ANLY-550 Assignment 1

Yilun Zhu

Collaborate with: Siyao (Logan) Peng

1 Fibonacci

Note: Not clear the algorithm should: (1) run one Fibonacci number as large as possible or (2) run Fibonacci numbers one by one and see how many FIbonacci numbers the algorithm can run in a minute. For this problem, I take the second one.

recursive: count 41, modulo 32459
iterative: count 15345, modulo 37925
matrix: count 122572, modulo 3525

The recursive one runs the slowest. The running time $T(n) = T(n-1)+T(n-2)+O(1)$, which is close to $2^{n/2}$. The iterative algorithm runs faster, because it stores the result of the previous calculation into an array. The running time $T(n) = O(n)$. The matrix one runs the fastest among the three methods. The matrix algorithm stores previous calculation and has multiplications between small digits (compared to additions in recursive and iterative), thus it takes fewer primitive operations.

2 Is A O(B)

A	B	O	o	Ω	ω	Θ
$\log n$	$\log n^2$	✓		✓		✓
$\sqrt[3]{n}$	$(\log n)^6$			✓	✓	
$n^2 2^n$	3^n	✓	✓			
$\frac{n^2}{\log n}$	$n \log n^2$			✓	✓	
$(\log n)^{\log n}$	$\frac{n}{\log n}$			✓	✓	
$100n + \log n$	$(\log n)^3 + n$	✓		✓		✓

3 Find functions with proofs

1. Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
if $f_1(x) = x$,
$\lim_{x\to\infty} \frac{f_1(x)}{f_1(x)} = \frac{f_1'(2x)}{f_1'(x)} = 2$
$\therefore f_1(2n)$ is $O(f_1(n))$

2. Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
if $f_2(x) = 2^x$,
$\lim_{x\to\infty} \frac{f_2(2^2x)}{f_2(x)} = \frac{f_2'(2^2x)}{f_2'(x)} = \frac{2\cdot 2^{2x}\cdot ln2}{2^x\cdot ln2} = 2^{x+1} = \infty$
$\therefore f_2(2n)$ is not $O(f_2(n))$

3. Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.
if $f(n) = O(g(n))$, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{f'(n)}{g'(n)} = c$ (c is a constant)
similarly, $\lim_{n\to\infty} \frac{g(n)}{h(n)} = \frac{g'(n)}{h'(n)} = d$ (d is a constant)
$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \frac{f'(n)}{h'(n)} = \frac{f'(n)}{g'(n)} * \frac{g'(n)}{h'(n)} = c * d$
$\because$ c, d are constants
$\therefore c * d$ is a constant
$\therefore f(n)$ is $O(h(n))$

4. Give a proof or a counterexample: if f is not $O(g)$, then g is $O(f)$.
if f is not $O(g)$
$\frac{f'}{g'} = \infty$
$\therefore \frac{g'}{f'} = \lim_{n\to\infty} \frac{g'}{f'} = 0$
$\therefore g$ is $O(f)$

5. Give a proof or a counterexample: if f is $o(g)$, then f is $O(g)$.
if f is $o(g)$
for all $n \geq N$, $f(n) < c * g(n)$ for all c
$\therefore$ there must exist positive constants c and N such that
for all $n \geq N$, $f(n) \leq c * g(n)$
thus, f is $O(g)$

4 Stoogesort

Proof
Suppose the input array is $A[i...i+3k]$ with length $3k-i+1$, i,k are integers.
The desired output is a sorted array A.

In the first phase, the subarray $A[i...i+2k-1]$ is sorted, which indicates all the elements in $A[i+k...i+2k-1]$ are no smaller than elements in $A[i...i+k-1]$, we write as:
(1) $A[i+k...i+2k-1] \geq A[i...i+k-1]$
thus, there is at least k elements that $A[i+k...i+3k] \geq A[i...i+k-1]$

Continue to the second phase, the subarray $A[i+k...i+3k]$ is sorted, which implies:
(2.1) all the elements in $A[i+2k...i+3k]$ are no smaller than any elements in $A[i+k...i+2k-1]$
(2.2) $A[i+2k...i+3k] \geq A[i+k...i+2k-1]$
$\therefore$ there is at least $k+1$ elements that $A[i+2k...i+3k] \geq A[i...i+k-1]$ according to (1)
$\therefore A[i+2k...i+3k] \geq A[i...i+2k-1]$

In the third phase, the subarray $A[i...i+2k-1]$ is sorted. Finally the array A is sorted.

Running time
$T(n) = 3 \cdot T(2n/3) + O(1)$
$a = 3, b = \frac{3}{2}, k = 0$
$\therefore a = 3 > \frac{3}{2}^0 = b^k$
$T(n) = O(n^{\log_{3/2} 3})$

5 Asymptotic bounds

1. $T(n) = 4T(n/2) + n^3$
$a = 4, b = 2, c = 1, k = 3$
$\because a = 4 < 2^3 = b^k$
$\therefore T(n) = O(n^k) = O(n^3)$

2. $T(n) = 17T(n/4) + n^2$
$a = 17, b = 4, c = 1, k = 2$
$\because a = 17 > 4^2 = b^k$
$\therefore T(n) = O(n^{log_b a}) = O(n^{log_4 17})$

3. $T(n) = 9T(n/3) + n^2$
$a = 9, b = 3, c = 1, k = 2$
$\because a = 9 = 3^2 = b^k$
$\therefore T(n) = O(n^k \cdot log(n)) = O(n^2 \cdot log(n))$

4. $T(n) = T(\sqrt{n}) + 1$
$T(n) = T(n^{\frac{1}{2^k}}) + k$
Suppose $n = 2^{2^k}$
$k = log(log(n))$
$T(n) = O(1) + k$
$\therefore T(n) = O(log(log(n)))$

6 Barleymow

The outer loop runs n times. The nested loop runs $(n+1)/2$ times in average for each outer loop. (The nested loop goes through the *container* from 1 to n, thus in average the nested loop has $(n+1)/2$ elements) In this case, the running time of the algorithm $T(n) = O(n \cdot (n+1)/2) = O(n^2)$.