Assignment 1: Fibonacci Race

Data Structures, CSCI-UA 102, Section 7
Instructor: Max Sklar

Empirically test several approaches to computing values of the Fibonacci sequence.

Goals

- Learn how to test an algorithm empirically for its running time.
- Get into the practice of writing code and working with Java.
- Be able to differentiate between 3 different programming techniques, including recursion and dynamic programming.

Background

The Fibonacci sequence was described by Italian mathematician Fibonacci (Leonardo of Pisa) in his 1202 book of calculations (Liber Abaci). This book introduces the Hindu-Arabic numerals to Europe and its practical applications. The sequence was used as a way to model the population growth of rabbits if left unchecked.

The Fibonacci sequence starts with the numbers 0 and 1. Each value of the Fibonacci sequence is the sum of the previous 2 values. The first few values of the Fibonacci sequences are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

The Fibonacci sequence can be defined by the following recurrence:

$$\begin{aligned}
&F(0) = 0,\\
&F(1) = 1,\\
&F(n) = F(n-1) + F(n-2), \text{ for } n \geq 2.
\end{aligned}$$

Mathematicians in the nineteenth century found a **closed form** for this sequence using the **golden ratio**.

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}},$$

where ϕ is defined as:

$$\phi = \frac{1+\sqrt{5}}{2} \text{ (the golden ratio).}$$

Instructions

Your mission is to build 3 different Fibonacci functions. Each of them accepts an `int` as the input (say `n`), and the output is a `long`, which is the nth value of the Fibonacci sequence. Each of these functions takes a different approach:

1. **A recursive function:** This version receives an `int n` and calls itself for `n-1` and `n-2` (except in the base cases).
2. **Dynamic programming:** In this case, the function starts with the base cases of 0 and 1 loaded into local variables and uses a loop to progressively build up higher and higher values until it reaches the nth Fibonacci number.
3. **Closed form:** This is the direct approach that does not require loops or recursion, just standard mathematical functions.

Your main function should be in a class called `FibonacciRace` and take two integer arguments passed in through the command line. A skeleton of FibonacciRace.java is provided.

- `arg[0]` is the `maxInput`, which represents the largest Fibonacci value to test.
- `arg[1]` is the `maxTime`, which represents the largest amount of time (in milliseconds) that we're willing to wait for an algorithm to finish before canceling a higher input.

Start with `n=0` and run all 3 algorithms in order and time them. After each attempt, output a line of data using `System.out.println()`. Each line should have 4 pieces of information:

1. The algorithm name (either "RECURSE", "DYNAMIC", or "FORMULA")
2. The Fibonacci index `n`
3. The result `F(n)`
4. The running time in milliseconds

For example, if you ran the dynamic programming function on `n=3`, which returned the answer 2 and took 10 milliseconds, your code should output the following string:

```
"DYNAMIC\t3\t2\t10"
```

After running this for `n=0`, increase `n` and iterate again, up to and including `n=maxInput`. After every iteration, check the running time against `maxTime`. If the running time for an algorithm exceeds `maxTime`, you should still output that results, but going forward, you should not run that algorithm for higher values of `n`.

For high enough values of `n`, you will get overflow issues as the answer is larger than the highest allowed `long` value. There is nothing you are required to do to correct for those cases.

If your main function is given arguments that are either not integers or negative integers, it should throw an exception. If the arguments are not present, then it should also throw an exception.

Hints

- Use `Integer.parseInt(string)` to convert the arguments into `int`s. This function throws an exception if one of the arguments is not a valid `int`.
- The following tests a piece of code to see how many milliseconds it took to run:

```
long startTime = System.currentTimeMillis();
// Put your code to be measured in here
long endTime = System.currentTimeMillis();
long timeToRun = endTime - startTime;
```

- For the closed formula, make use of the `Math.sqrt()` function and the Golden Ratio:

```
public static final double GoldenRatio = (1 + Math.sqrt(5)) / 2;
```

- A tab is created in a string by using the backslash as an escape character and the letter t. So `"\t"` represents a tab, and `"1\t2\t3"` represents the numbers 1, 2, 3 separated by tabs.
- Running your code from the command line will take two steps: compiling and running. For example:

```
javac FibonacciRace.java
java FibonacciRace 12 1000
```

- Make sure that you test different outputs, including large and small values.

Code Style

Here are some tips for code style:

- Your code should be generally readable and understandable.
- You may select any reasonable style (i.e., for indentation or bracket placement), but it should be consistent.
- Please add comments in any part of the code where it is not obvious to the reader what the code is doing and why.
- Please use descriptive and efficient names for your classes, functions, and variables. Local counters can be still be called something like i for index.

First Project Notes

- For this project, you will have to download Java and set up your environment so that you can compile an run Java programs. You can select any text editor or IDE to write these files. Try doing this yourself, but if you run into problems, come to office hours or email the instructor or section leader for assistance.
- We won't be using Java packages in this course. Those are neccesarily for large projects, but the projects in this course will typically be only a handful of files.
- One you have compiled FibonacciRace.java, you can also compile and run the provided FibonacciRaceTest.java. This will check your output for common mistakes, but it does not guarantee the assignment is completely correct.

Submission

The entire submission should be a zip file containing all relevant files and folders and should be uploaded to Brightspace by the due date. It should be zipped up in file with your netid as the name. This includes:

- Your Java code, which should include one or more Java files and may or may not be organized into packages.
- Your `readme.txt` file (described below).
- An `output.txt` file, for which you run your code with the parameters 60 and 2000.

README

Included in the `readme.txt` file should be the following:

TIME SPENT: An estimate of the time spent working on this project.

NOTES: Include the difficulties and roadblocks encountered while working on this project, including notable bugs that needed to be overcome. Are there any quirks in your solution that we should be aware of? Was the outcome what you expected?

RESOURCES AND ACKNOWLEDGEMENTS: Disclose the sources you consulted for help on this project. Possible human sources include: NYU Staff (the instructor, TA, tutor), other students, and anyone who helped you with the project. Sources that you consulted, including code that you found online, in the book, and output from LLMs. Describe the nature of your LLM usage.

There will never be a grading penalty for a complete and accurate accounting of resources and acknowledgments.