Clinton Hawkes

CS162-400

Lab 10 Summary

Predictions before lab:

Prior to beginning the lab, I sat and pondered what the results of the computations from both functions would look like. I expected the iteration to compute each Fibonacci number very quickly, and I figured it would scale linearly. This is because each increase in N of +1 means the functions only needs to loop and check one more set of numbers.

As for the recursive function, I expected it to take a lot longer to calculate the nth term in the sequence as the size of N grew. This is due to each increase of +1 in N requires an additional recursive function call to find the next term. That one additional function call requires two additional function calls, and those two additional function calls requires two more additional function calls each. This continues until the base case of N=0 is reached. The number of recursive function calls is almost doubled with every +1 to N. So I figured the time requirement would grow exponentially.

Results:

***Time is in system clock ticks.

N	Recursive Time	Iterative Time
18	33	12
19	43	11
20	67	10
21	160	19
22	386	27
23	470	18
24	783	22
25	903	13
26	1420	12
27	2399	19
28	4676	20
29	8166	21
30	13035	18
31	14771	19
32	22746	21
33	28136	18
34	46794	28
35	70770	20

36	104754	17
37	162718	16
38	254704	21
39	401534	27
40	645112	21
41	1034875	20
42	1682069	24
43	2707095	22
44	4399379	23
45	7110517	20
46	11386216	20
47	18439906	17
48	29864179	12
49	48847519	19
50	78814021	30
51	126336192	17
52	211825954	16
53	332221216	29
54	538145303	24

I began recording the results of the computations when N = 18. This is the Fibonacci term at which the times required to calculate began to diverge. As you can see, the number of system clock ticks for the recursive function to calculate the Fibonacci term grew with each +1 of N. As I predicted, the growth is indeed exponential.

The time required for the iterative function to calculate the Fibonacci term did not increase. I thought it would slowly increase as N grew larger. I suspect the time requirement for the iterative function did not increase in this test/lab because the N is not large enough to see the slow linear growth. With modern day processors running at 4 Ghz (4 billion clock cycles every second), an N of even 10,000 would barely start to show the impact.
*** I tested the iterative function at N = 10,000 and it took 80 system clock ticks.

I stopped increasing the size of N at 54, because it took my computer almost 10 minutes to compute the result for the recursive function while the iterative function took a small fraction of a second. I figured that large of a discrepancy between the two function’s computational time requirements fulfilled the requirements of this lab.

One surprising observation that I did not expect was that the time required for each additional value of N followed the golden ratio. I knew that the value of each additional Fibonacci term in the sequence converges on the golden ratio (1.618), but I did not know the time required for the calculation would also follow that pattern. It was a surprising find.

I noticed that the time required for the iterative function varies between ~10 and ~30. This was expected. When we are dealing with such short time segments, if any other process on the computer is using the cpu, my program has to wait a few clock cycles. This causes the variation that you see.

This was a valuable lab for me. Now that I am comfortable with c++, I need to start thinking about the most efficient way to tackle the problems I am faced with instead of just writing the easiest solution to the problem. For Fibonacci terms, iterative is definitely the most efficient.