Name: Carlos Carrillo-Calderon
Date: 11/30/2015
Course: CS_162

Lab 10

Recursive Fibonacci Function: This function recursively performs the Fibonacci sequence from 0 to 40. Since this function takes significant running time, 40 was the higher sequence number I could make without making the processor freeze.

Non-recursive Fibonacci Function: This function iteratively performs the Fibonacci sequence from 0 to 4000. Unlike the recursive Fibonacci function, this function allowed me to iterate it as many times as I wanted without making the processor freeze.

Not-tail recursive factorial function: This function recursively calculates factorial numbers using a single recursion approach.

Tail recursive factorial function: This function calculates factorial numbers using a double recursion approach.

Section to be tested	Input provided	Expected Output	Actual Output
NON-RECURSIVE -FIBONACCI FUNCTION	Numbers from 0 to 4000	A shorter running time than the Fibonacci recursive function.	
RECURSIVE -FIBONACCI FUNCTION	Numbers from 0 to 40	A longer running time than the Fibonacci non-recursive function.	
NOT-TAIL RECURSIVE FACTORIAL FUNCTION	12! 90 million times	A different running time than the tail recursive factorial function.	
TAIL RECURSIVE FACTORIAL FUNCTION	12! 90 million times	A different running time than the not-tail recursive factorial function.	

Analysis and discussion of results

First of all, I had to make the Fibonacci sequence longer in the non-recursive function (40x100) in order to create a large enough running time to be displayed on flip. Otherwise, it would only display 0 (something that does not happens in my Xcode).

But, even though the non-recursive function calculates 100 times more Fibonacci numbers than its recursive counterpart, it still takes a lot less running time than the recursive version. This confirms that some recursive functions can actually take a longer running time than iterative functions. These results also support Gaddis's (2014) affirmation, which states that recursive algorithms, such the Fibonacci sequence, "tend to be extremely inefficient and should always be avoided in favor of iteration" (924).

Something similar happens in the case of the not-tail recursive factorial function versus its tail recursive counterpart. In this case, the not-tail recursive factorial function takes less running time than the tail recursive function. It seems that the reason is that the number of function calls in the tail recursive factorial function is higher compared to the not-tail version. Consequently, it may cause a longer running time for the tail recursive factorial function. However, the difference is not as evident as in the case of the Fibonacci algorithm, even though there are more function calls in the not-tail recursive factorial function.

In conclusion, we could observe 2 different behaviors in terms of running time regarding these functions. First, it seems that the recursive function takes a lot more running time than its non-recursive counterparts. Second, it seems that the more function calls the recursive function has, the more running time it takes. That seems to be the case of the tail recursive function, which has 1 more function call than the non-tail recursive function. Also, this lab has been useful to get in the habit of including citations in the comments for each file, which is an important convention to always fallow in software development.