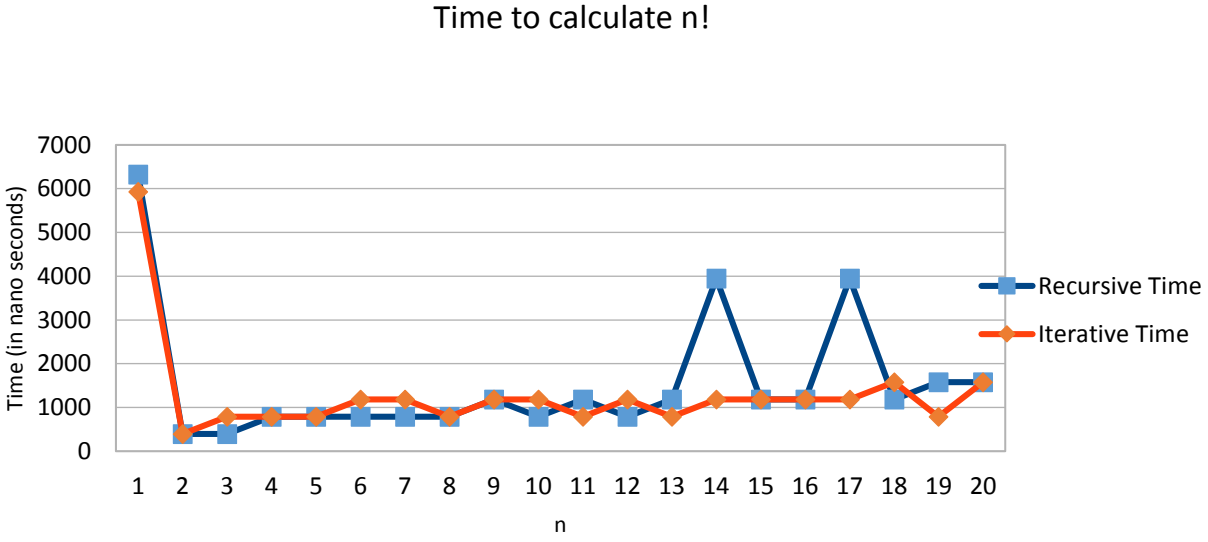
Test Recursive Factorial Start		
n	**result**	**Recursive Time**
1	1	6322
2	2	395
3	6	395
4	24	790
5	120	790
6	720	790
7	5040	791
8	40320	790
9	362880	1185
10	3628800	790
11	39916800	1186
12	479001600	791
13	6227020800	1186
14	87178291200	3951
15	1.30767E+12	1185
16	2.09228E+13	1185
17	3.55687E+14	3951
18	6.40237E+15	1185
19	1.21645E+17	1581
20	2.4329E+18	1580
Test Recursive Factorial End		

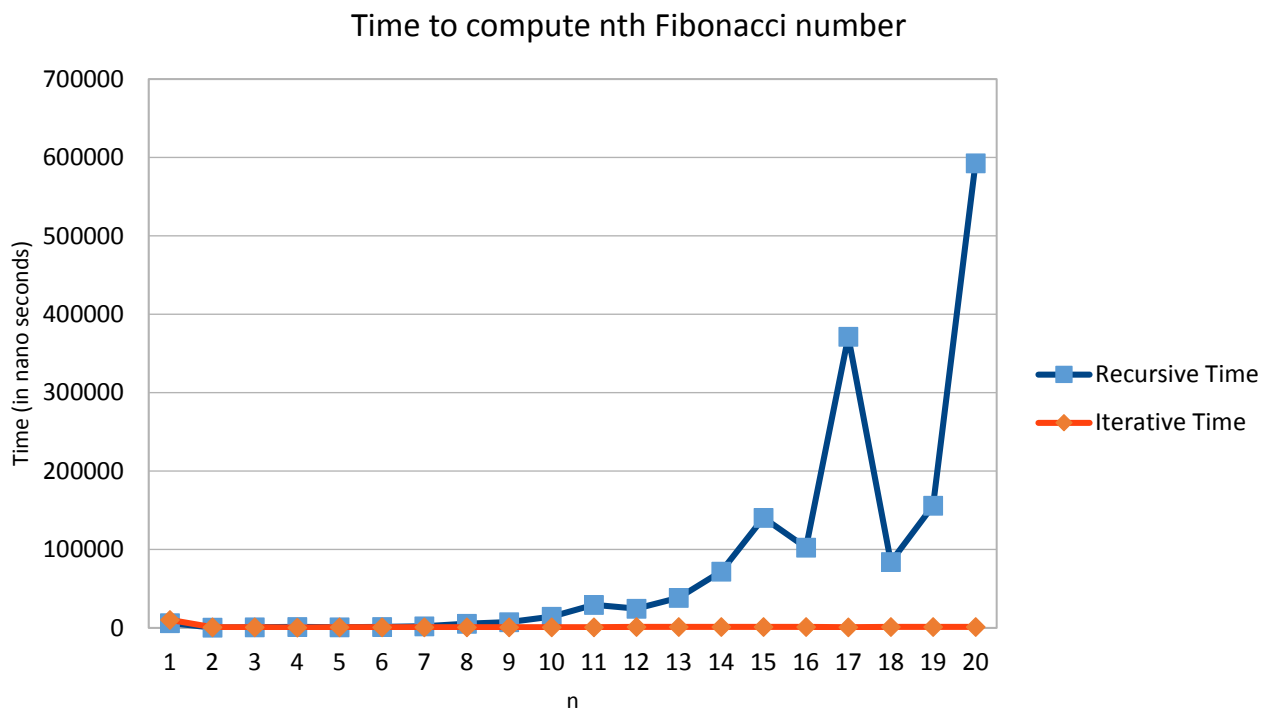
Test Iterative Factorial Start		
n	**result**	**Iterative Time**
1	1	5927
2	2	395
3	6	790
4	24	790
5	120	790
6	720	1185
7	5040	1185
8	40320	791
9	362880	1185
10	3628800	1185
11	39916800	790
12	479001600	1185
13	6227020800	790
14	87178291200	1185
15	1.30767E+12	1185
16	2.09228E+13	1185
17	3.55687E+14	1185
18	6.40237E+15	1580
19	1.21645E+17	790
20	2.4329E+18	1580
Test Iterative Factorial End		



I found that calculating the nth factorial, or n!, had fairly consistent time between iterative and recursive implementations. This is because the iterative and recursive implementations do essentially the same work, and the recursive implementation produces at worst one recursive call, resulting in no extra work compared to the iterative implementation.

Test Recursive Fibonacci Start		
n	**result**	**Recursive Time**
1	1	6322
2	1	395
3	2	790
4	3	1185
5	5	791
6	8	1185
7	13	1975
8	21	5531
9	34	7507
10	55	14223
11	89	29633
12	144	24496
13	233	38324
14	377	71909
15	610	140657
16	987	102332
17	1597	371397
18	2584	84157
19	4181	156066
20	6765	592654
Test Recursive Fibonacci End		

Test Iterative Fibonacci Start		
n	**result**	**Iterative Time**
1	1	10273
2	1	791
3	2	790
4	3	395
5	5	791
6	8	790
7	13	791
8	21	790
9	34	790
10	55	790
11	89	790
12	144	1185
13	233	1185
14	377	1185
15	610	1185
16	987	1185
17	1597	790
18	2584	1185
19	4181	1185
20	6765	1185
Test Iterative Fibonacci End		



It appears that when n is greater than 9, the time to calculate the nth fibonacci number using the recursive implementation increases at an exponential rate. This makes sense, as the recursive implementation creates two new function calls each time it recurses, resulting in an exponential growth of work. This is a stark contrast to the iterative implementation, which seems to require approximately O(n) time.