CS3230 : Design and Analysis of Algorithms (Spring 2015)
Tutorial Set #2
[For discussion during Week 4]

OUT: 26-Jan-2015 **Tutorials:** Mon & Fri, 02&06 Feb 2015

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

Prepare your answers to all the D-Problems in every tutorial set.

When presenting your solutions,

- First explain WHAT is problem,
- Think of a CLEAR EXPLANATION,
- Describe the main ideas,
- Illustrate with a good worked example;
- Can you sketch why the solution works;
- Give analysis of running time, if appropriate
- Can you think of other (perhaps simpler) solutions?

In CS3230, you learn to develop high-level abstractions when describing algorithms. Try not to speak in ML/AL (machine/assembly language) or "for (j=0; j<n; j++) do". Instead give names to your sets (of objects or things or data structures), talk about Depth-First Search, Binary Search, traverse the graph, sort the set, use a priority queue, etc. You are no longer in CS1010, CS1020, CS2010 or CS2020. Speak with greater sophistication, and at a higher level of abstraction.

Remember:

- You can **freely quote** standard algorithms and data structures covered in the lectures (and including from pre-requisites) modules, textbook. 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 Q>, but with the following modifications: blah, blah, blah….
f. By the Handshaking Lemma, $(d_1 + d_2 + d_2 + \ldots + d_n) = 2e$
(OK, if you still don't know the 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 time, don't say nonsense (without knowing it).]
Explain why the statement below is meaningless.

"The running time of algorithm Q is at least $O(n^3)$."

[**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 $k \ge d$, then $p(n) = O(n^k)$.

(b) If $k \le d$, 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 $p(n) = \omega(n^k)$.

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:

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

(b) Now, give a similar estimate for $T(n) = (\ln(1) + \ln(2) + \ln(3) + \ldots + \ln(n))$.

R4. [Fun with TELESCOPING]
Solve this recurrence using telescoping method: (can you "see" the *telescope*?)

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

D-Problems:

Solve these D-problems and prepare to discuss them in tutorial class. Your TA will call upon one of you 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 $f(n)=O(F(n))$ and $g(n)=O(G(n))$, 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. 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) $T(n) = 2T(n/2) + 20$.

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

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

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

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

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

D3. Karatsuba Algorithm.

Use the Karatsuba algorithm to compute the product of 2 integer "strings" (with n=4)
6161 x 3608.
Showing your working.
(*Note*: You only need to work *one* level of recursive call, namely, you can assume that the base case for the recursion is n=2, where n is the length of the integer "string". When n=2, you can just call the standard simple integer multiplication.)

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?

(b) Can you give the AA-pattern (algorithm analysis) pattern of the above algorithm.

(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?

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: [Stable Marriage Problem – Special Case] Consider the following preference matrix which embeds both preference list into one matrix. If the cell (i, j) of the matrix contains the entry (x, y), then
x is the j^{th} preference of the boy b_i while
y is the i^{th} preference of the girl G_j.
Hence, the boys' preference lists are read row-wise, while the girls' preference lists are read column-wise.

	G_1	G_2	G_3	. . .	G_n
b_1	$(1, n)$	$(2, n–1)$	$(3, n–2)$	. . .	$(n, 1)$
b_2	$(n, 1)$	$(1, n)$	$(2, n–1)$	. . .	$(n–1, 2)$
b_3	$(n–1, 2)$	$(n, 1)$	$(1, n)$		$(n–2, 3)$
. . .	. . .	. . .	. . .	. . .	. . .
b_n	$(2, n–1)$	$(3, n–2)$	$(4, n–3)$	. . .	$(1, n)$

Consider the matching solution in which each girl gets her k^{th} choice for some fixed k with $(1 \le k \le n)$. Show that such a matching solution is stable.
(Hint: Consider any two pairs (a, α) and (b, β) and show a and β will not run away and elope. Notice that for each entry (x, y) in this preference matrix, we have $x + y = (n+1)$.)

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.