CS 6150: HW1 – Data structures, recurrences

Submission date: Friday, Sep 6, 2019 (11:59 PM)

This assignment has 6 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Basics	7	
Bubble sort and binary search	8	
Tries and balanced binary trees	5	
Recurrences, recurrences	17	
Computing intersections	5	
Dynamic arrays: is doubling important?	8	
Total:	50	

Question 1: Basics .... **[7]**

In (b)-(d) below, answer YES/NO, along with a line or so of reasoning. Simply answering YES/NO will fetch only half the points.

(a) **[1]** Sign up for the course on Piazza!

(b) **[2]** Let $f(n)$ be a function of integer parameter n, and suppose that $f(n) \in O(n^2)$. Is it true that $f(n)$ is also $O(n^3)$?

(c) **[2]** Suppose $f(n) = \Omega(n^2)$. Is it true that $f(n) \in o(n^3)$?

(d) **[2]** Let $f(n) = n^{\log n}$. Is $f(n)$ in $o(2^n)$?

Question 2: Bubble sort and binary search .... **[8]**

We will re-visit two simple algorithms we saw in class.

(a) **[4]** Recall the bubble sort procedure (see lecture notes): while the input array $A[]$ is not sorted, go over the array from left to right, swapping i and $(i+1)$ if they are out of order. Given a parameter $1 < k < n$, give an input $A[]$ for which the procedure takes time $\Theta(nk)$. (Recall that to prove a $\Theta(\cdot)$ bound, you need to show upper and lower bounds.)

(b) **[4]** Consider the binary search procedure for finding an element x in a sorted array $A[0, 1, \dots, n-1]$. We saw that the total number of comparisons made is exactly $\lceil \log_2 n \rceil$. Suppose we wish to do better. One idea is to see if we can recurse on an array of size $< n/2$. To this end, consider an algorithm that (a) compares the query x with $A[n/3]$ and $A[2n/3]$, and (b) recurses into the appropriate sub-array of size $n/3$. How many comparisons are done by this algorithm? (You should write down a recurrence, compute a closed form solution, and comment if it is better than $\log_2 n$.)

Question 3: Tries and balanced binary trees .... **[5]**

I mentioned in class that prefix trees can be used as a "poor man's binary search tree". Let us see the sense in which this is true. Suppose we have a set of N integers all in the range $[1, N^3]$. If we treat them as binary strings (using the natural binary representation of the integers), how long does it take to (a) add a new integer in the same range to the data structure? (b) to query if some x belongs to the set?

(Bonus): in what sense is a "regular" binary search tree better?

Question 4: Recurrences, recurrences .... **[17]**

Solve each of the recurrences below, and give the best $O(\cdot)$ bound you can for each of them. You can use any theorem you want (Master theorem, Akra-Bazzi, etc.), but please state the theorem in the form you use it. Also, when we write $n/2$, $n/3$, etc. we mean the floor (closest integer less than the number) of the corresponding quantity.

(a) **[5]** $T(n) = 2T(n/2) + T(n/3) + n$. As the base case, suppose $T(0) = 0$ and $T(1) = 1$.

(b) **[6]** $T(n) = nT(\sqrt{n})^2$. This time, consider two base cases $T(1) = 4$ and $T(1) = 16$, and compute the answer in the two cases.

(c) **[6]** Suppose you have a divide-and-conquer procedure in which (a) the divide step is $O(n)$ (for an input of size n, irrespective of how we choose to divide), and (b) the "combination" (conquer) step is proportional to the product of the sizes of the sub-problems being combined. Then, (i) suppose we are dividing into two sub-problems; is it better to have a split of $(n/2, n/2)$, or is it better to have a split of $n/4, 3n/4$? (or does it not matter?) Next, (ii) is it better to divide into two sub-problems of size $n/2$, or three sub-problems of size $n/3$? (Assume that the goal is to minimize the leading term of the total running time.)

Question 5: Computing intersections .... **[5]**

When answering search queries using an "inverted index", we saw that the key step is to take the intersection of the InvIdx sets corresponding to the words in the query. Suppose that these sets are of size $s_1, s_2, \dots, s_t$ respectively (say we have t words in the query). Naïvely, computing the intersection takes time $O(s_1 + s_2 + \cdots + s_t)$, which is not great if one of the s_i is large (e.g., if one of the words in the query is a frequently-occurring one). Give an alternate algorithm whose running time is

$$O\big(s(\log s_1 + \log s_2 + \cdots + \log s_t)\big), \quad \text{where } s = \min_{1 \le i \le t} s_i.$$

Question 6: Dynamic arrays: is doubling important? .... [**8**]

Consider the 'add' procedure for dynamic arrays (DA) that we studied in class. Every time the add operation was called and the array was full, the procedure created a new array of twice the size and copied all the elements, and then added the new element.

Suppose we consider an alternate implementation, where the array size is always a multiple of some integer, say 100. Every time the add procedure is called and the array is full (and of size n), suppose we create a new array of size $n + 100$, copy all the elements and then add the new element.

For this new add procedure, analyze the asymptotic running time for N consecutive add operations.