CS3230 : Design and Analysis of Algorithms (Fall 2014)

Tutorial Set #2

[For discussion during Week 4]

S-Problem Due: Friday, 29-Aug, before noon.

OUT: 26-Aug-2014 **Tutorials:** Tue & Wed, 02-03 Sep 2014

IMPORTANT: Read "Remarks about Homework" – also applies to tutorials.

Submit solutions to S-Problem(s) by deadline given above.
Prepare your answers to all the D-Problems in every tutorial set.

When preparing to present your answers,
- Think of a CLEAR EXPLANATION
- Illustrate with a good worked example;
- Describe the main ideas,
- Can you sketch why the solution works;
- Give analysis of running time, if appropriate
- Can you think of other (perhaps simpler) solutions?

Helpful Hints Series: About Theorems and Proofs:

The trick to learning about what a theorem really says, how to prove it, what it really means, and *finally,* and *most importantly* of all, how to apply the theorem to solve problem – is not how *fast* you go through it. It lies in slowly reading it, really understanding the minute details, knowing all the steps, the *key properties* or *invariance*. And asking questions like so "*where can I use this*"?

When proving something, first make sure you know *exactly what it is that you must prove*. Write it down in English, and write in down *again* in mathematics. If you can't do that, then usually you cannot effectively prove it. (Since you do not know where to start and where to end.)

Remember:
- You can **freely quote** standard algorithms and data structures covered in the lectures (including from pre-req. modules), textbook, and homeworks. Explain **any modifications** you make to them, and how they may affect the running time.
- There is no need to copy/re-prove algorithms or theorems already cover already;
- Unless otherwise specified, you are expected to **prove (justify)** your results. All logarithms are base 2, unless otherwise stated.

Examples:
a. Use Quicksort to sort the array X[1..n] in increasing order;
b. Organize the set S as a Max-Heap (array-based);
c. Run a post-order traversal of the tree T, and at each node, the processing of the node is …
d. Run Dijkstra's algorithm for single-source shortest path from vertex w on graph $G=(V, E)$.
e. Do <some-std-alg X>, but with the following modifications: blah, blah, blah….
f. By the Handshaking Lemma, $(d_1 + d_2 + d_2 + \ldots + d_n) = 2*e$
(OK, if you still don't know about Handshaking Lemma, Google it and learn it.)

Of course, it is your responsibility to ensure that the algorithm that you quote ACTUALLY solves your problem.

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

R1. [Say the right thing, don't say nonsense (without knowing it).]
Explain why the statement below is meaningless.
"The running time of algorithm Q is at least $O(n^4)$."

[**Note by HW:** This question is *really* TRICKY. If it does not appear tricky to you, then you probably have not understood the question and do not fully *get* the solution.]

R2. [One more] ([CLRS] Problem 3-1, page 61) Asymptotic behavior of polynomials.

Let $$p(n) = \sum_{i=0}^{d} a_i n^i = (a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n^1 + a_0) \quad \text{where } a_d > 0,$$

be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.

(a) If $k \geq d$, then $p(n) = O(n^k)$.

(b) If $k \leq d$, then $p(n) = \Omega(n^k)$.

(c) If $k = d$, then $p(n) = \Theta(n^k)$.

(d) If $k > d$, then $p(n) = o(n^k)$.

(e) If $k < d$, then $p(n) = \omega(n^k)$.

(**Note:** It is much easier to prove, if you can use the Limit Theorem, right?)

R3. [Fun with Estimation using Bounding of Integrals]
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:

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

S-Problems: (To do and submit by due date given in page 1.)
Solve this S-problem and submit for grading.

S1. Master Theorem.
Use the Master Theorem (whenever possible) to solve for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is a constant for sufficiently small n.

(a) $T(n) = 3T(n/3) + 2(n^{0.5})$.

(b) $T(n) = 3T(n/3) + (3n + 5\lg n)$.

(c) $T(n) = 3T(n/3) + 8n^{1.5} + 13n)$

(d) $T(n) = 7T(n/2) + 21n^2$.

(e) $T(n) = 2T(n/4) + 34\sqrt{n}$.

D-Problems: Solve these D-problems and prepare to discuss them in tutorial class. You may be called upon to present your solution *or your best attempt at a solution*. Your solution presentation does NOT need to be fully correct, given your best attempt. The TA will help clarify and correct any issues or errors.

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.

D2. [Estimation of Summation using Bounding of Integrals]
(a) Use the method of bounding of integrals to get an estimate (similar to T2-R3) for

$$T(n) = \sum_{k=1}^{n} (\ln k) = (\ln(1) + \ln(2) + \ln(3) + \ldots + \ln(n)) = \ln(n!).$$

[Fun with TELESCOPING]
(b) Solve this recurrence using telescoping method: (can you "see" the *telescope*?)

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

What algorithms have running times that are modeled by this recurrence?

D3. Modified from Problem 3-3, pp 61-62 of [CLRS] [Sorting out order of growth rates]
Rank the following functions in *ascending order of growth*; that is, if function $g(n)$ *immediately* follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$g_1(n) = 21(n)^{0.5}$ $g_2(n) = 13n^3$ $g_3(n) = 34n+55$ $g_4(n) = n^2 \lg n$
$g_5(n) = 8$ $g_6(n) = 8(\lg n)^5$

To simplify notations, we write $f(n) << g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the following function n^2, n, $(2013n^2 + n)$, n^3 could be sorted either as $[n << n^2 \equiv (2013n^2 + n) << n^3]$ or as $[n << (2013n^2 + n) \equiv n^2 << n^3]$.

Do not turn in proofs for this problem, but you should do them anyway just for practice.

Advanced Problems – Try these for challenge and fun. There is no deadline for A-problems. *Turn in your attempts DIRECTLY to Prof. Leong. Do not combine it with your HW solutions.*)

M*-problems are more mathematical than D/S-problems, but not necessarily harder.

M1* [Almost identical, but NOT the same.]

Explain the subtle difference between the two lemmas below:

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

Hint: Illustrate when $f(n) = 13n^2 + 34n$, $g(n) = 8n^3 + 21n(\lg n)$.

A3. [How often does the maximum get updated?]

Consider the following simple code for finding the maximum of n numbers $A[1..n]$.

```
1. Algorithm Find-Max (A, n)
2. { Max-sf := –INFTY;
3.   for k:= 1 to n do {
4.     if (A[k] > Max-sf) then
5.       Max-sf := A[k]; endif
6.   } }
```

We already know that the running time is $\Theta(n)$, in fact it takes exactly n comparisons. We are now interested in the question "How many times is *Max-sf* updated in Line 5"?
In the worst-case, the answer is n. In the best-case, *Max-sf* is updated only ONCE.

In general, given a *random* permutation of the n numbers (let's assume the numbers are just $\{1,2,3,\ldots,n\}$), how many times (*on average*) is the variable *Max-sf* updated?

[**HINT:** The answer is ***not*** $n/2$. Can you build a recurrence and solve it?]

A4. [Do you love to break laptops?]

You work in Safe-Laptop, a company that *tests durability* of laptops. You test durability by dropping the laptop on a hard surface (like a cement floor) from n different pre-designated heights, in increasing order. Your task is to find the *maximum safe height* h^*, the maximum height from which you can drop the laptop safely (without breaking it). Safe-Laptop allows you to break *at most* m laptops in the process of finding h^*.

If you can only break 1 laptop ($m = 1$), then your only choice is to do a *linear scan* algorithm, which take $O(n)$ drops in the worst case. If $m = (\lg n)$, then you can find it in $O(\lg n)$ drops using a binary search strategy.

Safe-Laptop has ruled out both these extreme options due to budget and time constraint. They want a testing strategy that is *sub-linear* in the number of drops.

(a) Design an $o(n)$ (small-o) algorithm to find h^* when $m = 2$.

(b) For each $m > 2$, design an algorithm G for finding the h^* using at most m laptops. Let $g_m(n)$ be the number of drops using this algorithm. Then your algorithm G should satisfy that $g_m(n) = o(g_{m-1}(n))$ for each m.