CSCI 3110 Assignment 1 Solutions

October 10, 2012

1. In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case f $= \Theta(g)$).

	$f(n)$	$g(n)$	Relation	Reason
(c)	$100n + \log n$	$n + (\log n)^2$	$f = \Theta(g)$	Both are $\Theta(n)$
(d)	$n \log n$	$10n \log 10n$	$f = \Theta(g)$	Both are $\Theta(n \log n)$
(e)	$\log 2n$	$\log 3n$	$f = \Theta(g)$	Both are $\Theta(\log n)$
(f)	$10 \log n$	$\log(n^2)$	$f = \Theta(g)$	Both are $\Theta(\log n)$
(g)	$n^{1.01}$	$n \log^2 n$	$f = \Omega(g)$	$n^{0.01} = \Omega(\log^2 n)$
(h)	$n^2/\log n$	$n(\log n)^2$	$f = \Omega(g)$	$n = \Omega((\log n)^3)$
(i)	$n^{0.1}$	$(\log n)^{10}$	$f = \Omega(g)$	Any polynomial $= \Omega()$ any log
(j)	$(\log n)^{\log n}$	$n/\log n$	$f = \Omega(g)$	$2^{\log_2 n} = n$ so $(\log n)^{\log n + 1} = \Omega(n)$
(k)	$\sqrt{n}$	$(\log n)^3$	$f = \Omega(g)$	Any polynomial $= \Omega()$ any log
(l)	$n^{1/2}$	$5^{\log_2 n}$	$f = O(g)$	$5^{\log_2 n} = n^{\log_2 5}$ and $1/2 < \log_2 5$
(m)	$n2^n$	3^n	$f = O(g)$	$n = O((3/2)^n)$
(n)	2^n	2^{n+1}	$f = \Theta(g)$	$n = \Theta(n+1)$
(o)	$n!$	2^n	$f = \Omega(g)$	$n! > 2^n$ for all $n \geq 4$

2. Show that, if c is a positive real number, then $g(n) = 1 + c + c^2 + \ldots + c^n$ is:

   (a) $\Theta(1)$ if $c < 1$.

   We take the limit
$$\lim_{n\to\infty} \frac{g(n)}{1} = \lim_{n\to\infty} \frac{\sum_{i=0}^{n} c^i}{1} = \lim_{n\to\infty} \sum_{i=0}^{n} c^i = \frac{1}{1-c}$$
   for $c < 1$. This limit is constant, so $g = \Theta(1)$.

   (b) $\Theta(n)$ if $c = 1$.

   We expand
$$g(n) = 1 + c + c^2 + \ldots + c^n = \sum_{i=0}^{n} c^i = n + 1 = \Theta(n).$$

   (c) $\Theta(c^n)$ if $c > 1$.

   Each term c^i is dominated by c^n for $i < n$, so $g(n) = \Theta(c^n)$.

   The moral: in big-Θ terms, the sum of a geometric series is simply the first term if the series is strictly decreasing, the last term if the series is strictly increasing, or the number of terms if the series is unchanging.

3. The Fibonacci numbers $F_0, F_1, F_2, \ldots,$ are defined by the rule
$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}.$$

   In this problem we will confirm that this sequence grows exponentially fast and obtain some bounds on its growth.

(a) Use induction to prove that $F_n \geq 2^{0.5n}$ for $n \geq 6$.

We first establish the base cases $n = 6$ and $n = 7$.
$n = 6$ $F_6 = 8 \geq 2^{0.5 \cdot 3} = 8$.
$n = 7$ $F_7 = 13 \geq 2^{0.5 \cdot 3} \approx 11.3$.
Now, assume that the claim holds for all $n < k + 2$. Then

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &\geq 2^{0.5k} + 2^{0.5k+0.5} = \frac{(2^{0.5}+1)}{2} 2^{0.5k+1} = \frac{(2^{0.5}+1)}{2} 2^{0.5(k+2)} \\ &\geq 2^{0.5(k+2)}. \end{aligned}$$

Therefore, by induction, $F_n \geq 2^{0.5n}$ for $n \geq 6$.

(b) Find a constant $c < 1$ such that $F_n \leq 2^{cn}$ for all $n \geq 0$. Show that your answer is correct.

We use induction to prove that $c = 0.9$ works. We first establish the base cases $n = 0$ and $n = 1$.
$n = 0$ $F_0 = 0 \leq 2^{0.9 \cdot 0} = 1$.
$n = 1$ $F_1 = 1 \leq 2^{0.9 \cdot 1} \approx 1.86$.
Now, assume that the claim holds for all $n < k + 2$. Then

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &\leq 2^{0.9k} + 2^{0.9k+0.9} = 2^{0.9k}(1 + 2^{0.9}) = \frac{(1+2^{0.9})}{2^{1.8}} 2^{0.9(k+2)} \approx 0.82(2^{0.9(k+2)}) \\ &\leq 2^{0.9(k+2)}. \end{aligned}$$

Therefore, by induction, $F_n \leq 2^{0.9n}$ for $n \leq 6$.

(c) What is the largest c you can find for which $F_n = \Omega(2^{cn})$?

Let $b = 2^c$. Then we consider the relation $F_{n+2} = F_n + F_{n+1}$ and get $b^{n+2} = b^n + b^{n+1}$. This can be rearranged as $b^{n+2} - b^n - b^{n+1} = 0$. Solving for the roots of this equation gives $b = \frac{1 \pm 5^{0.5}}{2}$. Since b cannot be negative, $c = \log_2(b) = \log_2(\frac{1+5^{0.5}}{2}) \approx 0.69$.

4. (a) Show that two 2 x 2 matrices can be multiplied using 4 additions and 8 multiplications.

The formula to multiply two 2 x 2 matrices is:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix},$$

Which requires 4 additions and 8 multiplications.

(b) Show that $O(\log n)$ matrix multiplications suffice for computing X^n.

X^n can be computed with the following recurrence relation. This relation requires $O(\log n)$ matrix multiplications to compute X^n by making one recursive call to determine $X^{\lfloor n/2 \rfloor}$ and then applying a constant number of matrix multiplications.

$$\begin{aligned} X^n &= I \text{ if } n = 0 \\ &= X \text{ if } n = 1 \\ &= X^{n/2} \cdot X^{n/2} \text{ if } n \text{ is even} \\ &= X \cdot X^{\lfloor n/2 \rfloor} \cdot X^{\lfloor n/2 \rfloor} \text{ if } n \text{ is odd} \end{aligned}$$

(c) Show that all intermediate results of fib3 are $O(n)$ bits long.

We can use induction to prove the claim. Let

$$F = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Our base cases compute F^0 and F^1, which are $O(1)$ bits long. Now, assume that the claim holds for $1 \leq n < k$. If k is odd, then $F^k = F \cdot F^{\lfloor k/2 \rfloor} \cdot F^{\lfloor k/2 \rfloor}$. By the inductive hypothesis, computing this requires multiplying $O(1)$ bit numbers with $O(k/2)$ bit numbers and then with $O(k/2)$ bit numbers, resulting in $O(k)$ bit numbers. If k is even, then $F^k = F^{\lfloor k/2 \rfloor} \cdot F^{\lfloor k/2 \rfloor}$, which similarly has $O(k)$ bit numbers. Therefore, the claim holds by induction.

(d) Let $M(n)$ be the running time of an algorithm for multiplying n-bit numbers, and assume that $M(n) = O(n^2)$. Prove that the running time of fib3 is $O(M(n) \log n)$.

fib3 requires $O(\log n)$ multiplications, which each run in $O(M(n))$ time. Each of the $O(\log n)$ steps of the algorithm does $O(1)$ other work, so the algorithm requires $O(M(n) \log n)$ time.

(e) Can you prove that the running time of fib3 is $O(M(n))$?
We can prove this using induction. Our base cases are once again F^0 and F^1, which clearly require $O(M(n))$ time. Now, assume that the claim holds for $1 \leq n < k$. If k is odd, then $F^k = F \cdot F^{\lfloor k/2 \rfloor} \cdot F^{\lfloor k/2 \rfloor}$. By the inductive hypothesis, determining $F^{\lfloor k/2 \rfloor}$ requires $O(M(k/2) = O(M(k))$ time. Multiplying these arrays requires $O(M(k))$ time as well, so F^k can be found in $O(M(k))$ time. Similarly, F^k can be computed in $O(M(k)$ time if k is even. Therefore, the claim holds by induction.

5. Linear Search
PRE: An array
POST: An array index of key in A or -1 if not found
LINEARSEARCH(A, fst, lst, key) [1] $lst < fst$ $index = -1$ $key == A[fst]$ $index = fst$
LINEARSEARCH($A, fst + 1, lst, key$)

Proof of Correctness
Inductive hypothesis: The proc `LinearSearch` correctly searches for the key for all arrays of size $lst - fst + 1 < n$, $\forall n > 0$.
Base Case: Let $n = 1$(array size) Then, if key matches the only element, algorithm will correctly return the only index $fst = 1$. If the key does not match, a recursive call is made with $fst = 2$, $lst = 1$, returning a -1, performing the search correctly *(This step could also be done taking $n = 0 = lst - fst + 1 \rightarrow lst < fst$ and the proc.* correctly *returns -1 on the empty array.)*
Inductive Step: For $n = fst - lst + 1$ The proc. checks if the first element matches the key. If it matches, the index of the first element is returned. If not, it makes a recursive call on an array of size $n - 1 < n$, which from the inductive assumption, works correctly. Hence the proc. works correctly.

Running time: Base Case check: $O(1)$; Check of 1st element with key $O(1)$; Recursive call: $T(n-1)$
Hence we have $T(n) = T(n-1) + c$ ($\exists c > 0$). Unrolling this is simple: $T(n) = T(n-2) + 2c$ *etc etc* Proc terminates when $T(n) = T(0) + n \cdot c$, giving $T(n) \in O(n)$.

6. Trinary Search

(a) **Base Case:** $n = 0 = lst - fst + 1$, i.e, $lst < fst$ and the proc. returns -1.
Ind. Hyp.: TRINSEARCH works correctly for all i/p of size $n = k = lst - fst + 1$, $\forall k \geq 0$
Assume when $n = k + 1$, since $n > 0$, proc. jumps to line 4 and calculates *thrd* and *twrd*. If $key == A[thrd]$, then proc. return **index** and $fst \leq thrd \leq lst$, the proc. terminates correctly.
Else if $key < A[thrd]$, since $thrd - 1 - fst + 1 = thrd - fst < k + 1$, TRINSEARCH can return correct index based on **Ind. Hyp.**. **Else** if $key == A[twrd]$, then proc. return **index** and $fst \leq twrd \leq lst$, the proc. terminates correctly. **Else** since $twrd - 1 - thrd - 1 + 1 = twrd - thrd - 1 < k + 1$ and $lst - twrd - 1 + 1 = lst - twrd < k + 1$, TRINSEARCH can return correct index based on **Ind. Hyp.**.

(b) Since proc. splits the arrays into 3 slices in each cycle, $O(\log_3(n))$ recursions are needed. Although $\log_3(n) < \log_2(n)$, it is in the equivalent class $O(\log(n))$ as $O(\log_2(n))$. TRINSEARCH can be thought of more efficient in terms of actual running time but the same efficient as BINARY SEARCH in terms of Big-O notation.