Homework 3 - Analyze Fibonacci Brute Force and Recursive

Chris Winsor
Algorithms 2223
3/5/2016

Executive Summary: The Brute Force algorithm is linear in n. We are able to generate fibonacci series of 60,000 elements. The Recursive approach has an extremely high growth rate which my analysis shows to be higher than fibonacci(n). Our resources are exhausted at n=40.

Brute Force Algorithm:
Pseudocode for the Brute Force implementation is in Figure A. It shows line #4 is executed the most number of times. There is a single loop which suggests the algorithm will be linear in n.

At the bottom of Figure A we sum the per-line time-costs. Similar terms are combined and we observe T(n) is in indeed linear with respect to n. We conclude:

$$\text{fibBrute}(n) \in \Theta(n)$$

	fibBrute(n)	# times executed	time cost
1	if n < 2 return n	1	k1
2	fn2 = 0	1	k2
3	fn1 = 1	1	k3
4	for i = 2 to n	n-1	k4
5	fn = fn2 + fn1	n-2	k5
6	fn2 = fn1	n-2	k6
7	fn1 = fn	n-2	k7
8	return fn	1	k8

$$T(n) = k_1 + k_2 + k_3 + k_8 + (n-1)(k_4) + (n-2)(k_6 + k_7 + k_8)$$

$$T(n) = c_1 n + c_2$$

Figure A

When running our benchmarking tool the results are shown in Figure B. The results here are inconsistent with our expectations, specifically we do not see linear behavior as n grows. Instead we observe what appears to be high order growth. We suspect the Python data structures used for the fibonacci series are involved, but it is an opportunity for future study.

n	T(fibBruteForce(n))
10000	6538
10001	9548
10002	6503
20000	22159
20001	20539
20002	18089
30000	37608
30001	36547
30002	38070
40000	64057
40001	63135
40002	61651
50000	91430
50001	94213
50002	95636
60000	132325
60001	129856
60002	132349

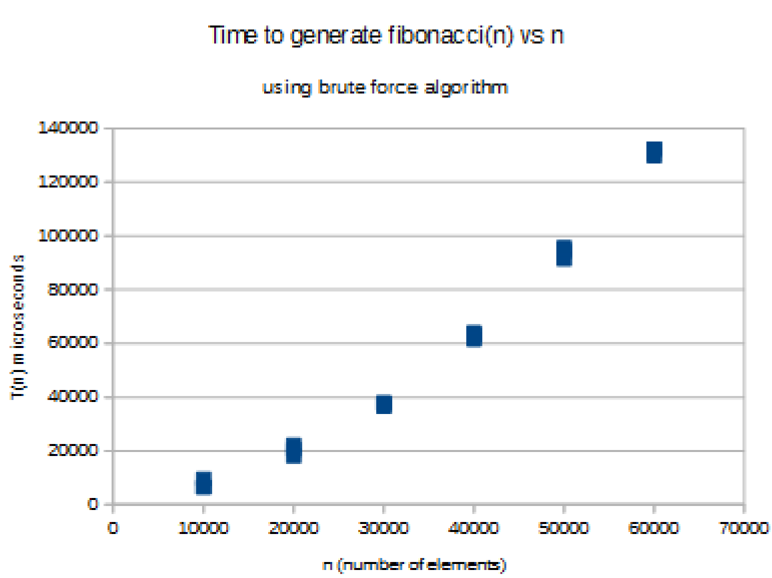

Figure B

Recursive Algorithm:
We next analyze the recursive implementation of fibonacci. Pseudocode for this is in Figure C. We observe the characteristic recursion in lines 3 and 4. Lines 1,2, and 5 are 'work' associated with the function. This algorithm is NOT divide-and-conquor so Master Theorem can not applied, instead we use Forward Substitution.

The initial steps of forward substitution are shown in Figure C, with the remaining work on following pages. The pattern that emerges is one if a fibonacci series itself - in other words - T(n) grows at a rate that is similar to the fibonacci series itself. This is an extremely high growth rate.

The results of the experiment are consistent with our expectations. Our environment is able to achieve a fibonacci sequence of length 30 before it reaches resouces are exhausted.

Our analysis of the forward substitution is on the following pages.

```
fibRecursive(n)

1 if n=0 return 0              k1(compare)
2 if n=1 return 1              k2(compare)
3 n1 = fibRecursive(n-1)
4 n2 = fibRecursive(n-2)
5 return n1 + n2               k3(sum)
```

using forward substitution

n	x(n)
0	k_1
1	$k_1 + k_2$
2	$fib(1) + fib(0) + k_1 + k_2 + k_3$
3	$fib(2) + fib(1) + k_1 + k_2 + k_3$
4	$fib(3) + fib(2) + k_1 + k_2 + k_3$
...	

$$T(n) = T(n-1) + T(n-2) + k$$

we have initial conditions $T(0) = k_1$ $T(1) = k_1$

Figure C

n	T(fibRecursive(n))
5	0
10	502
15	1534
20	15590
25	172969
30	844711

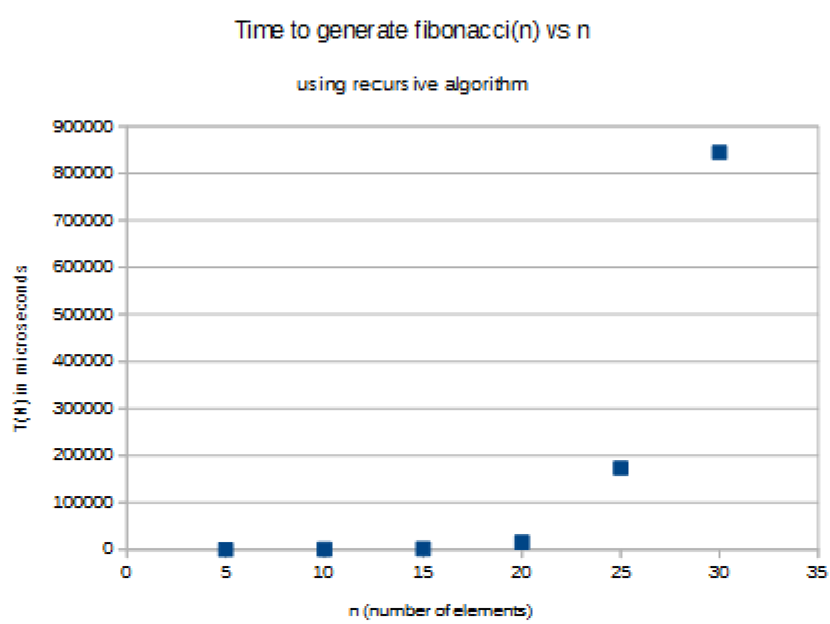

Figure D

Forward Substitution on FibbonacciRecursive(n)

n	$T(n)$		
0	k_1		c_0
1	k_1+k_2		c_1
2	$x(1)+x(0)+k_1k_2k_3 =$	k_1+k_2	c_0
		k_1	c_1
		$k_1+k_2+k_3$	c_7
3	$x(2)+x(1)+k_1k_2k_3 =$	k_1	c_0
		k_1+k_2	c_1
		$k_1+k_2+k_3$	c_7
		k_1+k_2	c_1
		$k_1+k_2+k_3$	c_7
4	$x(3)+x(2)+k_1k_2k_3$	k_1	c_0
		k_1+k_2	c_1
		$k_1+k_2+k_3$	c_7
		k_1+k_2	c_1
		$k_1+k_2+k_3$	c_7
		k_1	c_0
		k_1+k_2	c_1
		$k_1+k_2+k_3$	c_7
		$k_1+k_2+k_3$	c_7

c_0	c_1	c_7	
1			
0	1		
1	1	1	1
1	2	2	2
2	3	4	(2+1) 3+1

5 $x(4) + x(3) + k_1 + k_2 + k_3 =$

k_1	c_0	3	5	7	
$k_1 + k_2$	c_1				
$k_1 + k_2 + k_3$	c_7				
$k_1 + k_2$	c_1				
$k_1 + k_2 + k_3$	c_7				
k_1	c_0				
$k_1 + k_2$	c_1				
$k_1 + k_2 + k_3$	c_7				
$k_1 + k_2 + k_3$	c_7				
k_1	c_0				
$k_1 + k_2$	c_1				
$k_1 + k_2 + k_3$	c_7				
$k_1 + k_2$	c_1				
$k_1 + k_2 + k_3$	c_7				
$k_1 + k_2 + k_3$	c_7				

4+2+1

$T(5) = T(3) + T(4) + c_7$

The time-cost of $x(n)$
is the time-cost of $x(n-1)$
plus the time-cost of $x(n-2)$.
It itself is a fibonacci series!

this is a very high rate of growth...

6 $x(5) + x(4) + k_1 + k_2 + k_3$

c_0	c_1	c_7
$3+2=5$	$5+3=8$	
fib(5)	fib(6)	
fib(n-1)	fib(n)	

$fib^{+}(n) = fib^{+}(n-1) + fib^{+}(n-2) + 1$

12

This third term grows faster than fib()

Thus fib() is a lower bound

$$\boxed{\text{FibonacciRecursive}(n) \in \Omega(\text{fibonacci}(n))}$$