CS 141 midterm1B

Celyna Su

TOTAL POINTS

53 / 90

QUESTION 1

1 Problem 1 9 / 15

+ **0 pts** Completely wrong or no answer

+ **2 pts** Some ideas of iterative substitution, but the derivation is completely incorrect.

✓ **+ 6 pts** **Iterative substitution step T(n)=T(n/9^i)+sqrt{n}(1/3^{i-1}+1/3^{i-2}+...+1) correct.**

+ **3 pts** Geometric sum step T(n)=T(n/3^i)+(3/2)\sqrt{n}(1-1/3^{i}) correct

✓ **+ 3 pts** **Determining i step i = log_9(n) correct**

+ **3 pts** Exact solution T(n) = (3\sqrt{n}-1)/2 correct (OK even if not completely simplified T(n)=1+(3/2)\sqrt{n}(1-1/3^{log_9 n})

QUESTION 2

2 Problem 2 15 / 15

+ **0 pts** Completely wrong or no answer

✓ **+ 2 pts** **a=2, b=4, f(n)=\sqrt{n}\log(n), note that n^{log_4 2}=\sqrt{n}**

+ **4 pts** Recognizes that case II applies, but not correct choice/justification

✓ **+ 10 pts** **Case II applied correctly, i.e., f(n) \in \Theta(\sqrt{n} log^k n) for k=1**

✓ **+ 3 pts** **Conclusion correct T(n) \in \Theta(\sqrt{n} log^2 n)**

QUESTION 3

3 Problem 3 15 / 15

+ **0 pts** Completely wrong or no answer

+ **3 pts** T(n) = T(something wrong) + O(n)

+ **5 pts** T(n) = T(n/3) + T(something wrong) + O(n)

+ **10 pts** T(n) = T(something wrong) + T(2n/3+4) + O(n)

✓ **+ 15 pts** **Correct answer T(n) = T(n/3)+T(2n/3+4)+O(n), OK with no explanation, OK even if +4 is a different constant**

QUESTION 4

4 Problem 4 0 / 15

✓ **+ 0 pts** **No answer or non-sense**

+ **2 pts** Claims that the algorithm is correct (it's not)

+ **7 pts** Correctly claims that the algorithm does not work, but the counterexample is incorrect (i.e., the algorithm would produce the correct answer on the provided counterexample) or missing

+ **15 pts** Correctly claims that the algorithm does not work, and the counterexample is correct (n=2^q)

QUESTION 5

5 Problem 5 6 / 15

+ **0 pts** Completely wrong or no answer

✓ **+ 4 pts** **Explains how to split the 2D points by finding the median of the x coordinates (sorting if points are not already sorted)**

✓ **- 1 pts** **No mention that the points are assumed to be pre-sorted by x-coordinates so that the split in L and R can be done in linear time**

✓ **+ 3 pts** **Explains searching for the closest pair in L and R recursively, set d1 = closest pair in L, d2 = closest pair in R**

+ **4 pts** Explains building the set of points on the strip [median-d, median+d] where d=min{d1,d2}

+ **4 pts** Explains how to search in the strip in linear-time (for each point on the strip, only the next seven points in y-order need to be considered)

QUESTION 6

6 Problem 6 8 / 15

+ **0 pts** Completely wrong or no answer

+ **1 pts** Algorithm not clear at all

+ **4 pts** Vague description, some ideas

✓ **+ 7 pts** **Correct idea, not explained in sufficient details or important facts missing/incorrect**

+ 11 pts Algorithm correct, explained in sufficient details

✓ **+ 1 pts** **Time complexity analysis addressed, but not correct**

+ 4 pts Time analysis correct and justified (recursion has log k levels, each level requires nk time)

CS 141, Spring 2019 Midterm 1/B

First name: Celyna

Last name: Su

Student ID: 862037643

- This exam is **closed book, closed notes**, 80 minutes long
- Read the questions carefully
- No electronic equipment allowed (cell phones, tablets, computers, ...)
- Write legibly. What can't be read will not be graded
- Use pseudocode, Python, or English to describe your algorithms (no C/C++/Java)
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in CS 141 or CS 14, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your solution
- If you have a question about the meaning of a question, come to the front of the class

81
9
729

Problem 1. (15 points [writing/solving recurrence relations])
Solve exactly (that is, without using any asymptotic notation) the following recurrence relation by iterative substitutions

$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{9}\right) + \sqrt{n} & n > 1 \end{cases}$$

① $T(\frac{n}{9}) + \sqrt{n}$

② $(T(\frac{n}{81}) + \sqrt{\frac{n}{9}}) + \sqrt{n} \rightarrow T(\frac{n}{81}) + \frac{\sqrt{n}}{3} + \sqrt{n}$

③ $\left(T(\frac{n}{729}) + \sqrt{\frac{n}{81}}\right) + \frac{\sqrt{n}}{3} + \sqrt{n} \rightarrow T(\frac{n}{729}) + \frac{\sqrt{n}}{9} + \frac{\sqrt{n}}{3} + \sqrt{n}$

$$T(n) = T\left(\frac{n}{9^i}\right) + \sqrt{n}\left(\frac{1 - 9^{i-\frac{1}{2}}}{1-9}\right)$$

$$T(n) = T\left(\frac{n}{9^{\log_9 n}}\right) + \sqrt{n}\left(\frac{1 - 9^{\log_9 n - \frac{1}{2}}}{-8}\right)$$

$\sqrt{n} + \frac{\sqrt{n}}{9} + \frac{\sqrt{n}}{3} \ldots$

$\frac{n}{9^i} = 1$

$\rightarrow T(1) - \sqrt{n}\left(\frac{1 + 9^{\log_9 n - \frac{1}{2}}}{8}\right)$

$\frac{\sqrt{n}}{3^{i-1}} + \frac{\sqrt{n}}{3^i} + \frac{\sqrt{n}}{3^{i+1}}$

$n = 9^i$

$i = \log_9 n$

$1 - \sqrt{n}\left(\frac{1 + 9^{\log_9 n - \frac{1}{2}}}{8}\right)$

$\frac{\sqrt{n}}{9^i} + \frac{\sqrt{n}}{9^i}$

$\frac{\sqrt{n}}{9^i} \quad \frac{\sqrt{n}}{9^{i-\frac{1}{2}}} \quad \frac{\sqrt{n}}{9^{i-1}}$

$\sqrt{n}\left[\frac{1}{9^i} + \frac{1}{9^{i-\frac{1}{2}}} + \frac{1}{9^{i-1}}\right]$

$\sqrt{n}\left(\frac{1 - 9^{i-\frac{1}{2}}}{1-9}\right)$

Problem 2. (15 points [writing/solving recurrence relations])
Using the Master method, give an asymptotic tight bound for $T(n)$ in the following recurrence relation

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{4}\right) + \sqrt{n}\log n & n > 1 \end{cases}$$

$f(n) < \log_4 2$

$\exists k \geq 0 : f(n) \in \theta(n^{\log_b a} \log^k n)$

$\rightarrow \theta(n^{\log_b a} \log^{k+1} n)$

$\sqrt{n}\log n \in \theta(n^{\log_4 2} \log^k n)$

$\sqrt{n}\log n \in \theta(n^{\frac{1}{2}} \log^k n)$

$\rightarrow \theta(\sqrt{n}\log^k n)$

$k = 1$

$\rightarrow \theta(\sqrt{n}\log^2 n)$

$\sqrt{n}\log n \quad \sqrt{n}^x = a$

$x = \frac{1}{2}$

for $n \geq d$

$af(\frac{n}{b}) \leq \delta f(n)$

$\frac{n}{4} \leq \sqrt{n}\log n$

$\exists \varepsilon < 0 : f(n) \in O(n^{\log_b a - \varepsilon}) \rightarrow \theta(n^{\log_b a})$ $a \leq b$

$\exists k \geq 0 : f(n) \in \theta(n^{\log_b a} \log^k n) \rightarrow \theta(n^{\log_b a} \log^{k+1} n)$ $a = b$

$\exists \varepsilon > 0 : f(n) \in \Omega(n^{\log_b a + \varepsilon}) \rightarrow \theta(f(n))$ $a \geq b$

$af(\frac{n}{b}) \leq \delta f(n)$ for $n \geq d$

Problem 3. (15 points [writing/solving recurrence relations])

In the algorithm SELECT described in class (linear-time selection), the input elements are divided into $\lceil n/5 \rceil$ groups of 5. Suppose you modify the algorithm to divide the input elements into $\lceil n/3 \rceil$ groups of 3 instead. Let $T(n)$ denote the worst-case running time of the modified algorithm as a function of the input size n. Write a recurrence relation for $T(n)$, but do NOT solve it.

$\frac{n}{5} \rightarrow \frac{n}{3}$

$T(n) = T\left[\frac{n}{3}\right] + T\left(\frac{2n}{3} + 4\right) + O(n)$

$T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$ $\quad O(n)$

linear select

Problem 4. (20 points [divide & conquer])

An array A is said to have a *majority element* if more than half of the entries in A are exactly the same. Given an unsorted array $A[1 \ldots n]$ of n items (where n is a power of two) we want to determine whether A has a majority element, and if so, return such an item. Consider the following divide and conquer algorithm for this problem.

Algorithm FIND-MAJORITY (A : **array**)
1 $n \leftarrow |A|$
2 **if** $n \leq 1$ **then return** $A[1]$
3 **else**
4 $a_1 \leftarrow$ FIND-MAJORITY$(A[1, \ldots, n/2])$
5 $a_2 \leftarrow$ FIND-MAJORITY$(A[n/2+1, \ldots, n])$
6 **if** the number of times a_1 appears in A is $\geq n/2$ **then return** a_1
7 **else if** the number of times a_2 appears in A is $\geq n/2$ **then return** a_2
8 **else return** NULL

Is this algorithm correct i.e., does FIND-MAJORITY always return the majority element, if A has one in it (or NULL otherwise)? Give a counterexample if your answer is "No", a brief argument of correctness (e.g., a proof by induction on n, the size of A) if your answer is "Yes".

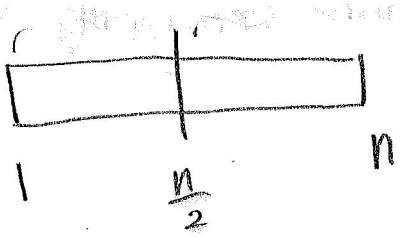
half or

if n is 0, you can't return A[1] because it's empty (should be null)

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

$O(\log n)$

A [1 ... n]



n ← |A|

If n = 0 then return NULL

If n ≤ 1 then return A[1]

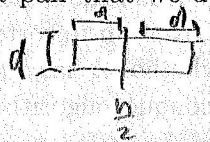
else

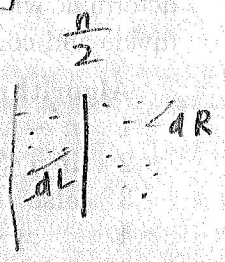
a_1 ← Find Majority (A[1, ..., n/2])

a_2 ← Find Majority (A[n/2 + 1, ... n])

Problem 5. (15 points [divide & conquer])

Write the pseudo-code for the $O(n \log n)$-time algorithm for closest pair that we described in class.





① divide the array in half

② set the left side (L) a from 1 to $\frac{n}{2}$

③ set the right side (R) from $\frac{n}{2}$ to n

④ Recursively find the closest pair of L and R

⑤ Split on side w/ from lower pair

⑥ continue to split until the d x d unit blocks have been reached

⑦ check ea. column recursively to find closest pair

Problem 6. (20 points [divide & conquer])

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Describe a divide-and-conquer algorithm that takes $O(kn \log k)$ time. Make sure you explain why your algorithm runs in $O(kn \log k)$ time.

Hint: Use as a sub-routine ("black-box") the MERGE algorithm we described in class to merge two sorted arrays with n elements in $O(n)$-time.

$O(n \log n) \rightarrow$ mergesort
max subsequence
closest pair

① Merge the k sorted arrays into one

② Recursively divide in half until you have 1 element $\rightarrow O(\log k)$

③ While merging upwards, compare left and right elements of all the subarrays until you end back with 1 array. (this is $O(kn)$)

this algorithm is $O(kn \log k)$ time because comparing the left and right elements of the subarrays and merging them is $O(kn)$ time, and the second step of recursively dividing in half is $O(\log k)$.