CS 6150: HW1 – Data structures, recurrences

Pranav Rajan, Collaborators: Adithya Badideymurali, Ryan Howell, Frost Office Hours

Submission date: Monday, Sep 13, 2021 (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 basics	5	
Deletion in prefix trees	6	
Binary search and test pooling	10	
Recurrences, recurrences	16	
Dynamic arrays: is doubling important?	6	
Total:	50	

Note. *When asked to describe and analyze an algorithm, you need to first write the pseudocode, provide a running time analysis by going over all the steps (writing recurrences if necessary), and provide a reasoning for why the algorithm is correct.* Skipping or having incorrect reasoning will lead to a partial credit, even if the pseudocode itself is OK.

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! - Yes

(b) [**2**] Let $f(n)$ be a function of integer parameter n, and suppose that $f(n) \in O(n \log n)$. Is it true that $f(n)$ is also $O(n^2)$?
Ordering of Dominating Terms: $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3$....
Since $n^2 > \log n$, in the list of ordering of dominating terms, it is correct to say that $f(n)$ is also On^2.

(c) [**2**] Suppose $f(n) = \Omega(n^{2.5})$. Is it true that $f(n) \in o(n^5)$?
Ordering of Dominating Terms: $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3$....
By the definition of Ω which represents a lower bound, our algorithm cannot have a better runtime than $n^{2.5}$ In the case of the upper bound, we are not limited to $o(n^5)$ since any class of functions with a power greater than 2.5 is a valid upper bound according to the order of dominating terms. Thus it is incorrect to say that $f(n) \in o(n^5)$.

(d) [**2**] Let $f(n) = n^{\log n}$. Is $f(n)$ in $o(2^{\sqrt{n}})$?
Using L'Hopital's rule we can prove that $f(n) = o(g(n))$ by showing $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
$\rightarrow 0 < n^{\log n} < c \cdot 2^{\sqrt{n}}$
$\rightarrow \log(n^{\log n}) < c \cdot \log 2^{\sqrt{n}}$
$\rightarrow \log n(\log n) < \sqrt{n} \log(2)$
$\rightarrow \lim_{n\to\infty} \frac{\log n(\log n)}{\sqrt{n}} = 0$ by L'Hopital's Rule.
Since we have proven L'Hopital's Rule to hold true, we can say that $n^{\log n}$ is $o(2^{\sqrt{n}})$.

Question 2: Bubble sort basics ..... **[5]**

Recall the bubble sort procedure we saw in Lecture 1 (see 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. As I mentioned in class, the running time of the algorithm depends on the input.

Given a parameter $1 < k < n$, give an input array $A[]$ for which the bubble sort procedure takes time $\Theta(nk)$. (Recall that to prove a $\Theta(\cdot)$ bound, you need to show upper and lower bounds.)

Question 3: Deletion in prefix trees ..... **[6]**

In class, we saw how a prefix tree can be used to store a *set of strings* (which we called a dictionary) over some alphabet Σ. Specifically, we saw how to implement the `add` and `query` operations on such a data structure. (See the lecture notes for more details.) Now, consider implementing the `delete` operation. As suggested in the notes, one way to do this is to mimic the query, and for the node corresponding to the word, set the "IsWord" boolean to false. However, if we are adding and removing multiple strings, this can lead to many tree paths that were created, but don't correspond to any word currently in the dictionary.

Show how to modify the data structure so that this can be avoided. More formally, if S is the set of words remaining after a sequence of add/delete operations, we would like to ensure space utilization that is the same (possibly up to a constant factor) of the space needed to store only the elements of S. If your modifications impact the running time of the `add`, `query`, and `delete` operations, explain how.

Question 4: Binary search and test pooling ..... **[10]**

"Test pooling" is a trick that is used when testing for a disease is expensive or has limited availability. The idea is the following: suppose we have n people (numbered $1, 2, \ldots, n$ for convenience), instead of testing each one, samples from a subset S of the people are combined and tested, where a test runs in $O(1)$ time regardless of the size of S. If at least one of the people in S has the disease, the test comes out positive, and if none of the people in S has the disease, it comes out negative. (Let us ignore the test error for this problem.)

It turns out that if only a "few" people have the disease, this is much better than testing all n people.

(a) **[4]** Suppose we know that *exactly one* of the n people has the disease and our aim is to find out which one. Describe an algorithm that runs in time $O(\log n)$ for this problem. (For this part, pseudocode suffices, you don't need to analyze the runtime / correctness.)

(b) **[6]** Now suppose we know that *exactly two* of the n people have the disease and our aim is to identify the two infected people. Describe **and analyze** (both runtime and correctness) an algorithm that runs in time $O(\log n)$ for this problem.

Question 5: Recurrences, recurrences ..... **[16]**

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 or equal to the number) of the corresponding quantity. **Please show how you obtained your answer.**

(a) **[4]** $T(n) = 3T(n/3) + n^2$. As the base case, suppose $T(n) = 1$ for $n < 3$.

(b) **[6]** $T(n) = 2T(n/2) + T(n/3) + n$. As the base case, suppose $T(0) = T(1) = 1$.

(c) **[6]** $T(n) = 2(T(\sqrt{n}))^2$. As the base case, suppose $T(1) = 4$.

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

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 32. Every time the add procedure is called and the array is full (and of size n), suppose we create a new array of size $n + 32$, 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.