Analysis of Algorithms

This is an empirical and mathematical time complexity analysis of three algorithms used to generate the Nth Fibonacci number. fibonacci_direct() is an iterative algorithm. fibonacci_recursive() is a recursive algorithm and fibonacci_memo() is a memoized recursive algorithm. The empirical analysis tables provide run time and factor increase of run time for returning increasing (N) fibonacci numbers along with a graph charting the run time growth of the three algorithms. The mathematical analysis seeks to determine the complexity of the algorithms on the basis of their idealized, mathematical qualities.

Empirical Complexity Analysis

fibonacci_direct()

N	Time	Factor increase
2	0.0000003860	
4	0.0000004950	1.28
8	0.0000006120	1.24
16	0.0000008690	1.42
32	0.0000016530	1.90

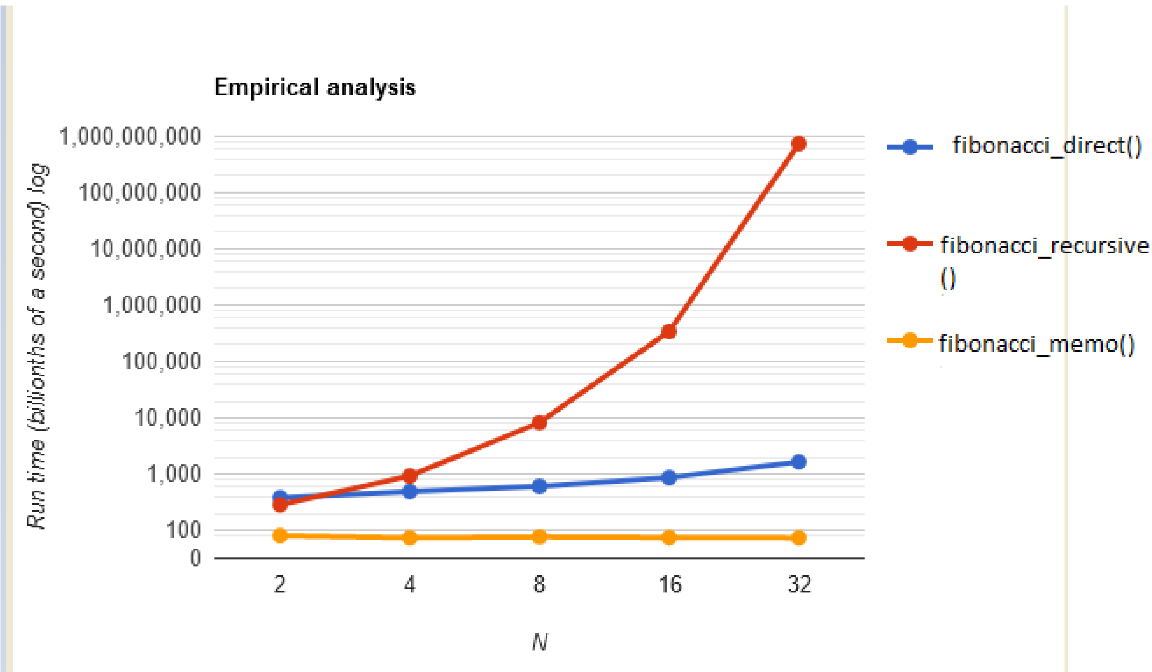
fibonnaci_recursive()

N	Time	Factor increase
2	0.0000002860	
4	0.0000009300	3.25
8	0.0000082640	8.89
16	0.0003417300	41.35
32	0.7469503570	2185.79

fibonacci_memo()

N	Time	Factor increase
2	0.0000000810	
4	0.0000000740	.91
8	0.0000000770	1.04
16	0.0000000750	.97
32	0.0000000740	.99

Log plot of run time for algorithms



Based on the empirical results, the direct function appears linear O(n) with factor increases growing closer to two with each doubling of N; the recursive is increasing at a rate much faster than polynomial ($32^2/16^2 < 2185$) and thus would appear to have an exponential $O(2^n)$ upper bound and the memoized function appears constant O(1).

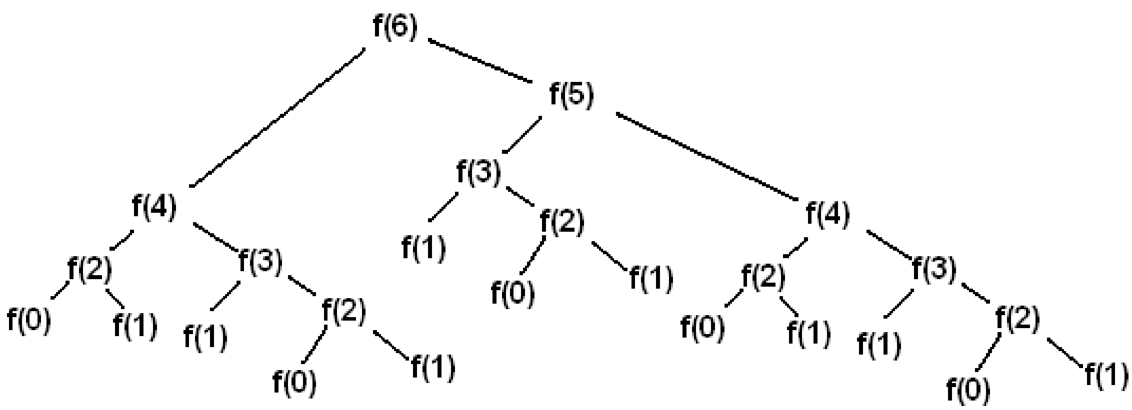
Mathematical complexity analysis

fibonacci_direct()

The direct function contains a single loop, so by the loop counting method we would expect a complexity of O(n).

fibonacci_recursive()

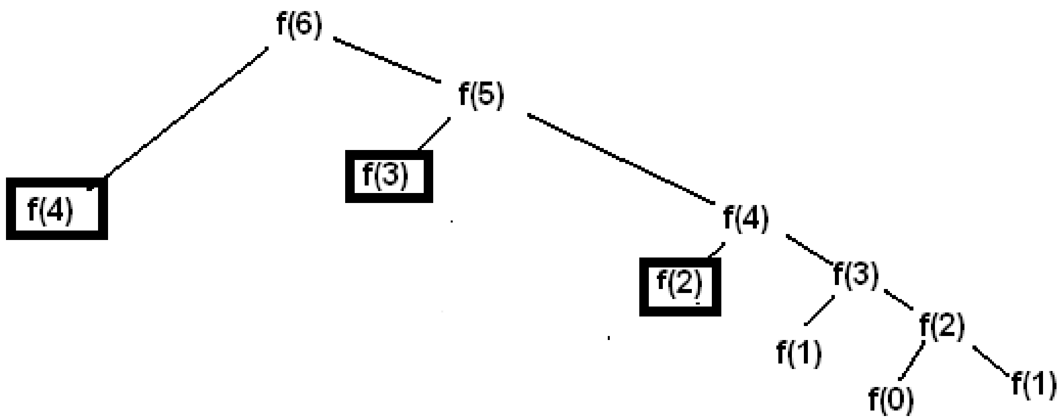
If we look at a fibonacci recursive call tree -



we see that each additional level of n nearly doubles the number of calls. It's not quite a doubling, but approximates closer and closer to the golden ratio, an irrational number 1.618033988749.... The ratio for f(3) = 1.66..., f(4) = 1.8, f(5) = 1.66.., f(6) = 1.66.., f(7) = 1.6. Because of the approximate doubling we would expect, at most, a complexity of $O(2^n)$.

fibonacci_memo()

In the memoized fibonacci call tree each additional level of n only requires calling the function once more along with retrieving the stored memo, so despite the empirical results we would expect the function to grow linearly with increasing n with complexity O(n) rather than be constant.



f(6)
f(5)
f(4)
f(3)
f(4)
f(2)
f(3)
f(1)
f(2)
f(0)
f(1)