CS3230: Design and Analysis of Algorithms (Spring 2020)

Homework Set #1

OUT: 24-Jan-2020 **DUE:** Thursday, 7-Feb-2020, 6:00pm

IMPORTANT: Write your NAME, Matric No, Tut. Gp in your Answer Sheet.

- Start each of the problems on a *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 in undergraduate general office before the due date and time.
- When you submit your homework, please try to make it short. The page limit is 8.

IMPORTANT:

You are advised to start on the homework *early*. For some problems, you need to let it incubate in your head before the solution will come to you. Also, start early so that there is time to consult the teaching staff during office 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 office hours for these purposes!

It is always a good idea to email the teaching staff before going for office 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. (*This 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 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.

(**Note:** Each problem is worth 10 marks.)

S1. Modified from Problem 3-3, pp 61-62 of [CLRS] [Sorting out order of growth rates]

Rank the following functions in *increasing order of growth*; that is, if function $f(n)$ is *immediately* before function $g(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$g_1(n) = n!$	$g_2(n) = 3n(\lg n)$	$g_3(n) = n^{\lg n}$	$g_4(n) = lg\ (n!)$
$g_5(n) = n^{(1/\lg n)}$	$g_6(n) = (\lg n)!$	$g_7(n) = 2^n - n^2$	$g_8(n) = n(\lg \lg n)$

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 four functions n^2, n, $(2013n^2 + n)$ and n^3 could be sorted in increasing order of growth as follows: ($n << n^2 \equiv (2013n^2 + n) << n^3$).

Do not turn in proofs for this problem, but you should do them anyway just for practice.
(*Note:* $\lg n = \log_2 n$.)

Soln:
$n^{(1/\lg n)} << n\ (\lg \lg n) << \lg (n!) \equiv 3\ n\ (\lg n) << (\lg n)! << n^{\lg n} << 2^n - n^2 << n!$

S2. [Copy from Term Test 2018/2019 Sem2, Solving recurrence]

(i) $T(n) = 2\ T(n/2) + n \lg \lg n$
(ii) $T(n) = 4\ T(n/2) + n$
(iii) $T(n) = 4\ T(n/2) + n \lg \lg n$
(iv) $T(n) = 4\ T(n/2) + n^2$
(v) $T(n) = 4\ T(n/2) + n^2 \lg \lg n$
(vi) $T(n) = 4\ T(n/2) + n^3$
(vii) $T(n) = 4\ T(n/2) + n^3 \lg \lg n$

(a) Among the above recurrences, please indicates those that can be solved by master theorem. For each recurrence that is solvable by master theorem, please indicate which case it belongs to (either case 1, 2, or 3) and state the time complexity.
(b) For recurrences that are not solvable by master theorem, please solve them.

Soln:
(a)

(i) $T(n) = 2\ T(n/2) + n \lg \lg n$:
NO!

(ii) $T(n) = 4\ T(n/2) + n$:
YES! Case 1, $\Theta(n^2)$.

(iii) $T(n) = 4\ T(n/2) + n \lg \lg n$:
YES! Case 1, $\Theta(n^2)$.

(iv) $T(n) = 4\ T(n/2) + n^2$:
YES! Case 2, $\Theta(n^2 \log n)$.

(v) $T(n) = 4\,T(n/2) + n^2 \lg \lg n$:
NO!

(vi) $T(n) = 4\,T(n/2) + n^3$:
YES! Case 3, $\Theta(n^3)$.

(vii) $T(n) = 4\,T(n/2) + n^3 \lg \lg n$:
YES! Case 3, $\Theta(n^3 \lg \lg n)$.

(b)
For (i), $T(n) = 2\,T(n/2) + n \lg \lg n$.

We can solve it using telescoping. Dividing both sides of equation (i) by n, we have:

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + \lg lg\, n \quad .$$

By telescoping, we also have:

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + \lg lg\,(n/2) \quad .$$

$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + \lg lg\,(n/4) \quad .$$

$$\frac{T(n/8)}{n/8} = \frac{T(n/16)}{n/18} + \lg lg\,(n/8) \quad .$$

…

$$\frac{T(2)}{2} = \frac{T(1)}{1} + \lg lg\,(2) \quad .$$

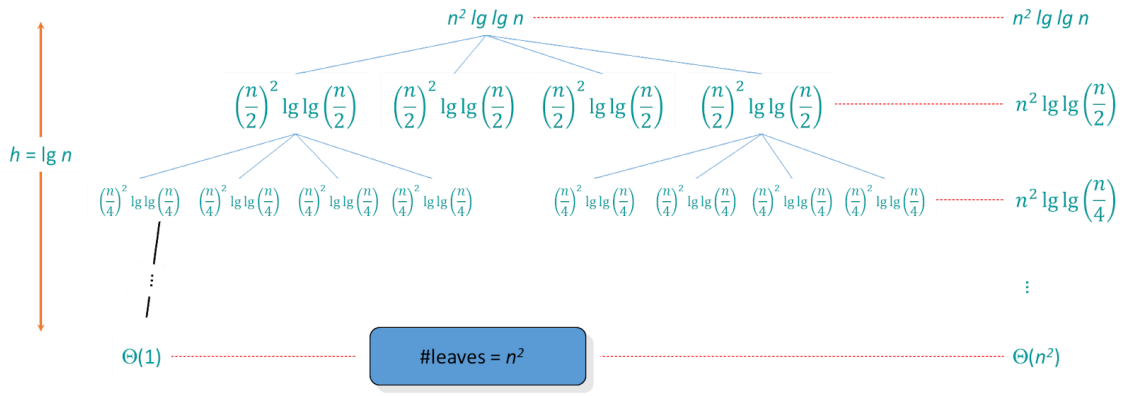
By summing over all equations, we have:

$$\frac{T(n)}{n} = \frac{T(1)}{1} + \lg lg\, n \; + \lg lg\,(n/2) \; + \ldots + \lg lg\,(2) \quad .$$

$$\frac{T(n)}{n} = \frac{T(1)}{1} + lg((lg\, n)!) \quad .$$

Hence, $T(n) = \Theta(n\, lg\, n\, \lg lg\, n)$.

For (v), $T(n) = 4\,T(n/2) + n^2 \lg \lg n$



$$T(n) = \sum_{i=0}^{\lg n - 1} n^2 \lg\lg\left(\frac{n}{2^i}\right) = n^2 \sum_{i=0}^{\lg n - 1} (\lg(\lg n - i)) = n^2 \lg\left((\lg n)!\right)$$

$$= \Theta(n^2 \lg n \lg\lg n)$$

S3. [Iterative algorithm]

Consider the function fun(x, y). What is its output? What is the invariant for the while loop? Can you show that this algorithm correctly compute the output?

```
fun(x, y)
1. ans=0; p=x; q=y;
2. while (q>0) do
3.     r = q mod 2; q = q div 2;
4.     ans = ans + r*p;
5.     p = p * 2;
6. return ans;
```

Solution: The output of the function is x*y.

Let $b_k b_{k-1} \ldots b_2 b_1 b_0$ be the binary representation of y. Then, the binary representation of y div 2^i is $b_k \ldots b_i$ and the binary representation of y mod 2^i is $b_{i-1} \ldots b_1 b_0$.

The invariant is: For the ith iteration, q is y div 2^i. p is x * 2^i, and ans is x * (y mod 2^i).

Initiation: In the 0^{th} iteration, i=0. We have q is y, p is x and ans is x * (y mod 2^0) = 0. This is exactly the assignment in step 1.

Maintenance: After the i^{th} iteration, q is y div 2^i, p is x * 2^i, and ans is x * (y mod 2^i).
Then, the (i+1)th iteration goes through steps 3-5.
After step 3,
r is q mod 2 = (y div 2^i) mod 2 = b_i.
q is set to be q div 2 = (y div 2^i) div 2, which is y div 2^{i+1}.

After step 4,
ans = $x*(y \bmod 2^i) + b_i * x * 2^i. = x*(y \bmod 2^i + b_i\, 2^i) = x*(y \bmod 2^{i+1})$

After step 5,
p is set to be $p * 2 = x * 2^{i+1}$.

In summary, after the $(i+1)^{th}$ iteration, q is $y \operatorname{div} 2^{i+1}$, p is $x * 2^{i+1}$, and ans is $x * (y \bmod 2^{i+1})$.

Termination: The while loop terminates when q=0, which is the k^{th} iteration. According to the invariant, ans is $x * (y \bmod 2^k) = x*y$.

The algorithm computes the correct output.