CS 6150: HW1 – Data structures, recurrences

Submission date: Monday, Sep 13, 2021 (11:59 PM)

This assignment has 6 questions, for a total of 57 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Basics	14	
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:	57	

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 ..... **[14]**

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 $f(n) \in O(n \log n)$. Is it true that $f(n)$ is also $O(n^2)$?

(c) [**2**] Suppose $f(n) = \Omega(n^{2.5})$. Is it true that $f(n) \in o(n^5)$?

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

Solution

(a) [**1**] Yes, I already signed up.

(b) [**2**] Yes. $f(n) \in O(n \log n) \Rightarrow \exists n_0, c \; s.t. f(n) \leq cn \log n, \; \forall n \geq n_0$, which also implies $f(n) \leq cn^2, \; \forall n \geq n_0$ and thus $f(n) \in O(n^2)$.

(c) [**2**] No. For example, let $f(n) = n^6$ then $f(n) \in \Omega(n^{2.5})$. However, $f(n) \notin o(n^5)$.

(d) [**2**] Yes. Let $f(n) = n^{\log n} = 2^{(\log n)^2}$ and $g(n) = 2^{\sqrt{n}}$, then we can see that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^{(\log n)^2}}{2^{\sqrt{n}}} = \lim_{n\to\infty} 2^{(\log n)^2 - \sqrt{n}} = 0$ which implies $f(n) \in O(2^{\sqrt{n}})$.

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.)

Answer.
Suppose we have the following array,

$$A = [n-(k-1), n-(k-2), ..., n, 1, 2, ..., n-k] \tag{1}$$

If we run the algorithm, it will first bubble up the n and then bubble up $n-1$ and so on until $n-(k-1)$ where we have done k iterations and we would have a sorted array. We can easily see that each iteration takes n times and we need to do k rounds. Thus the overall time complexity is exactly nk. The upper bounds can be $2nk$ while the lower bounds can be nk, so the it is a $\Theta(nk)$.

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.

Answer.

We can delete the unneeded node. Please check pseudocode shown in below Algorithm for more details. The general idea is to delete the node without descendent recursively. Its running time is $O(|w|)$, the length of the word being deleted. It won't influence the running time of **add, query, and delete**.

Algorithm 1 Delete key

Input string

```
1: procedure DELETION
2:     node ← last node of after querying string
3:     node.isword=False
4:     while (descendant of node is null) and (node.isword is False) do
5:         parent ← node's parent
6:         delete node
7:         node ← parent
```

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.)

Answer.

The pseudocode is shown in algorithm 2

(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.

Answer.

The pseudocode is shown in algorithm 3. The general idea is to use binary search to find the two people with disease.

Algorithm 2 Binary-Search-test

```
Input n people
Output Binary-Search-test(1,n)
function BINARY-SEARCH-TEST(left, right)
    if left = right then
        if test(left)==positive then
            return left
    middle ← left + (right − left)//2
    if test(left:middle)==positive then
        return Binary-Search-test (left,middle)
    else
        return Binary-Search-test (middle+1,right)
```

Algorithm 3 Binary-Search-test-2people

```
Input n people
Initialize result=[] #to store results
Output result=Binary-Search-tests(result,l,n)

function BINARY-SEARCH-TEST—TWOPEOPLE(result,left, right)
    if left = right then
        if test(left)==positive then
            append left to result
            return
    middle ← left + (right − left)//2
    if test(left:middle)==positive then
        return Binary-Search-test (result,left,middle)
    if test(middle+1:right)==positive then
        return Binary-Search-test (result,middle+1,right)
```

Running time. We can split the group $\log(n)$ times. And since there are only two people is positive, the worst case is that the two people are in different groups. At each time, we only need to test at most four times since at most two parent nodes are positive. So the runining time is $O(4\log(n)) = O(log(n))$

Correctness: At initial step, we can divide the n people into two halves, where the two positive people may in one group or different groups. But we will only select the groups with positive people. So only the positive groups are left after the first step. Recursively, we will always keep the groups with positive people when parent group is positive. When the group size is 1, we will append it to the result. So it is correct.

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 $n < 3$.
Answer.
Method:Guess-n-prove. We want to prove by induction that $T(n) < Cn^2$, for base case $n < 3$, it is true $\forall C > 1$. In induction we will assume that $T(n) < Cn^2 \forall k < n$, so we have to prove it is tru for $k = n$,

$$T(n) = 3T(n/3) + n^2 < 3C\frac{n^2}{9} + n^2 = \frac{(C+3)n^2}{3} \tag{2}$$

$\forall C > \frac{3}{2}$, we could have $T(n) < Cn^2$.
We can apply the Akra–Bazzi method to get a more tight upper bound. We can solve $p = 1.36$,

$$T(n) = \Theta(n^{1.36}(1 + \int_1^n x^{-1.36}dx) = \Theta(n^{1.36}(1 + \frac{n^{-0.36} - 1}{-0.36}))) \in O(n^{1.36}) \tag{3}$$

(b) [**6**] $T(n) = 2T(n/2) + T(n/3) + n$. As the base case, suppose $T(0) = T(1) = 1$. **Answer.**
Method:Guess-n-prove. We want to prove by induction that $T(n) \leq Cn^2$, for base case $n = 0, 1$, it is true $\forall C > 1$. In induction we will assume that $T(n) < Cn^2 \forall k < n$, so we have to prove it is tru for $k = n$,

$$T(n) = 2T(n/2) + T(n/3) + n < 2C\frac{n^2}{4} + C\frac{n^2}{9} + n = \frac{11Cn^2}{18} + n \tag{4}$$

There exists a C to satisfy above equation, so we could have $T(n) < Cn^2$.

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

Answer.
method:Plug-n-chug.

$$\begin{aligned} T(n) &= 2(T(n^{\frac{1}{2}}))^2 \\ &= 2 \cdot 2^2(T(n^{\frac{1}{4}}))^4 \\ &= 2 \cdot 2^2 \ldots 2^{2^{(b-1)}}(T(n^{\frac{1}{2^{(b)}}}))^{2^b} \end{aligned} \tag{5}$$

Now to find b, we know the base case of $T(2)$, so $n^{\frac{1}{2^b}} = 2 \Rightarrow b = log_2 log_2(n)$. in the analysis of

the base cases we have $T(2) = 2^5$,

$$\begin{aligned} T(n) &= 2 \cdot 2^2 \dots 2^{2^{b-1}} (T(n^{\frac{1}{2^b}}))^{2^b} \\ &= 2 \cdot 2^2 \dots 2^{2^{b-1}} (2^5)^{2^b} \\ &= 2 \cdot 2^2 \dots 2^{2^{b-1}} (2^5)^{\log_2 n} \\ &= 2 \cdot 2^2 \dots 2^{2^{b-1}} n^5 \\ &= 2^{1+2+4+\dots+2^{b-1}} n^5 \\ &= 2^{2^b-1} n^5 \\ &= (\frac{1}{2}) 2^{2^b} n^5 \\ &= (\frac{1}{2}) 2^{2^b} n^5 \\ &= (\frac{1}{2}) n \cdot n^5 \\ &= (\frac{1}{2}) n^6 \\ &= O(n^6) \end{aligned} \tag{6}$$

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.

Answer.

Let's assume that the initial size of the dynamic array is C. When we create the new array for r_1 times, we have done $C \cdot (1 + 2 + 3 + \dots + r_1) = C\frac{\cdot(1+r_1)r_1}{2}$ times copy. $C \cdot r_1 = N \Rightarrow r_1 = \frac{N}{C}$, So the running time of this N consecutive add operations is $\frac{CN+N^2}{2C}$, namely $O(N^2)$.