CS4102 - ALGORITHMS - SPRING 2020

DUE THURSDAY, FEBRUARY 6 AT 11P VIA COLLAB

H2(MST3K)

Collaboration Policy: You are encouraged to collaborate with up to 4 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: list your collaborators

Sources: list any sources, e.g. Cormen, et al, Introduction to Algorithms

PROBLEM 1 *Logs*

Prove that $\log n \in o(n^\varepsilon)$ for any positive constant $\varepsilon > 0$. That is, $\log n$ grows more slowly than any polynomial, even those with fractional powers, e.g., $\varepsilon = 0.00001$. (Note: you may find the limit definitions of the order-classes (CLRS, pp. 50-51) helpful for this problem.)

PROBLEM 2 *Master Theorem*

Prove a Θ asymptotic bound on the following recurrences using the Master Theorem. Be sure to state which case and why that case applies in your proof. If the Master Theorem does not apply, explain why it fails. (Note: you may need to use substitution before applying the Master Theorem.)

1. $T(n) = 6T(n/2) + n^2$
2. $T(n) = 27T(n/3) + n^3$
3. $T(n) = 4T(n/2) + n^2 \log n$
4. $T(n) = 9T(n/3) + n^2\sqrt{n}$
5. $T(n) = 3T(n/9) + \sqrt{n}$
6. $T(n) = 2T(\sqrt{n}) + 2$

PROBLEM 3 *Fast Exponentiation*

Given a pair of positive integers (a, n), give pseudo-code for a divide and conquer algorithm that computes a^n using only $O(\log n)$ multiplications. Prove that your algorithm is $\in O(\log n)$.

PROBLEM 4 *Castle Hunter*

We are planning a new board game called *Castle Hunter*. This game works similarly to *Battleship*, except instead of trying to find your opponent's ships on a two dimensional board, you're trying to find and destroy a castle in your opponent's one dimensional board. Each player will decide the layout of their terrain; castles are placed on every hill. Specifically, each castle is placed such that they are higher than the surrounding area, i.e., they are on a local maximum. (After all, hilltops are easier to defend!) Each player's board will be a list of n floating point values corresponding to the elevation. To guarantee that a local maximum exists somewhere in each player's list, we will force the first two elements in the list to be (in order) 0 and 1, and the last two elements to be (in order) 1 and 0.

To make progress, you name an index of your opponent's list, and she/he must respond with the value stored at that index (i.e., the elevation of the terrain). To win you must correctly identify that a particular index is a local maximum (the first and last elements don't count); i.e., find *one* castle. An example board is shown in Figure 1. [We will require that all values in the list, excepting the first and last pairs, be unique.]

0	1	4	23	18	14	15	13	1	0
0	1	2	3	4	5	6	7	8	9

Figure 1: An example board of size $n = 10$. You win if you can identify any one local maximum (a castle); in this case both index 3 and index 6 are local maxima.

1. Devise a strategy which will guarantee that you can find a local maximum in your opponent's board using no more than $O(\log n)$ queries, prove your run time and correctness.
2. Now show that $\Omega(\log n)$ queries are required by *any* algorithm (in the worst case). To do this, show that there is a way that your opponent could dynamically select values for each query as you ask them, rather than in advance (i.e., cheat, that scoundrel!) in such a way that $\Omega(\log n)$ queries are required by *any* guessing strategy you might use.