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
2		15
3		10
4		15
Total		50

Name: ____________________________

Please read and sign the following:
HONOR PLEDGE: I pledge that I have neither given nor received unauthorized aid during this examination.

Signature: ____________________________

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**] $T(n) = 4T(n/2) + n^3 \log n + 3$

$T(n) = \Theta(n^3 \log n) = O(n^3 \log n)$
We apply the master theorem here.
Since a = 4, b = 2, $n^{log_b^a} = n^2$. $f(n) = n^3 \log n + 3 = \Omega(n^{2+\epsilon})$ and for $c = \frac{2}{3} < 1$ and all sufficient large n, $af(n/b) = \frac{n^3}{2}log(\frac{n}{2})+12 < cf(n) = \frac{2}{3}n^3logn+2$. Thus, $T(n) = \Theta(n^3 \log n) = O(n^3 \log n)$.

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

$T(n) = O(nlog(n))$
We can apply this formula iteratively to see that:

$T(n) = T(n-1)+log(n) = T(n-2)+log(n)+log(n-1) = ... = T(0)+log(n!)$

$log(n!) > log(n * (n-1) * .... * (\frac{n}{2})) > log(\frac{n}{2})^{\frac{n}{2}} = \frac{n}{2}log(\frac{n}{2})$ Thus, $T(n) = \Omega(nlog(n))$ Also we have $log(n!) < log(n^n) = nlogn$ Thus, $T(n) = O(nlog(n))$.
So, $T(n) = O(nlog(n))$

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 $T(n) = O(n^3)$.

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

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.

Solution:
Neither statement nor proof are not correct. By using induction it should not directly substitute $T(n) = O(n^3)$. Instead it should make a guess and rewrite the claim like $T(n) \leq cn^3$. Then it would derive that $T(n) \leq (c+1)n^3$. There does not exist c to make it satisfy the guess. Thus the guess is wrong.

Based on the master theorem, $T(n) = O(n^3 log(n))$.

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.

Solution:
If there exists such a data structure supporting both INSERT and DELETE-MIN, we can implement an $O(n)$ sort algorithm using this data structure. Given any array with n numbers, we first insert the n numbers into this data structure then DELETE-MIN n times. Since the i-th DELETE-MIN will find the i-th smallest number in the array, we can simply collect all the values of DELETE-MIN to get a sorted array. But we know, comparion-based sort algorithm is at least $O(nlogn)$, which causes contradiction.

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

Solution:
The 2nd smallest element of a max-heap might be one of the leaf nodes or a parent of a leaf node.

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

Solution:

$$\begin{aligned}
&E[\sum_i I(|A[i] - i| \le 2)] \\
=&\sum_i E[I(|A[i] - i| \le 2)] \\
=&\frac{3}{n} + \frac{4}{n} + \frac{5}{n}(n-4) + \frac{4}{n} + \frac{3}{n} \\
=&\frac{5n-6}{n}
\end{aligned}$$

Problem 4. [**15 POINTS**]

You are given two sorted arrays $X[1 \dots n]$ and $Y[1 \dots n]$. 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.

Solution

```
function MEDIAN($X, Y, n$)
    $i \leftarrow 1$ $j \leftarrow 1$
    while $n > 1$ do
        if $X[i] < Y[j]$ then
            $i \leftarrow i + 1$
        else
            $j \leftarrow j + 1$
        end if
        $n \leftarrow n - 1$
    end while
    if $X[i] < Y[j]$ then
        return $X[i]$
    else
        return $Y[i]$
    end if
end function
```

Proof of Correctness:

After k-th while loop, $min(X[i], Y[j])$ is the $(k+1)$-th smallest number of $X \cup Y$. Therefore, after $(n-1)$-th while loop, the return value is the n-th smallest number of $X \cup Y$, which is the lower median.

Runtime:

Because there are n-1 while loop and computation within one loop is constant time, the algorithm is $\Theta(n)$.

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

Solution:

```
function MEDIAN($X, Y, X_l, X_r, Y_l, Y_r$)
    if $X_l = X_r$ then
        return $X[X_l]$
    else
        $X_m \leftarrow (X_l + X_r)/2$
        $Y_m \leftarrow (Y_l + Y_r + 1)/2$
        if $X[X_m] \leq Y[Y_m]$ then
            return MEDIAN($X, Y, X_l, X_m, Y_m, Y_r$)
        else
            return MEDIAN($X, Y, X_m, X_r, Y_l, Y_m$)
        end if
    end if
end function

MEDIAN($X, Y, 1, n, 1, n$)
```

Proof of Correctness:

Function MEDIAN is to find the median number in $X[X_l...X_r] \cup Y[Y_l...Y_r]$. $X[X_m]$ and $Y[Y_m]$ are the median numbers of $X[X_l...X_r]$ and $Y[Y_l...Y_r]$ respectively. We first compare $X[X_m]$ and $Y[Y_m]$

If $X[X_m] \leq Y[Y_m]$, we know that the numbers in $X[X_l...X_m - 1]$ are all less than or equal to the numbers in $X[X_m...X_r] \cup Y[Y_m...Y_r]$. Since $X[X_m...X_r] \cup Y[Y_m...Y_r]$ contains more than a half numbers of $X[X_l...X_r] \cup Y[Y_l...Y_r]$, it implies the numbers in $X[X_l...X_m - 1]$ must be less than or equal to the median number of $X[X_l...X_r] \cup Y[Y_l...Y_r]$. Similarly, we can know that $Y[Y_m + 1...Y_r]$ are all greater than or equal to the median number of $X[X_l...X_r] \cup Y[Y_l...Y_r]$. So we can discard $X[X_l...X_m - 1]$ and $Y[Y_m + 1...Y_r]$ and keep the median number unchanged because the sizes of the discarded part - $X[X_l...X_m - 1]$ and $Y[Y_m + 1...Y_r]$ are same. This means to find the median number in $X[X_l...X_r] \cup Y[Y_l...Y_r]$ is equal to find the median number in $X[X_m...X_r] \cup Y[Y_l...Y_m]$.

If $X[X_m] > Y[Y_m]$, we can discard $X[X_m + 1...X_r]$ and $Y[Y_l...Y_m - 1]$ based on similar reasons. Thus, to find the median number in $X[X_l...X_r] \cup Y[Y_l...Y_r]$ is to find the median number in $X[X_l...X_m] \cup Y[Y_m...Y_r]$.

Runtime:

Because at each step we discard a half of numbers and each step's comparison is constant time, we have

$$T(n) = T(\frac{n}{2}) + O(1)$$

So, the computation time is $O(log(n))$