CS3230 : Design and Analysis of Algorithms (Spring 2015)

Homework Set #1 [Wake-up Call]

[*On pre-requisite material for CS3230* (*a revision, if you like*) and Stable Marriage]

OUT: 13-Jan-2015 **DUE:** Thursday, 22-Aug-2015, before 12:00 noon

First, read "Remarks about Homework".

Solve all the S-Problems in every homework set.

- Start each problem on a *new, separate* page.
- Make sure your name and matric number is on each sheet.
- Write legibly. If we cannot read what you write, we cannot give points. In case you CANNOT write legibly, please type out your answers and print out hard copy.
- To hand the homework in, **staple them together** and drop them into the CS3230 dropbox outside Prof. Leong's office (COM1 03-17) before the due date and time.

Note: This homework is largely designed to test your familiarity with the prerequisite materials from CS1231, CS2020/CS2010 – many of these problems can also appear in those classes. (*Yes, the "math" in CS1231 is very useful for CS3230, you'll see*.) The main aim of HW1 is to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.**

IMPORTANT:
Start on the homework *early*. *Start reading it RIGHT AWAY* (even if you can't start solving it.) For some problems, you need to let it incubate in your head before the solution will come to you.

Starting early gives you time to consult the teaching staff during their consultation hours. Some students might need some pointers regarding writing the proofs, others may need pointers on writing out the algorithm (idea, example, pseudo-code), while others will need to understand *more deeply* the material covered in class before they apply them to solving the homework problems. Use the consultation hours for these purposes!

It is always a good idea to email the teaching staff before going to the consultation hours or if you need to schedule other timings to meet. Do it in advance.

HOW TO "Give an Algorithm":

When asked to "give an algorithm" to solve a problem, your write-up should *NOT* be just the code. (*Giving code only will receive very low marks*.) Instead, it should be a short essay. The first paragraph should summarize the problem and what your results are. The body of the essay should consist of the following:

- A description of the algorithm *in English* and, if helpful, pseudo-code.
- One *worked example or diagram* to show more precisely how your algorithm works.
- A *proof* (or indication) of the correctness of the algorithm
- An *analysis* of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions which are described clearly. Convoluted and obtuse descriptions will receive low marks.

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 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*.

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

R1. **(a)** What is the *worst-case* running time for Bubblesort on n elements?
(b) What is the *worst-case* running time for Heapsort on n elements?
(c) What is the *worst-case* running time for building a BST using n repeated insertions?

R2. Show that (a) and (b) are true:
(a) $\sum_{k=1}^{N}(\ln k) = \Theta(N \log N)$ **(b)** $\sum_{k=1}^{N}(k) = \Theta(N^2)$
Where, in this HW1, have you seen algorithms with these running time analyses?

Standard-Problems -- Solve these problems and turn them in by the due-date.

S1. [Two fundamental processes in all of CS]

(a) (5 points) [Shock your friends – Guess the number game.]
Consider the following party game, commonly played among young people: Ken thinks of an integer between 1 and 500 (inclusive). He has a bag of 18 snickers bars. Steven is supposed to guess Ken's number. Each guess that Steven makes, Ken first eats a snickers bar, and then answers with one of the following:

"Correct"	if Steven's guess is correct,
"Lower"	if Steven's guess is wrong, and the correct number should be lower,
"Higher"	if Steven's guess is wrong, and the correct number should be higher.

When Steven guesses correctly, he wins all the remaining snickers bars in Ken's bag.

Please help Steven formulate *a strategy to win the maximum number* of snickers bars. Who gets to eat more snickers bars? Ken or Steven?

(b) (5 points) [Counting – the Crowd-Sourcing Way]
The following 5-step algorithm is *modified* from Prof. D.J. Majan's 3-step algorithm given in the first lecture for CS50 in Harvard University, 2012. It is supposed to be "*played*" in a lecture theatre full of students. (The only requirement is that the student can add two integers, work in pairs, and that they can synchronize all pairings (in Step 2), if needed.)

Algorithm: (Steps 2-3 are executed in "parallel" [and synchronized])
1. Stand up and think of the number 1. // *initialize*
2. Each person pairs up with someone standing; // *one phase*
   3.1 Add up your numbers and adopt the sum as your new number;
   3.2 One person from each pair sits down; (* *the other continues* *)
4. Go back to Step 2 until one person is left standing. // *a simple repeat loop*
5. The last person standing reports his number; // *Post-process*

Credit: Prof. D. J. Majan's first lecture for CS50 in Harvard University, 2012
Youtube video: https://www.youtube.com/watch?v=vuuJ8cGcl30 (8:35—12:00)

(i) We start with n students in the lecture theatre. Argue why this algorithm terminates? What is the number reported in Step 5? How many additions will be done in all?

(ii) In the synchronized case, in Step 2 *all the pairings happen at the same time,* say in one time unit. (In odd cases, *one* person will be unpaired in this phase, which is fine.) How many *phases* (of Step 2-3) will it take?

(iii) Answer part **(ii)** again, if we do not require that Steps 2 to be synchronized.

S2. [Stable Marriage Prioblem]

(a) [Executing Gale-Shapley algorithm] Run the Gale-Shapley algorithm *twice* for the preference lists given below. For the first run let the boys $\{a, b, c, d\}$ propose, and in the second run, let the girls $\{A, B, C, D\}$ propose.

boys	1st	2nd	3rd	4th
a	*C*	*D*	*B*	*A*
b	*C*	*A*	*D*	*B*
c	*B*	*C*	*D*	*A*
B	*C*	*B*	*D*	*A*

girls	1st	2nd	3rd	4th
A	*a*	*b*	*c*	*d*
B	*a*	*d*	*c*	*b*
C	*b*	*a*	*c*	*d*
D	*d*	*b*	*c*	*a*

Hence, or otherwise, conclude that there is only one stable pairing for this preference lists. [*Hint:* Use the theorem about male-optimal and female-pessimal when the boys propose.)

(b) [Solution is not unique] Given two boys $\{a, b\}$ and two girls $\{A, B\}$, design preference lists for the boys and the girls so that both possible pairings are stable.

(c) [Tough Instance] Design preference lists for 3 boys $\{a, b, c\}$ and 3 girls $\{A, B, C\}$ so that the Gale-Shapley algorithm will take 7 rounds (7 proposals made). Explain why this is the largest number of rounds for n=3. [*Note:* If you cannot force 7 rounds, try 6 rounds.]

S3. (10 points) [Room Assignment for Activities] You are in a summer camp, and there are n activities scheduled for the campers. Each activity $A_k = [s_k, e_k]$ has start time s_k and end time e_k. The problem is to assign the activities to rooms so that in each room, the activities do not "clash" (overlap in time). We want to find an *optimal room assignment* that *minimizes* the number of rooms used. As a CS3230 student, you are asked to help out.

[Line-Sweep Algorithm] You are given an algorithm (out of *many*) for solving this problem, as follows: Suppose there are n activities. Let T be the set of $2n$ start and end times of the n activities – we call them "event points". Each event point is either a start time or an end time. (For simplicity, we first assume that these $2n$ times are distinct.)

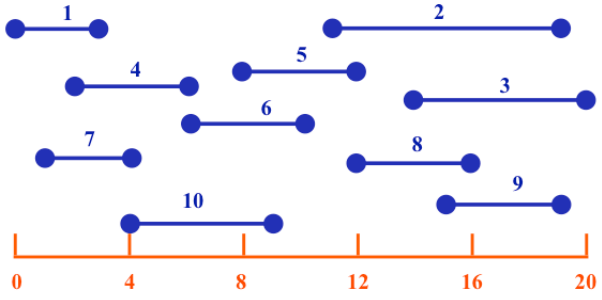
Sort the set T in increasing order. The algorithm starts with 0 room.
Next, the algorithm scans the event points in $T[1..2n]$, in increasing order.
For each *event point* $T[k]$, there are two cases:

Case 1: ($T[k]$ is the start time of activity A_k) We look for a "free" room. If a free room R exists, we assign A_k to room R and mark R "busy". If there is no "free" room, we start a new room R_{new}, assign A_k to room R_{new} and mark R_{new} "busy".

Case 2: ($T[k]$ is the end time of activity A_k) We look up the room R that the activity A_k was assigned to, and mark room R "free".

(a) An instance of this problem is given below:

$A_1 = [0, 3]$, $A_2 = [11, 18.9]$, $A_3 = [14, 20]$, $A_4 = [2, 5.9]$, $A_5 = [8, 11.9]$,
$A_6 = [6, 10]$, $A_7 = [1, 3.9]$, $A_8 = [12, 16]$, $A_9 = [15, 19]$, $A_{10} = [4, 9]$,



Note:
The intervals are not all drawn fully to scale.

(**S3(a)** continued…)

(i) Run the Line-Sweep Algorithm (LSA) on this problem instance. Draw the *partial room-assignment* done by the algorithm after it has finished processing the first 13 event points.
(ii) Draw also the *final room assignment* produced by the algorithm. How many rooms are required? Is this assignment optimal?

(b) You are told that the Line-Sweep Algorithm (LSA) will, *by magic*[1], always give an *optimal assignment*. Your task is to give an *efficient implementation* of the LSA to run in worst-case O(n lg n) time. Describe the data structures used to implement the set T and the set of rooms and whatever else to make your LSA algorithm efficient. Analyze the running time of your algorithm.
[*Note:* You should be implementing the LSA. DO NOT implement a different algorithm.]

(c) *Optional*: What if the event points are not all distinct? How will it affect the problem, the algorithm, the optimality of the assignment?

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.

A1. [Neighbour-Close Strings] A *neighbour-close* string is a string such that each character differs from the next character by *at most one* position. For example, the string "`ghhihgffffefg`" is a neighbour-close string. On the other hand, the string "`ghhihgffffefh`" is not a neighbour-close string, because the last letter "`f`" differs from its next character "`h`" by *two*, instead of one position. We want to delete as few characters as possible to make the remaining string neighbour-close. For the string "`absabkcb`", deleting the character "`s`" and "`k`" (two characters) gives "`ababcb`", which is neighbour close.

Formulate this problem as a graph problem, and give a *graph algorithm* for solving it.

A2. [Detailed analysis – continued from HW1-S1(a)]
This is a continuation of HW1-S1(a). But to simplify the analysis, we modify the game so that Ken thinks of a number from 1 to 512. Everything else remains the same. Now suppose we run the "Guess-the-number" game between Steven and Ken a total of 512 times and we keep the score of each game, and each time Ken will think of a different number (but they can be in any order). Assuming that Steven uses the same algorithm, do a detailed analysis of the total "score" of Steven (and Ken) over the 512 games. Who has the advantage, and by how much?

[1] You will prove this magic later in the course. ☺