CSOR 4231 Midterm Exam 1
October 8, 2019, 10:10 AM

Rules; Answer each question completely and concisely. When you give an algorithm, be sure to give the most efficient one you can, explain why it is correct, and to analyze its running time. You do not need to repeat the details or pseudocode for any algorithm used in class, you can use it as a black-box.

Answer in the pages given. You can use the fronts and backs. Please make clear what you expect the graders to read and what is scratch work.

Question	Points	Total
1	10	10
2	12.5	15
3	5	10
4	9	15
Total	36.5	50

Name: Hongmin Zhu (hz2637)

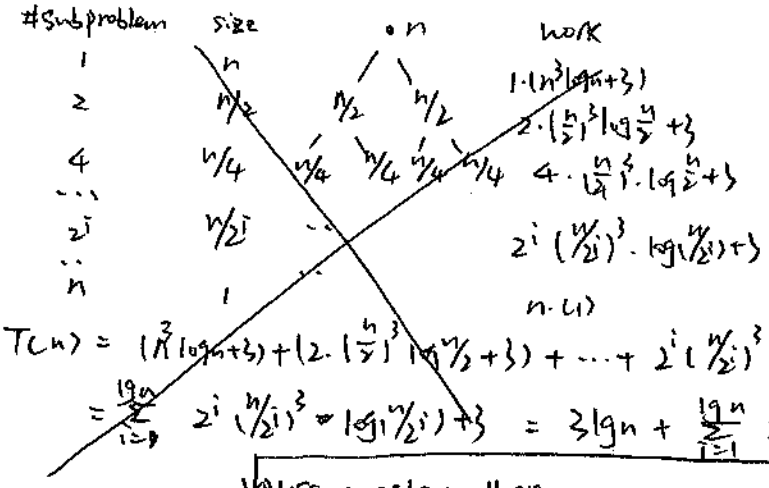
Please read and sign the following:

HONOR PLEDGE: I pledge that I have neither given nor received unauthorized aid during this examination.

Signature: Hongmin Zhu

Problem 1. [10 POINTS] Solve the following recurrences by giving big-O upper bounds. Justify the solution, either by the master theorem, or some other method learned in class.

a) [5 POINTS] $T(n) = 4T(n/2) + n^3 \log n + 3$



$T(n) = (n^3 \log n + 3) + (2\cdot(\frac{n}{2})^3 \log \frac{n}{2} + 3) + \cdots + 2^i((n/2^i)^3 \log n/2^i + 3)$

$= \sum_{i=0}^{\lg n} 2^i (n/2^i)^3 \cdot \log(n/2^i) + 3) = 3\lg n + \sum_{i=1}^{\lg n} 2^i (n/2^i)^3 \log(n/2^i) = \sum_{i=1}^{\lg n} \frac{n^3}{4^i}(\lg n - i) + 3\lg n$

=

Use master theorem

$a = 4, b = 2, f(n) = n^3 \log n + 3$

$n^{\log_b a} = n^{\log_2 4} = n^2 < f(n)$

$af(n/b) \le c \cdot f(n)$

$4(n/2)^3 \log(n/2) \le c \cdot n^3 \log n + 3c$

$\frac{n^3}{2} \log n/2 \le c \cdot n^3 \log n + 3c$

$\frac{n^3}{2}\log n - \frac{n^3}{2} \le c n^3 \log n + 3c$

$\frac{n^3}{2}\log n \le c \cdot n^3 \log n + 3c + \frac{n^3}{2}$

we choose $c = \frac{1}{2}$, so $af(n/b) \le c \cdot f(n)$

by case 3 of master theorem

$T(n) = \Theta(n^3 \log n)$ 5

b) [5 POINTS] $T(n) = T(n-1) + \log n$

$T(n) = T(n-1) + \log n$

$= T(n-2) + \log(n-1) + \log(n)$

$= T(n-3) + \log(n-2) + \log(n-1) + \log(n)$

$\cdots$

$= T(0) + \log(1) + \log(2) + \cdots + \log(n)$

Assume $T(0) = 1$

$T(n) = 1 + \log 1 + \log 2 + \cdots + \log n \le 1 + \log n + \log n + \cdots + \log n = O(n \log n)$

5

$a=8 \quad b=2$

$n^{\log_b a} = n^{\log_2 8} = n^3 = n^3$

$O(n^3 \cdot \lg n)$

Problem 2 [15 POINTS] Short Answer

a)[5 POINTS] Consider $T(n) = 8T(n/2)+n^3$. Consider the following proposed proof by substitution that $T(n) = O(n^3)$.

$$\begin{aligned} T(n) &= 8T(n/2) + n^3 \\ &= 8O((n/2)^3) + n^3 \\ &= 8O(n^3/8) + n^3 \\ &= O(n^3) + n^3 \\ &= O(n^3) \,. \end{aligned}$$

证明proof有问题 是证明过程有问题中 非具结论.

make a guess rewrite claim like $T(n) \le cn^3$ Then it would derive that $T(n) \le c(n)n^3$ There is no c to make it satisfy the ques

Is the statement correct? Is the proof correct? Why or why not? If you think the proof is incorrect, please point out where the error is.

+4.5

$T(n) = 8T(n/2) + n^3$

$a=8, \ b=2, \ f(n)=n^3$

$n^{\log_b a} = n^{\log_2 8} = n^3 = f(n)$

$\therefore$ by master theorem case 2,

$T(n) = \Theta(n^3 \log n)$

$\therefore$ the statement is wrong,

the proof is also wrong, the proof ~~throws big-O conclusion~~ uses the statement itself to proof, so it's circular, the proof should use induction, and throws big-O conclusion at the end

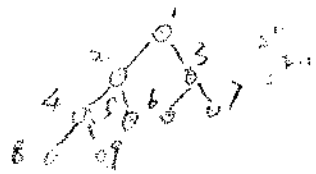
+5

b)[5 POINTS] Ollie the Optimist tells you about a great data structure he has designed that can support both of the operations INSERT and DELETE-MIN in $O(1)$ worst-case time. Explain to Ollie why he must be mistaken.

if the data structure put numbers in sorted way, Delete-min can be $O(1)$ time, however, since ~~it~~ numbers are sorted, Insert a number will take $O(n)$ time

if the data structure ~~is like Hashmap, inserting number~~ doesn't care about the order of numbers, so we can insert a number in anywhere, that only takes $O(1)$ time. However, if we try to find the min, we need to loop over all the numbers to find the smallest, that takes $O(n)$ time,

in summary, we can't satisfy Insert and Delete-Min using $O(1)$ time at the same time



$2^{i-1} = 4$

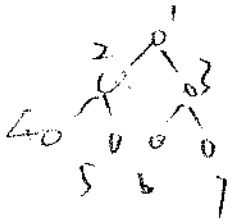
$i = 3$

$2^{i-1} = n$

$n = 4$

$4 = \log_2 h+1 \quad h-1 = 2 \Rightarrow h = 3, \quad \lfloor \log n \rfloor = 2$

c)[5 POINTS] In a MAX-HEAP of size n, in which positions might the 2nd smallest element be.



2nd Smallest element means an element that ranks at 2 to the last by the definition of max-heap, each root or subroot is bigger than its two children considering the tree form of heap.

the heap size is n, so there are $\lfloor \log n \rfloor + 1$ layers in the tree, so for smallest in max heap, we consider from bottom to root.

~~if 2nd Smallest is~~

if $n \geq 7$, the 2nd smallest is on the leaves, last layer

if $4 \leq n \leq 5$, the 2nd smallest is on the second layer

if $1 \leq n \leq 3$, the 2nd Smallest is on the first layer

by the definition of max heap, each node is bigger than its two children, so if consider heap in the form of tree, we need to find from bottom to top (from leaves to root)

Since we are looking ~~from~~ for 2nd smallest, ~~which is~~

if $n > 5$, the 2nd smallest is on the leaves, which is the last layer of ~~the~~ tree

if $3 \leq n < 5$, the 2nd smallest is on the second layer of tree

if $1 \leq n < 3$, the 2nd smallest is the root, at the first layer of tree

one of the leaf nodes or a parent of a leaf node

+3

Problem 3. [10 POINTS] Suppose that you are given an array $A[1 \ldots n]$ where each value $A[i]$ in the array is a random integer chosen uniformly from the set of integers $\{1, 2, \ldots, n\}$. We say that an array entry is "well placed" if $|A[i] - i| \leq 2$. What is the expected number of well-placed entries? (Hint: use indicator random variables to give your answer.)

- Let X_i be the indicator random variable associated with the event in which $|A[i]-i| \leq 2$
- $X_i = I\{|A[i]-i| \leq 2\}$
- Let X be the random variable denoting total number of well-placed entri
- $X = \sum_{i=1}^{n} X_i$
- take the expectation for both sides

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

So the task is to calculate $E[X_i]$

for each entry i, to satisfy $|A[i]-i| \leq 2$, $A[i]$ could be chosen from $i-2, i-1, i, i+1, i+2$, 5 numbers in total, (chosen uniformly)

so $E[X_i] = Pr(X_i = 1) \cdot 1 + Pr(X_i = 0) \cdot 0$

$$= \frac{5}{n}$$

Consider $i=1, 2, n-1, n$ seperately

$$\therefore E[X] = \sum_{i=1}^{n} E[X_i] = 5 \sum_{i=1}^{n} \frac{1}{n} = 5 \ln n \quad \times.$$

i from 1 to n not n!!

$i=1, i=2, i=n-1, i=n$

边4个取不到前后的2个，分开考虑

$$\Rightarrow \frac{3}{n} + \frac{4}{n} + \frac{5}{n}(n-4) + \frac{4}{n} + \frac{3}{n}$$

$$\Rightarrow \frac{5n-6}{n}$$

Problem 4. [15 POINTS]

You are given two sorted arrays $X[1 \ldots n]$ and $Y[1 \ldots n]$. Together they have $2n$ numbers. You want to find the median of those $2n$ numbers. (Note: if your algorithm returns a number very close to the median, that is fine. In other words, we are not checking whether you return the lower or upper median.)

a) [5 POINTS] Give a $\Theta(n)$ time algorithm to find the median of these two ~~numbers~~ arrays.

① Combine X and Y into a big array A, so A has 2n entries

② use the SELECT algorithm mentioned in class to find the median

findmedian (X, Y)

1. A ← combine X and Y without sorting
2. & median ← SELECT (A)

running time: Combine 2 arrays $O(n)$

SELECT algorithm $O(n)$

So the total running time is $O(n)$

X[1, ..., n] Y[1, ..., n]

T(n) = 2T(n/2) + 1
= 4T(n/4) + 2
= 2^i T(1) + i

b) [**10 POINTS**] Give an $O(\log n)$ time algorithm to find the median of these two ~~numbers~~ arrays. (Hint: Use recursion.)

T(n) = T(n/2) + 1

X[n] = X[n/2]

X[n] <

```
Median(X, Y, p, r)
1.  m ← (p+r)/2
2.  if m > X[n/2 - 1]
3.      Median(X[0, n/2], Y, m, r)
4.  else
5.      if m > Y[n/2 - 1]
6.          Median(X, Y[0, n/2], p, m)
7.      else
~~the algorithm should use time T(n) = T(n/2) + O(1)~~
~~So that the total running time is O(log n)~~
8.          Median(X, Y[n/2, n], m, r)
```

idea 3 but no details

don't see n/2

$T(n) = T(n/2) + O(1)$

$T(n) = O(\log n)$