Lab 10 – Analysis + Reflection

Predictions:

The non-recursive Fibonacci function iterates through a while loop from 2 to N adding the first and second numbers to get the next number in the sequence. This is fairly straightforward for a computer, so I expect this will go quickly.

The recursive Fibonacci function calls itself with n-1 and n-2 over and over again until it reaches 1 and 0 (the first 2 numbers of the Fibonacci sequence. Knowing what I know about computers, I expect this to take longer for the computer because it involves more functions being on the stack while trying to reach 1 and 0. Thus, it is more memory intensive and should take longer.

N	RECURSIVE TIME	ITERATIVE TIME
1	0	0
5	.002	.001
10	.001	0
15	.001	.001
20	.003	.001
30	.019	0
40	1.021	0
41	1.638	.001
42	2.616	0
43	4.371	0
44	7.003	0
45	11.512	0
46	17.945	0
47	28.985	0
48	47.897	0
49	76.043	0
50	took too long	took too long
60	took too long	took too long
70	took too long	took too long

Analysis:

These results speak for themselves. It is clear to me that iterative Fibonacci is efficient while recursive Fibonacci is wildly inefficient. I can understand why though. If you think about when N is in the 40's, there are numerous Fibonacci function calls on the stack waiting for 0 and 1 to be reached so that the function calls can be popped off the stack. That is a lot of functions for the computer to keep track of at one time. Meanwhile, the iterative Fibonacci only has to worry about keeping track of 3 variables, rather than the countless function calls that the recursive method utilizes.

It begs the question of why even use recursive functions at all. In this day and age of speed and performance, it seems to be inefficient. I know we have learned that recursive functions can simplify code, but this lab has me wondering if simplification is worth the drawbacks in performance.