Jacob Esswein

CS 3130

Professoor Galina

9/22/2019

Project 1 - Part E

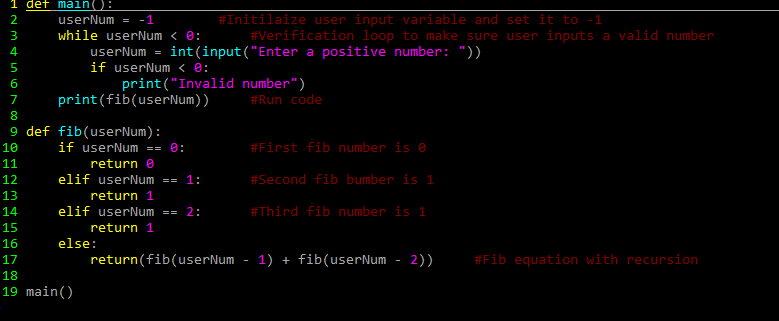
Recursive vs.. Iterative for The Fibonacci Sequence

**Objective:**

The objective of this project to was to compare the efficiency of calculating Fibonacci numbers recursively vs. iteratively, and to further our understanding of order of growth. We do this by first calculating the theoretical order of growth for both algorithms, then run our programs for find the experimental order of growth, which can be found in Part C of this project.

**Theoretical order of growth for Recursive and Iterative algorithms for the Fibonacci Sequence:**

**Recursive Implementation**



The time complexity of the recursive algorithm is: **Tn = Tn-1 + Tn-2 ; Tn ∈ O(Φ)n**

The recursive implementation of the Fibonacci sequence takes an exponential amount of time to run since it will repeat itself many times over (recalling itself/function) multiple times. For example, whenever we recursively call the Fibonacci sequence and we let n = 5 we get:

F(5) = 5

/ \

F(4)=3 F(3)=2

/ \ / \

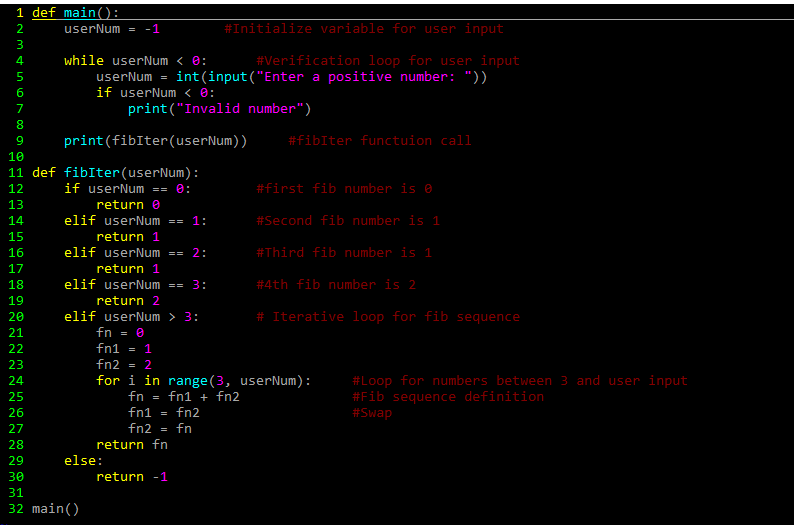
F(3)=2 F(2)=1 F(2)=1 F(1)=1

/ \

F(2)=1 F(1)=1

In this we see that F(3)=2 is calculated twice, which wastes time. From this, you can imagine that with an exponential n, that the amount of wasted time will grow rapidly.

**Iterative Implementation**



The time complexity of an iterative algorithm for the Fibonacci sequence is **Tn ∈ O(n)**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Code run** | **Cost** | **# of times run** |
| 1 | **If** n **is** 0 | C1 | 1 |
| 2 | **Return** 0 | C2 | 1 |
| 3 | **If** n **is** 1 | C3 | 1 |
| 4 | **Return** 1 | C4 | 1 |
| 5 | **If** n **is** 2 | C5 | 1 |
| 6 | **Return** 1 | C6 | 1 |
| 7 | **If** n **is** 3 | C7 | 1 |
| 8 | **Return** 2 | C8 | 1 |
| 9 | Fn = 0 | C9 | 1 |
| 10 | Fn1 = 1 | C10 | 1 |
| 11 | Fn2 = 2 | C11 | 1 |
| 12 | **For** I **to** n | C12 | (n-1)+1 =n |
| 13 | Fn = fn1 + fn2 | C13 | N |
| 14 | Fn1 = fn2 | C14 | N |
| 15 | Fn2 = fn | C15 | N |
| 16 | **Return** fn | C16 | 1 |

C1+C2+C3+C4+C5+C6+C7+C8+C9+C10+C11+n(C12+C13+C14+C15)+C16 **∈ O(n)**

The iterative algorithm/function will store the two most recently calculated numbers, then use those two numbers to calculate the next number in the Fibonacci sequence. By doing this, we eliminate the wasted time that we would normally have in a recursive algorithm/function, due to the reduced amount of calculations.

**Experimental Results for Recursive Algorithm:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **Trial 1** | **Trial 2** | **Trial 3** | **Trial 4** | **Trial 5** | **Average** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 3.814697265625e-06 | 3.337860107421875e-06 | 2.384185791015625e-06 | 2.6226043701171875e-06 | 2.6226043701171875e-06 | 2.956390380859375e-06 |
| 10 | 1.8358230590820312e-05 | 1.6689300537109375e-05 | 1.6689300537109375e-05 | 1.621246337890625e-05 | 1.6450881958007812e-05 | 1.6880035400390624e-05 |
| 15 | 0.0001780986785888672 | 0.0001766681671142578 | 0.0001761913299560547 | 0.00017595291137695312 | 0.00017499923706054688 | 0.00017638206481933593 |
| 20 | 0.00017638206481933593 | 0.0018641948699951172 | 0.001851797103881836 | 0.0018737316131591797 | 0.0018537044525146484 | 0.0018609523773193359 |
| 25 | 0.020853042602539062 | 0.02068805694580078 | 0.02251887321472168 | 0.021716833114624023 | 0.0206148624420166 | 0.02127833366394043 |
| 30 | 0.2355027198791504 | 0.23641037940979004 | 0.23279619216918945 | 0.24962329864501953 | 0.23133563995361328 | 0.23713364601135253 |
| 35 | 2.5994772911071777 | 2.6273508071899414 | 2.746936798095703 | 2.6630730628967285 | 2.7718207836151123 | 2.6817317485809324 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **Trial 1** | **Trial 2** | **Trial 3** | **Trial 4** | **Trial 5** | **Average** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 3.5762786865234375e-06 | 2.6226043701171875e-06 | 2.1457672119140625e-06 | 2.1457672119140625e-06 | 1.6689300537109375e-06 | 2.4318695068359373e-06 |
| 10 | 4.291534423828125e-06 | 2.6226043701171875e-06 | 2.384185791015625e-06 | 1.9073486328125e-06 | 1.9073486328125e-06 | 2.6226043701171875e-06 |
| 15 | 4.76837158203125e-06 | 3.337860107421875e-06 | 2.86102294921875e-06 | 2.6226043701171875e-06 | 2.384185791015625e-06 | 3.1948089599609376e-06 |
| 20 | 5.4836273193359375e-06 | 3.814697265625e-06 | 3.0994415283203125e-06 | 2.86102294921875e-06 | 2.384185791015625e-06 | 3.528594970703125e-06 |
| 25 | 5.0067901611328125e-06 | 4.291534423828125e-06 | 3.5762786865234375e-06 | 3.0994415283203125e-06 | 2.6226043701171875e-06 | 3.719329833984375e-06 |
| 30 | 5.4836273193359375e-06 | 4.291534423828125e-06 | 3.814697265625e-06 | 3.337860107421875e-06 | 3.337860107421875e-06 | 4.0531158447265625e-06 |
| 35 | 5.7220458984375e-06 | 4.0531158447265625e-06 | 4.0531158447265625e-06 | 3.5762786865234375e-06 | 3.5762786865234375e-06 | 4.1961669921875e-06 |

**Experimental Results for Iterative Algorithms:**

Based on the results seen in the graph above, we can conclude that the recursive algorithm has an exponential time complexity, while the iterative function has a linear time complexity.

**Conclusion:**

From this research we can conclude that the recursive algorithm grows faster due to it having to perform the same computations multiple times, while the iterative function does not, meaning its execution time is a lot less. So in all, recursive **O(Φ)n** and iterative **O(Φ)n**