269              [illegible](ms)
                   32               2178309              [illegible](ms)
                   33               3524578              [illegible](ms)
                   34               5702887              [illegible](ms)
                   35               9227465              [illegible](ms)
                   36              14930352              [illegible](ms)
                   37              24157817              [illegible](ms)
                   38              39088169              [illegible](ms)
                   39              63245986              [illegible](ms)
                   40             102334155              [illegible](ms)
-----------------------------
Method: sumGauss
                Input                Result              Timing
                    1                     1              [illegible](ms)
                   10                    55              [illegible](ms)
                  100                  5050              [illegible](ms)
                 1000                500500              [illegible](ms)
                10000              50005000              [illegible](ms)
               100000            5000050000              [illegible](ms)
              1000000          500000500000              [illegible](ms)
             10000000        50000005000000              [illegible](ms)
            100000000      5000000050000000              [illegible](ms)
           1000000000    500000000500000000              [illegible](ms)
```

Examples of graphs of the Fibonacci and Summation algorithms:
NOTE: THESE GRAPHS ARE INCORRECT - DO NOT COPY THEM!
They are provided only as samples of appropriate graphs.