Deniz Polat

22103590

101-001

CS 101 – INTRODUCTION TO ALGORITHMS & PROGRAMMING I

FALL 2021 – HOMEWORK 2

Due: December 8, 2021

I have implemented the two given algorithms for calculating the Fibonacci numbers, and used `nanoTime` method to measure the running times in nanoseconds for these two different algorithms. Algorithm A, `calcFibA` is a recursive method, and the other one, `calcFibB`, is non-recursive.

I made calls to `calcFibA` and `calcFibB` respectively for values n from 0 to 50 with increments of five in the main method that I wrote. The output is shown below:

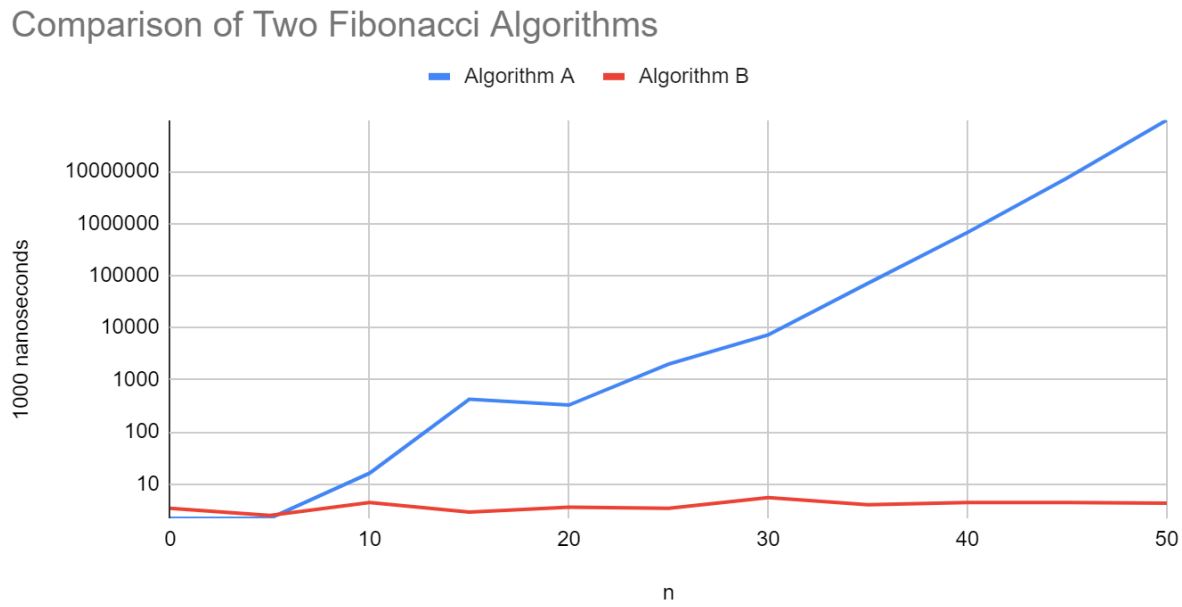
`CalcFibA`

```
1       2200   // for n=0, F_0 is calculated in 2200 nanoseconds.
2       2200   // for n=5, F_5 is calculated in 2200 nanoseconds.
3       16100  ...
4       421900
5       327700
6       1999400
7       7284600
8       70770600
9       675328500
10      7447803800
11      96504630900
```

`CalcFibB`

```
1       3400   // for n=0, F_0 is calculated in 3400 nanoseconds.
2       2500   // for n=5, F_5 is calculated in 2500 nanoseconds.
3       4400   // ...
4       2900
5       3600
6       3400
7       5500
8       4000
9       4400
10      4400
11      4300
```

In order to observe the differences between the running times of these two algorithms, I plotted a graph. Before the plotting, I divided all the nanoseconds values by 1000 so that the graph would look more logical in terms of large values. The graph is given below:



As it can be seen from the graph, as the values go larger, it took way more time for execution for the recursive method, Algorithm A. The reason for this case is how many times a recursive method is called, or called itself. For our case, it calls itself twice, `calcFibA(n - 1) + calcFibA(n-2)`, for each method. At each time that the method is called, n times, it calls itself for more number of times. This number can be represented by 2^n. That is why it takes a relatively large amount of time for n values to be executed.

On the other hand, the measured running time for Algorithm B is very close to a constant value. The reason is that contrast to the recursive method, the function is called for one time. In the recursive method, all the values are calculated again and again. In this method, the result is calculated by holding the previous values. That is why it is way faster to execute Fibonacci numbers with Algorithm B.

What if the n value goes to a larger value, like 100, instead of 50? As expected, it would take hours for the values as n goes to 100 to be executed with the recursive method, Algorithm A. With Algorithm B, as the n becomes larger, the result will eventually exceed `long.MAX_VALUE`. When it is exceeded, the algorithm will fail to calculate the correct values for Fibonacci numbers.

References

"Recursion." *Code Academy,* Codeacademy, 2021. www.codecademy.com/learn/algorithmic-concepts-java/modules/recursion-java/cheatsheet.