Discrete Structures :: CS 207 :: Autumn 2019

Practice Problem Set 12 Released: Sunday October 27, 2019

1. For each of the following, reason if it is true or false:

   (a) $3n^2 + 10n^{\log n} = O(n^2)$

   (b) $n^{\log n} = O(2^{\sqrt{n}})$

   (c) $n \log n + n/2 = \Theta(n \log n + \sqrt{n})$

2. Suppose $f(n) = O(g(n))$. For each of the following, reason if it is true or false:

   (a) $f(n)^2 = O(g(n)^2)$

   (b) $2^{f(n)} = O(2^{g(n)})$

   (c) $\log(f(n)) = O(\log(g(n)))$

3. In each case below, either apply the Master Theorem (as stated in class) to obtain a $\Theta(\cdot)$ expression for $T(n)$ (assuming T is a monotonously increasing function), or point out why the Master Theorem is not applicable. In the latter case, can you follow the argument used to prove the Master Theorem to still obtain a solution?

   (a) $T(n) = 3/2 \cdot T(\lceil n/2 \rceil) + n$, with $T(1) = 1$.

   (b) $T(2n) = 2T(n) + \sqrt{n}$, with $T(1) = 1$.

   (c) $T(n) = 2T(\lceil n/2 \rceil) + n \log n$, with $T(1) = 1$.

   (d) $T(n) = 2T(\lceil n/2 \rceil) + n^n$, with $T(1) = 1$

4. Find an exact solution to the following recurrence for n which are powers of 4. Then use induction to prove that your solution is correct.

   $$T(n) = 2T(\lceil n/4 \rceil) + \sqrt{n}$$

   for $n \geq 0$, and $T(1) = 1$.

5. As discussed in class, there are 2 reasonable algorithms for finding the maximum of n elements in an array – (1) scan the array one element at a time, and maintain the maximum of all the elements seen so far, or (2) divide the array into 2 halves (of almost equal sizes) and recursively find the maximum in those 2 halves, and compare them to find the maximum for the entire array.

   Extend both these algorithms so that they find both the maximum and the second maximum in an array of size $n \geq 2$. Use pseudocode to clearly describe your algorithms (taking care of corner cases). Derive upper bounds on the total number of comparisons made by the two algorithms, using big-O notation, as a function of n.

   For each of your algorithms, does it make exactly the same number of comparisons on all possible inputs? If not, what is the maximum and minimum number of comparisons it can make, over all possible inputs? For the purpose of analysis, you may consider n to be a power of 2 (though you should write your algorithms to work correctly for all $n \geq 2$).

6. Consider the following mystery function, which takes as input a list of real numbers and outputs a non-negative real number.

```
1: function Foo(A_1, A_2, ... A_n: real numbers)
2:     d := |A_1 − A_2|
3:     for i := 1 to n do
4:         for j := 1 to n do
5:             q := |A_i − A_j|
6:             if i ≠ j and q < d then
7:                 d := q
8:     return d
```

   (a) Give a brief English description of what the function Foo computes.

   (b) How many times are lines 5 and 6 executed, as a function of n?

   (c) Express the running time of the algorithm expressed as $\Theta(f(n))$ for a simple function f? Briefly justify your answer.

   (d) This is a really bad algorithm for this task. Write pseudocode for a function with a better big-O running time.

   Hint: First sort the input.

(e) Express the running time of your new algorithm from part d as a $\Theta(\cdot)$ expression. (You can use the fact that sorting can be done in $\Theta(n \log n)$ time. How much additional time do you take? What is the overall time?)

7. Given a rooted tree G, let the total number of nodes, the number of leaves and the height of the tree be denoted by $N(G)$, $L(G)$ and $H(G)$, respectively.

   In the following, fix a constant $m > 1$. Let $\mathcal{G}$ denote the set of all *full* m-ary rooted trees. In each of the following cases, give a function f as required, or argue that such a function f does not exist and give the best (smallest) possible function g as required.

   (a) $\forall G \in \mathcal{G}$, $L(G) = \Theta(f(N(G)))$, and $\forall G \in \mathcal{G}$, $L(G) = O(g(N(G)))$.

   (b) $\forall G \in \mathcal{G}$, $H(G) = \Theta(f(N(G)))$, and $\forall G \in \mathcal{G}$, $H(G) = O(g(N(G)))$.

   (c) $\forall G \in \mathcal{G}$, $L(G) = \Theta(f(H(G)))$, and $\forall G \in \mathcal{G}$, $L(G) = O(g(H(G)))$.

8. **Recursion and Induction.** A major open question in theoretical computer science (called the "Sensitivity Conjecture") was resolved this summer by Hao Huang. But as it turned out, the last step of this solution – which took over 30 years before Huang found it – was much simpler than anyone expected. If you are interested, you can read this step itself here: `https://www.cs.stanford.edu/~knuth/papers/huang.pdf`. In this problem, we shall restrict ourselves to expanding on a couple of easy observations used there.

   Recursively define a $2^n \times 2^n$ matrix, denoted by A_n as follows (here I_t stands for the $t \times t$ identity matrix):

$$A_0 = \begin{bmatrix} 0 \end{bmatrix}$$
$$A_n = \begin{bmatrix} A_n & I_{2^{n-1}} \\ I_{2^{n-1}} & -A_n \end{bmatrix} \qquad \text{for } n > 0.$$

   For example, $A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $A_2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$.

   (a) Prove by induction that $A_n^2 = nI_{2^n}$.

   (b) Suppose the 2^n rows and columns of A_n are numbered from left to right as $0, \ldots, 2^n - 1$. Then show that the $(i, j)^{\text{th}}$ entry of A_n is ± 1 if i and j differ in exactly one digit in their binary representations, and otherwise it is 0.

   Hint: If you ignored the negative sign in the definition of A_n, can you reinterpret the resulting recurrence as defining the adjacency matrix of a graph?