CS 141 homework3

Celyna Su

TOTAL POINTS

98.5 / 100

QUESTION 1

1 Problem 1 25 / 25

✓ **- 0 pts Correct**

- 3 pts Unclear explanation for how we get the range [5,8].

- 5 pts Unclear explanation.

- 25 pts No solution

- 2 pts How directly jumped to p = 8 or p = 9 ?

- 3 pts No explanation why jumped from p = 5 to p = 8.

- 2 pts Incorrect range

- 10 pts Incorrect

- 0.5 pts Did not specify the range

QUESTION 2

2 Problem 2 25 / 25

✓ **+ 10 pts Correct D&C algorithm**

✓ **+ 10 pts Correct recurrence relations**

✓ **+ 5 pts Correct analysis the time complexity**

+ 0 pts Incorrect answer.

- 3 pts Didn't indicate that split the polynomial into two n/2 DEGREE polynomials.

QUESTION 3

3 Problem 3 25 / 25

- 3 pts Running time of Algorithm not shown

- 15 pts Algorithm is incorrect

✓ **- 0 pts Correct**

QUESTION 4

4 Problem 4 23.5 / 25

- 0 pts Correct

- 12.5 pts No solution for a) part

- 12.5 pts No solution for b) part

- 1 pts Incorrect T(n) for a) part

✓ **- 1 pts Incorrect T(n) for b) part**

- 3 pts No proof for a) part

- 3 pts No proof for b) part

- 2 pts Unclear/Incorrect/Incomplete proof for a) part

- 2 pts Unclear/Incorrect/Incomplete proof for b) part

- 0.5 Point adjustment

What is B in part a) ?

CS 141, Spring 2019 Homework 3

Posted: April 18th, 2019 Due: April 25th, 2019

Name: Celyna Su

Student ID #: 862037643

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted

Problem 1. (25 points)

A CS 141 student has been trying to speed-up Karatsuba's divide-and-conquer integer multiplication algorithm. Given two numbers x, y with n bits each, her algorithm (1) first divides both x and y into four equal-length pieces, then (2) expresses the product $x \cdot y$ using p multiplications of these $n/4$-bit pieces, followed by a constant number of additions, subtractions and shifts. How small p needs to be in order to give a faster algorithm than the Karatsuba's algorithm covered in class? You can assume n to be a power of 4, and $p > 4$. Justify your answer.

Answer:

The recurrence relation for the above student's algorithm is $T(n) = pT(n/4) + dn$.

The only way for this student's algorithm to be faster than Karatsuba's is for the exponential value of $O(n^{log_4 p})$ to be less than $O(n^{log_2 3})$. Using the Master's Theorem, and the fact that we were given $p > 4$, we know that $a = p$ and $b = 4$.

We want the program to be faster than $O(n^{log_2 3})$, or $O(n^1.59)$, so:

$\log_4 p < \log_2 3$

$\frac{\log p}{\log 4} < \frac{\log 3}{\log 2}$

$\log p < \frac{\log 4}{\log 2} \cdot \log 3$

$\log p < 2 \log 3$

$\log p < \log 9$

$4 < p < 9$

In conclusion, p must be between the 4 and 9 to speed-up Karatsuba's divide-and-conquer algorithm.

Problem 2. (25 points)

Give a divide-and-conquer algorithm ("Karatsuba-like") for multiplying two polynomials of degree n in time $O(n^{\log_2 3})$.

Answer:

Let one polynomial, call it $A(x)$, be $A(x) = a_0 + a_1 + a_2x + a_xx^2 + ... + a_{n-1}x^{n-1}$ and the second polynomial, called $B(x)$, be $B(x) = b_0 + b_1 + b_2x + ... + b_{n-1}x^{n-1}$. Assume that $n = 2^j$. Split $A(x)$ and $B(x)$ into two polynomials:
$A(x) = A_0(x) + x^{n/2}A_1(x)$ and
$B(x) = B_0(x) + x^{n/2}B_1(x)$ where

$A_0(x) = a_0 + a_1x + a_2x^2 + ... + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}$
$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + a_{\frac{n}{2}+2}x^2 + ... + a_{n-1}x^{\frac{n}{2}-1}$
$B_0(x) = b_0 + b_1x + b_2x^2 + ... + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1}$
$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1}x + b_{\frac{n}{2}+2}x^2 + ... + b_{n-1}x^{\frac{n}{2}-1}$

$A(x) \cdot B(x)$ can be simplified if we multiplied the above as follows:
$AB = (A_0 + x^{\frac{n}{2}}A_1)(B_0 + x^{\frac{n}{2}}B_1)$
$= A_0B_0 + x^{\frac{n}{2}}(A_0B_1 + A_1B_0) + x^nA_1B_1$
$= A_0B_0 + x^{\frac{n}{2}}((A_0 - A_1)(B_1 - B_0) + A_0B_0 + A_1B_1) + x^2A_1B_1$

The recurrence relation is

$$T(n) = \begin{cases} \Theta(1), & n = 1 \quad (1) \\ 3T(\frac{n}{2}) + O(n), & n > 1 \quad (2) \end{cases}$$

Using Master's Theorem, we know that $a = 3$, $b = 2$ and because $a > b^d$, we can conclude that $T(n) \in O(n^{log_2 3})$

Problem 3. (25 points)

Describe and analyze an algorithm that takes an unsorted array A of n integers (in an unbounded range) and an integer k, and divides A into k equal-sized groups, such that the integers in the first group are lower than the integers in the second group, and the integers in the second group are lower than the integers in the third group, and so on. For instance if $A = \{4, 12, 3, 8, 7, 9, 10, 20, 5\}$ and $k = 3$, one possible solution would be $A_1 = \{4, 3, 5\}, A_2 = \{8, 7, 9\}, A_3 = \{12, 10, 20\}$. Sorting A in $O(n \log n)$-time would solve the problem, but we want a faster solution. The running time of your solution should be bounded by $O(nk)$. For simplicity, you can assume that is n a multiple of k, and that all the elements are distinct. **Note:** k is an input to the algorithm, not a fixed constant.

Answer:

1. Use the linear-select algorithm to find the $\frac{n}{k}$ smallest integer, denoted as x.x will be the largest integer in the first group.
2. Split the array into integers around x, meaning grouping integers that are less and greater than x. The first group, which includes all the integers less than x and x, consists of $\frac{n}{k}$ elements.
3. We then repeat the process for the rest of the array that is not yet in groups of $k - 1$. This step will create k number of groups, all sized $\frac{n}{k}$.

According to the slides, grouping into k groups will have a runtime of $O(nk)$. There are also at most n number of comparisons for dividing the array into groups sized $\frac{n}{k}$. Thus, for each of these groups, we get a linear runtime. In conclusion, the overall runtime would be $O(nk)$.

Problem 4. (25 points)

In the algorithm SELECT described in class (linear-time selection), the input elements are divided into $n/5$ groups of 5.

1. Suppose you modify the algorithm to divide the input elements into $n/7$ groups of 7 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 $T(n)$, then give a proof that $T(n) \in O(n)$.

2. Suppose you modify the algorithm to divide the input elements into $n/3$ 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 $T(n)$, then provide an argument that $T(n)$ is not $O(n)$.

Answer:
(1.) $4([\frac{1}{2}[\frac{n}{7}]] - 2) = \frac{2n}{7} - 8$
$\text{MAX}(|L|, |R|) \leq n - (\frac{2n}{7} - B) = \frac{5n}{7}$
$T(n) = T(\frac{n}{7}) + T(\frac{5n}{7} + B) + O(n)$

Proof:
Base Case: $T(7) = T(1) + T(13) + O(n)$, which is runtime of $O(n)$

$T(n) = T(\frac{n}{7}) + T(\frac{5n}{7} + B) + dn$
$= c(\frac{n}{7} + 1) + c(\frac{5n}{7} + B) + dn$
$= \frac{cn}{7} + c + \frac{5cn}{7} + 8c + dn$
$= \frac{6cn}{7} + c + 9c + dn$
$= cn - [\frac{cn}{7} - 9c - dn]$

To prove O(n) is bounded by cn, prove that $[\frac{cn}{7} - 9c - dn] \leq cn$:
$cn - c(\frac{n}{7} - 9) - dn]$
$\frac{cn}{7} - 9c - dn \geq 0$
$\frac{cn}{7} - \frac{63c}{7} \geq dn$
$\frac{c(n-63)}{7} \geq dn$
$c \geq \frac{7dn}{n-63} \geq 0$
$cn \geq \frac{cn}{7} - 9c - dn$
Whatever d is, we can choose c accordingly, so this means the recurrence is bound by $O(n)$.

(2.) $1(\frac{1}{2} \cdot \frac{n}{3}) - 2 = \frac{n}{6} - 2$
$|R| \geq \frac{n}{6} - 2$ and $|L| \leq n - \frac{n}{6} - 2 = \frac{5n}{6} + 2$
$T(n) = T(\frac{n}{3}) + T(\frac{5n}{6} + 2) + O(n)$

Proof:
We break it into a tree, with n on top and two children, $\frac{2n}{6}$ and $\frac{5n}{6}$, derived from the recurrence relation. By adding the two children together, we get $\frac{7}{6} \geq n$ which proves that all subsequent groups will also be more than or exactly n, so $\notin O(n)$. It is $O(n \log n)$.