CS3230 : Design and Analysis of Algorithms (Spring 2015)

Tutorial Set #2

[SOLUTION SKETCHES with Suggestions]

[DO NOT give to future students: Let them have a chance to learn.]

Routine Practice Problems *-- do not turn these in -- but make sure you know how to do them.*

R2. [One more] ([CLRS] Problem 3-1, page 61) Polynomial Rule

Solution: Proving this by definition is do-able (can you do it?), but tedious. Use the sum theorem or the limit theorem for a simpler proof.

R3. [Fun with Estimation using Bounding of Integrals]

(a) Reproduce by yourself, the proof for estimation of $H(n) = (1/1 + 1/2 + 1/3 + \ldots + 1/n)$, using the method of integration. Namely, reproduce the proof that:

$$\ln(n+1) \le \sum_{k=1}^{n} \frac{1}{k} \le (\ln n) + 1$$

Answer: This is found in [CLRS], pp. 1154-1156. Another method is in pp. 1153-1154. Also found in lecture notes. Do the two inequalities separately.

Here, I give you the thinking process:
First ask if function is monotonic increasing / decreasing.
(else, no go with estimation by integration!)
Then do inequalities separately. Be careful with *end points* of the summation/integral.
Then just be careful with the calculus of it.

Friendly Reminder: Make sure you know how to do this YOURSELF.

(b) DIY with the same thinking process. And method.

R4. [Fun with TELESCOPING]

(b) Solve this recurrence using telescoping method: (can you "see" the *telescope*?)

$A_n = A_{n-1} + (n-1)$ for $n>1$, $A_1 = 0$

Answer: A little rewriting of the recurrence (make it more convenient)

$A_n = (n-1) + A_{n-1}$ Now, the telescoping starts…
$= (n-1) + (n-2) + A_{n-2}$ (substitute $A_{n-1} = (n-2) + A_{n-2}$)
$= (n-1) + (n-2) + (n-3) + A_{n-3}$ (substitute $A_{n-2} = (n-3) + A_{n-3}$)
$= (n-1) + (n-2) + (n-3) + (n-4) + A_{n-4}$ (substitute $A_{n-3} = (n-4) + A_{n-4}$)
. . . . . .
$= (n-1) + (n-2) + (n-3) + \ldots + (2) + A_2$ (substitute $A_3 = (2) + A_2$)
$= (n-1) + (n-2) + (n-3) + \ldots + (2) + (1) + A_1$ (substitute $A_2 = (1) + A_1$)
$= (n-1) + (n-2) + (n-3) + \ldots + (2) + (1) + 0$ (substitute $A_1 = 0$)

Complete the rest of the steps yourself. You MUST already know your A, B, C. YES?

Question: Can you see the telescope?

D1. [Two very useful theorems that you can use, repeatedly.]
Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove that

Lemma 1: If $f(n) = O(F(n))$, $g(n) = O(G(n))$, then $f(n) + g(n) = O(\max(F(n), G(n)))$.

Lemma 2: If $f(n) = O(g(n))$, then $\lg(f(n)) = O(\lg(g(n)))$,
where $\lg(g(n)) \geq 1$ and $f(n) > 1$ for all sufficiently large n.

ANSWER:

Before we prove it, let's see what it means first and how to use it.

First, we try one example.
Let $f(n) = 173n^2 + 61n + 151$, and $F(n) = n^2$, then clearly, $f(n) = O(F(n))$
Let $g(n) = 963n + 106 \lg n + 96$, and $G(n) = n$, then clearly, $g(n) = O(G(n))$
Then *Lemma 1 says*,
$173n^2 + 1024n + 106 \lg n + 247 = f(n) + g(n) = O(\max(n^2, n)) = O(n^2)$,

It also says, that when we analysis a program/software, and there two different parts
Part A has running time $T(n)$, and
Part B has running time $S(n)$,
then, total running time is just the *max of the two* when expressed in asymp. notations.

For example, in algorithm Array-Sum (L02a-Slide-7), initialization and post-processing is $O(1)$, while main loop is $O(n)$, so running time is $O(n)$. So, Lemma says, total time is $O(n)$.

In Selection-Sort (L02a-S45), we analyze (a) the running time of all the Find-Max(A, j) calls separately from (b) the rest of the code which clearly runs in $O(n)$ time. The running time for Part (a) is $O(n^2)$. So, Lemma 1 tells the total running time is $O(n^2)$.

So, Lemma 1 is really a *short cut* for Analysis of Algorithms. *Cool*, right?

Note: Do you see what Lemma 1 is really "*saying*"?
Please continue to learn the "language" of Lemmas and Theorems?

Now, to prove Lemma 1:
First, what are we given: $f(n) = O(F(n))$, $g(n) = O(G(n))$, (1)
And what do we need to prove: $f(n) + g(n) = O(\max(F(n), G(n)))$. (2)

From (1), we know that there is are constants C_1 and n_0, and D_1 and m_0, such that
$f(n) \leq C_1 \cdot F(n)$ [5] whenever $n \geq n_0$,
$g(n) \leq D_1 \cdot G(n)$ [6] whenever $n \geq m_0$,

To make things simpler to understand, let $H(n) = \max\{F(n), G(n)\}$.
Try to visualize the function $H(n)$.

We need to prove that we can find some constant E_1 and p_0, such that
$f(n) + g(n) \leq E_1 H(n)$ whenever $n \geq p_0$,

Then for any $n \geq p_0 = \max\{n_0, m_0\}$, [5] and [6] are both true.
For this value of n, we have two cases:
Case 1: if $F(n) \leq G(n)$, then $H(n) = G(n)$
$f(n) + g(n) \leq C_1 \cdot F(n) + D_1 \cdot G(n) \leq (C_1 + D_1) \cdot G(n) = (C_1 + D_1) \cdot H(n)$
Case 2: if $G(n) \leq F(n)$, then $H(n) = F(n)$
$f(n) + g(n) \leq C_1 \cdot F(n) + D_1 \cdot G(n) \leq (C_1 + D_1) \cdot F(n) = (C_1 + D_1) \cdot H(n)$

Thus, choose $E_1 = (C_1 + D_1)$, and $p_0 = \max\{n_0, m_0\}$, we have proven (2).

Partial Proof of Lemma 2:

First, what are we given: $f(n) = O(g(n))$ (3)

And what do we need to prove: $\lg(f(n)) = O(\lg(g(n)))$ (4)

Note: We are also given "where $\lg(g(n)) \geq 1$ and $f(n) > 1$ for all sufficiently large n" all that is to make the functions behave nicely (not *naughtily*), i.e. non-negative.

Make use of this mathematical fact about strictly increasing functions:

Since $f(n) = \lg n$ is a strictly increasing function, we know that

If $f(n) < g(n)$, then $\lg(f(n)) < \lg(g(n))$,

DIY the rest. (Use Proof of Lemma 1 as a guide.)

D2. Master Theorem.

(a) $T(n) = 2T(n/2) + 20$.

$f(n) = 20 = \Theta(n^0)$, a=2, b=2. So $n^{\log_b a} = n^{\log_2 2} = n^1 = n$

Since $f(n)=O(n^{1-\varepsilon})$, for ε=0.5. This is **Case 1**. Hence, $T(n)= \Theta(n)$

(b) $T(n) = 2T(n/2) + (3n + 4\lg n)$.

$f(n) = (3n + 4\lg n) = \Theta(n)$, a=2, b=2. So $n^{\log_b a} = n^{\log_2 2} = n^1 = n$.

Since $f(n) = \Theta(n) = \Theta(n^{\log_b a})$. This is **Case 2**, with (k=0). Hence, $T(n)= \Theta(n \lg n)$

(c) $T(n) = 2T(n/2) + 3n^2$

$f(n) = 3n^2 = \Theta(n^2)$, a=2, b=2. So. $n^{\log_b a} = n^{\log_2 2} = n^1 = n$.

So $f(n)= \Omega(n^{1+\varepsilon})$ for ε=0.5 (for example).

Next check regularity condition, i.e. $a\, f(n/b) \leq c\, f(n)$, for some constant $c < 1$.

$3(n/2)^2 = (\frac{1}{4})\, 3n^2 \leq c\, f(n)$, for $c = 1/4$. (the regularity condition is satisfied)

This is **Case 3**. Hence, $T(n) = \Theta(n^2)$

(d) $T(n) = 4T(n/2) + 3n^2$.

$f(n) = 3n^2 = \Theta(n^2)$, a=4, b=2. So $n^{\log_b a} = n^{\log_2 4} = n^2$.

Since $f(n)= n^2 = \Theta(n^{\log_b a})$. This is **Case 2** with (k=0). Hence, $T(n)= \Theta(n^2 \lg n)$

(e) $T(n) = 8T(n/2) + 3n^2$.

$f(n) = 3n^2 = \Theta(n^2)$, a=8, b=2. So $n^{\log_b a} = n^{\log_2 8} = n^3$.

Since $f(n)= n^2 = O(n^{3-\varepsilon})$, for ε=0.5. This is **Case 1**. Hence, $T(n)= \Theta(n^3)$

(f) $T(n) = 2T(n/4) + \sqrt{n}$.

$f(n) = \sqrt{n} = \Theta(n^{0.5})$, a=2, b=4. So $n^{\log_b a} = n^{\log_4 2} = n^{0.5}$.

Since $f(n) = \sqrt{n} = \Theta(n^{\log_b a})$. This is **Case 2** & ($k$=0). Hence, $T(n)= \Theta(\sqrt{n}\,(\lg n))$

Note: After you have finished solving all of them. Now, just look at the solutions and try to make some critical observations in the relationship between a, b, and $f(n)$. Learn to develop some intuition for the following, given a and b, how big can $f(n)$ be and still be in Case 1? When does it go to Case 2? When does it go to Case 3?.

D3. Karatsuba Algorithm

Use the Karatsuba algorithm to compute the product of 2 integer "strings" (with n=4)
6161 x 3608. Showing your working.

Solution:

$x = 6161 = 10^2*61 + 61$ $x_1 = 61,\ x_0 = 61$
$y = 3608 = 10^2*36 + 08$ $y_1 = 36,\ y_0 = 08$

Then

$x_1 + x_0 = 61 + 61 = 122$ $x_0*y_0 = 61*08 = 488$ (multiply)
$y_1 + y_0 = 36 + 08 = 44$ $x_1*y_1 = 61*36 = 2196$ (multiply)
$(x_1+x_0)*(y_1+y_0) = 122*44 = 5368$ (multiply)

So,

$x*y = 10^4\, x_1*y_1 + 10^2\, ((x_1+x_0)*(y_1+y_0) - x_0*y_0 - x_1*y_1) + x_0*y_0$ (add/subtr only)
$= 10^4(2196) + 10^2(5368 - 488 - 2196) + 488$ (add/subtr only)
$= 21960000 + 268400 + 488$
$= 22228888$

If we do if by long multiplication:

```
      6161
      3608
==========
     49288
     0000
   36966
  18483
==========
 22228888
==========
```

D4. [Finding Customer Details for ColdCore (pun on Cold Call)]

You have just started an internship in a company called ColdCore. On the second day, you received a random set of m telephone numbers, stored in an array $Q[1..m]$, and a database of n telephone numbers, stored in a sorted array $S[1..n]$ (namely, $S[k] < S[k+1]$ for all k). Your boss wants you to implement the following high-level algorithm:

```
Find-Customer-Details(S, Q)
  for k:=1 to m do {
    determine the position p, such that S[p] = Q[k] using Binary-Search;
        (if Q[k] is not in S[1..n], then set p = –1)
    if (p <> –1) then Print p.    (* you can use p as pointer to get other info  *)
  }
```

(a) What is the running time of the above algorithm? **Solution:** $\Theta(m\ (\lg n))$

(b) Can you give the AA-pattern (algorithm analysis) pattern of the above algorithm.
Solution: Sum-of-Log Pattern.

(c) After working on the problem for one day using the algorithm by your boss, you have a good idea. You request your boss to give the array $Q[1..m]$ as a sorted array. Supposing your boss does that, can you get a *faster* algorithm? What is the running time?

Solution: It depends on the value of m.
If $(m \geq n)$, we can perform a merge, which takes $O(m)$ time.
If $(n > m \geq n/(\log n))$, we can perform a merge, which takes $O(n)$ time.
If $(m < n/(\log n))$, we use the algorithm in **(a)**, which takes $O(m \log n)$ time.