ANLY-550 Homework 1

Yi Li, Feb. 1

1. (a) The recursive method is the slowest method.

My results:

40 th fib is: 102334155

time: 60.68246603012085

40 th fib % 65536 is 32459

time: 62.39828896522522

After one minute, we got the 40th Fibonacci number. When modulo 65536, we still got the 40th Fibonacci number, and there is little difference between their running time.

(b) The iterative method is much faster.

My results:

801079 th fib

time 60.00001502037048

132462546 th fib % 65536 is 6589

time 60.000000953674316

After one minute, we get the 801,079th Fibonacci number. When modulo 65536, we got the 132,462,546th Fibonacci number, which is about 160 times bigger than the former number. This shows that avoiding large integers saves a lot of time.

(c) The matrix method is the fastest method.

My results:

1686371 th fib

time: 60.00003218650818

41369542 th fib % 65536 is 59970

time: 60.00000190734863

After one minute, we got the 1,686,371st Fibonacci number approximately. When modulo 65536, we got the 41,369,542nd Fibonacci number, which is about 20 times bigger than the former number. Again, this shows that multiplying large integers takes much more running time.

2.

A	B	O	o	Ω	ω	Θ
$\log n$	$\log(n^2)$	*yes*	*no*	*yes*	*no*	*yes*
$\log(n!)$	$\log(n^n)$	*yes*	*yes*	*no*	*no*	*no*
$\sqrt[3]{n}$	$(\log n)^6$	*yes*	*yes*	*no*	*no*	*no*
$n^2 2^n$	3^n	*no*	*no*	*yes*	*yes*	*no*
$(n^2)!$	n^n	*no*	*no*	*yes*	*yes*	*no*
$\frac{n^2}{\log n}$	$n\log(n^2)$	*no*	*no*	*yes*	*yes*	*no*
$(\log n)^{\log n}$	$\frac{n}{\log(n)}$	*no*	*no*	*yes*	*yes*	*no*
$100n + \log n$	$(\log n)^3 + n$	*yes*	*no*	*yes*	*no*	*yes*

3.1 Assume that $f_1(n) = n$, then $f_1(2n) = 2n$, so $f_1(2n)$ is $O(n)$, that is, $f_1(2n)$ is $O(f_1(n))$.

3.2 Assume that $f_2(n) = 2^n$, then $f_2(2n) = 2^{2n}$. Obviously, $f_2(2n)$ is not $O(f_2(n))$.

3.3 If $f(n)$ is $O(g(n))$, then there are constants c_1 and N_1, such that $f(n) <= c_1 g(n)$, for all $n >= N_1$. If $g(n)$ is $O(h(n))$, then there are constants c_2 and N_2, such that $g(n) <= c_2 h(n)$, for all $n >= N_2$. So $f(n) <= c_1 g(n) <= c_1 c_2 h(n)$, for all $n >= max\{N_1, N_2\}$, that is, $f(n)$ is $O(h(n))$.

3.4 Counterexample: Assume that $f(n) = \sin n$, $g(n) = \cos n$, then f is not $O(g)$, and g is not $O(f)$.

3.5 Proof: If f is $o(g)$, then $\lim_{n\to\infty} f(n)/g(n) = 0$, which means that there is a constant N, such that $f(n) < g(n)$ for all $n >= N$. So f is $O(g)$.

4. The algorithm is defined as follows:

If the value at the end is smaller than the value at the start, swap them. If there are 3 or more elements in the list, then: Stooge sort the initial $\lceil 2/3n \rceil$ of the list; Stooge sort the final $\lceil 2/3n \rceil$ of the list; Stooge sort the initial $\lceil 2/3n \rceil$ of the list again.

The algorithm should sort the initial $\lceil 2/3n \rceil$ of the list (the first part) and the finial $\lceil 2/3n \rceil$ of the list (the second part), so that the first part and the second part are overlapped. Otherwise, giving a list of length = 4, the first part ($\lfloor 2/3 * 4 \rfloor = 2$ values) and the last part (also 2 values) are not overlapped, thus the sort can fail.

Proof: We write an list as $A[i..j]$, and set $k = \lceil (j-i)/3 \rceil$. It is true when the length of the list A is 1 or 2. Now we assume that this algorithm correctly sorts all lists whose length is shorter than $A[i..j]$, and let us show it works for list whose length is $A[i..j]$.

After sorting the initial $\lceil 2/3n \rceil$ of the list, we know that every element of $A[(i+k)..(j-k)]$ is no smaller than every element of $A[i..(i+k-1)]$. We write this as $A[(i+k)..(j-k)] >= A[i..(i+k-1)]$. And there are at least $length(A[(i+k)..(j-k)]) = j - i - 2k + 1$ elements no smaller than every element of $A[i..(i+k-1)]$.

Then after sorting the final $\lceil 2/3n \rceil$ of the list, we know that $A[(j-k+1)..j] >= A[(i+k)..(j-k)]$. And $A[(j-k+1)..j]$ is sorted. Since $A[(i+k)..j]$ has at least $(j - i - 2k + 1)$ elements no smaller than each element of $A[i..(i+k-1)]$, and $length(A[(j-k+1)..j]) <= j - i - 2k + 1$, we conclude that $A[(j-k+1)..j] >= A[i..(i+k-1)]$. So $A[(j-k+1)..j] >= A[i..(j-k)]$. Finally, $A[i..(j-k)]$ is sorted, so the whole list is sorted.

As $T(n) = 3T(2/3n) + n$, so the running time of this algorithm is $O(n^{\log_{1.5} 3})$, which is about $O(n^{2.71})$.

5.1 $T(n) = 1 + 3n(n-1)/2$.

Proof: When $n = 1$, $T(1) = 1 + 0 = 1$, the statement holds. Assume that $T(n) = 1 + 3n(n-1)/2$, then $T(n+1) = T(n) + 3(n+1) - 3 = 1 + 3n(n-1)/2 + 3n = 1 + 3(n+1)n/2$.

5.2 $T(n) = \sum_{k=1}^{n} 2^{n-k}(2k-1) = 2^{n+1} + 2^n - 2n - 3$.

Proof: When $n = 1$, $T(1) = 4 + 2 - 2 - 3 = 1$, the statement holds. Assume that $T(n) = 2^{n+1} + 2^n - 2n - 3$, then $T(n+1) = 2T(n) + 2(n+1) - 1 = 2(2^{n+1} + 2^n - 2n - 3) + 2n + 1 = 2^{n+2} + 2^{n+1} - 2(n+1) - 3$.

6. According to the master theory:

6.1 $a = 4, b = 2, k = 3$, so $a < b^k$, the asymptotic bound is $O(n^3)$.

6.2 $a = 17, b = 4, k = 2$, so $a > b^k$, the asymptotic bound is $O(n^{\log_4 17})$.

6.3 $a = 9, b = 3, k = 2$, so $a = b^k$, the asymptotic bound is $O(n^2 \log n)$.

6.4 Assume $T(2) = 1$, then $T(2^2) = T(2) + 1 = 2, T(2^{2^2}) = T(2^2) + 1 = 3, T(2^{2^3}) = T(2^{2^2}) + 1 = 4, ..., T(2^{2^n}) = T(2^{2^{n-1}}) + 1 = n + 1$. So $T(n) = \log_2(\log_2 n) + 1$, and the asymptotic bound is $O(\log(\log n))$.

7. When n is a power of 2, $T(n) = n \log_2 n - n + 1$. When n is not a power of 2, $T(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$.

Proof: For n is a power of 2. When $n = 1$, $T(1) = 0 - 1 + 1 = 0$, the statement holds. Assume that $T(n) = n \log_2 n - n + 1$, then $T(2n) = 2T(n) + 2n - 1 = 2(n \log_2 n - n + 1) + 2n - 1 = 2n \log_2(2n) - 2n + 1$.

For n is not a power of 2. When $n = 1$, $T(1) = 0 - 1 + 1 = 0$, the statement holds. Assume that $T(n) = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$, then $T(2n) = T(n) + T(n) + 2n - 1 = 2(n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1) + 2n - 1 = 2n\lceil \log_2(2n) \rceil - 2^{\lceil \log_2(2n) \rceil} + 1$.