CS 141, Spring 2019 Homework 3

Problem 1. (35 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.

Solution: The divide and conquer algorithm has the following recurrence relation:

$$T(n) = p \cdot T(n/4) + O(n)$$

We have to find the biggest possible p such that this algorithm is asymptotically better than $O(n^{\log_2 3})$. Since we have it follows that

$$T(n) \in O(n^{\log_4 p})$$

We need a p such that

$$\log_4 p \ < \ \log_2 3$$

Since $\log_2 3 = \log_4 3 / \log_4 2 = 2 \log_4 3 = \log_4 9$, this is equivalent to $p < 9$. So to get a faster algorithm, p must be not bigger than 8.

Problem 2. (25 points)

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

Answer: Suppose the two polynomials we want to multiply are $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$ and $B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$. We assume that n is a power of two (otherwise, we can always pad the coefficients with zeros to reach the "next" power of two). Let us break $A(x)$ and $B(x)$ into two polynomials as follows.

$$\begin{aligned} A(x) &= A_0(x) + x^{n/2} A_1(x) \\ B(x) &= B_0(x) + x^{n/2} B_1(x) \end{aligned}$$

where

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_{n/2-1} x^{n/2-1} \\ A_1(x) &= a_{n/2} + a_{n/2+1} x + a_{n/2+2} x^2 + \ldots + a_{n-1} x^{n/2-1} \\ B_0(x) &= b_0 + b_1 x + b_2 x^2 + \ldots + b_{n/2-1} x^{n/2-1} \\ B_1(x) &= b_{n/2} + b_{n/2+1} x + b_{n/2+2} x^2 + \ldots + b_{n-1} x^{n/2-1} \end{aligned}$$

Then the problem of multiplying $A(x)B(x)$ can be decomposed in the problem of multiplying $A_0(x), A_1(x), B_1(x), B_1(x)$ as follows. We omit "(x)" to reduce the clutter.

$$\begin{aligned} AB &= (A_0 + x^{n/2}A_1)(B_0 + x^{n/2}B_1) \\ &= A_0B_0 + x^{n/2}(A_0B_1 + A_1B_0) + x^nA_1B_1 \\ &= A_0B_0 + x^{n/2}((A_0 - A_1)(B_1 - B_0) + A_0B_0 + A_1B_1) + x^nA_1B_1 \end{aligned}$$

Therefore, we need 3 multiplications of two polynomials of degree $n/2$ (namely, A_0B_0, A_1B_1 and $(A_0 - A_1)(B_1 - B_0)$) and $O(n)$ additional work for the sum and the differences.

The recurrence relations is

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

which can be solved using the Master Theorem, concluding 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 (however, the integers inside each group do not need to be sorted). 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 distict. **Note:** k is an input to the algorithm, not a fixed constant.

Answer: Our algorithm first uses linear-time SELECT to find the n/k-th smallest element $s_{n/k}$, then scan the original array and PARTITION it into two groups: (1) the elements smaller or equal to $s_{n/k}$ and (2) the elements bigger that $s_{n/k}$. At this point, the first group contains the smallest n/k elements of the array, which is the bottom group A_1. Then we use SELECT to find the n/k-th order statistic of the remainder of the array, and partitions around it, etc., until all the groups have been separated. Each SELECT and PARTITION requires linear time in the number of remaining elements, which is at most n, so the total running time is $O(nk)$.

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: For groups of 7, the number of elements greater than x (and the number of element less than x) is at least

$$4\left(\lceil\frac{1}{2}\lceil\frac{n}{7}\rceil\rceil-2\right)\geq\frac{2n}{7}-8$$

and the recurrence relation becomes

$$T_7(n)=\begin{cases}\Theta(1) & n<70\\ T_7\left(\lceil n/7\rceil\right)+T_7(5n/7+8)+O(n) & n\geq 70\end{cases}$$

We prove by induction that $T_7(n)\in O(n)$.

$$\begin{aligned}T_7(n) &= T_7\left(\lceil n/7\rceil\right)+T_7(5n/7+8)+O(n)\\ &\leq c\lceil n/7\rceil+c(5n/7+8)+O(n)\\ &= cn-[c(n/7-9)-dn]\\ &\leq cn\end{aligned}$$

The last inequality holds because $n\geq 70$, therefore $n/7-9>0$, and because we have the freedom to choose any c that would make $c(n/7-9)-dn$ positive.

For groups of 3, the number of elements greater than x (and the number of element less than x) is at least

$$2\left(\lceil\frac{1}{2}\lceil\frac{n}{3}\rceil\rceil-2\right)\geq\frac{n}{3}-4$$

and the recurrence relation becomes

$$T_3(n)\leq T_3\left(\lceil n/3\rceil\right)+T_3(2n/3+4)+O(n)$$

If we ignore the ceilings for simplicity, we get

$$T_3(n)\leq T_3(n/3)+T_3(2n/3+4)+O(n)$$

Now consider the recursion tree: in going from level i to level $i+1$, each problem of a given size s is divided into two problems, of size $s/3$ and $2s/3$, respectively (unless s is small so that a base case happens). Thus, the sum of the problem sizes in level $i+1$ is at most the sum of the problem sizes in level i. Inductively, in every level, the sum of the problem sizes is at most n. Since the work done for each subproblem is proportional to the size, for each level, the total work done within that level is $O(n)$. Since there are at most $\log_{3/2} n$ levels, the total work over all levels is $O(n\log n)$. In fact, only group of odd size ≥ 5 make the algorithm work in linear time.

The answer above suffices for full credit. Dealing with the ceilings precisely doesn't change the final answer, but makes the analysis much more difficult. Here are some notes (written by Prof. Neal Young) on that, for completeness.

The problem size at level 0 is n, but the sum of the problem sizes at level 1 is $\lceil n/3\rceil+\lceil 2n/3\rceil$, which could be as large as $n+2$. Generally, in going from level i to level $i+1$, we can add one extra to the sum of the problem sizes for each node in level $i+1$. We can prove by induction that the sum of the problem sizes in level i is at most n *plus* the number of nodes in the tree in levels $0,1,\ldots,i$. Thus, the total work is $O((n+m)\log n)$, where m is the number of nodes in the tree. To finish, we would need to show that $m=O(n)$. It's easy to see that there are at most 2^i nodes

in level i, and at most $\log_{3/2} n$ levels, but this bound is too loose, as it would allow more than $2^{\log_{3/2} n} = n^{\log_{3/2} 2} \gg n$ nodes altogether.

Instead, we'll ditch the recursion-tree method and use induction. We'll prove inductively that $T(n) \le cn \log_2 n$ for some large enough constant c.

The inequality will hold for all small n as long as c is large enough, so in the inductive proof we can assume as base cases that the bound holds for all n below any constant that we choose.

For the inductive step, using $\log_2(z+1) = \log_2(z) + \int_z^{z+1} \frac{dz}{z} \le \log_2(z) + 1/z$, we have

$$\begin{aligned}
T(n) &\le T(\lceil n/3 \rceil) + T(\lceil 2n/3 \rceil) + an \\
&\le c\lceil n/3 \rceil \log_2(\lceil n/3 \rceil) + c\lceil 2n/3 \rceil \log_2(\lceil 2n/3 \rceil) + an \\
&\le c(n/3+1) \log_2(n/3+1) + c(2n/3+1) \log_2(2n/3+1) + an \\
&= c(n/3) \log_2(n/3+1) + c(2n/3) \log_2(2n/3+1) + an + O(c \log n) \\
&\le c(n/3)[\log_2(n/3) + O(1/n)] + c(2n/3)[\log_2(2n/3) + O(1/n)] + O(c \log n) + an \\
&= c(n/3) \log_2(n/3) + c(2n/3) \log_2(2n/3) + O(c) + O(c \log n) + an \\
&= c(n/3)[\log_2 n - \log_2 3] + c(2n/3)[\log_2 n - \log_2(3/2)] + O(c) + O(c \log n) + an \\
&= cn \log_2 n - cn[(1/3) \log_2 3 + (2/3) \log_2(3/2)] + O(c) + O(c \log n) + an \\
&\ge cn \log_2 n - 0.01cn + O(c) + O(c \log n) + an \\
&\ge cn \log_2 n.
\end{aligned}$$

(The last inequality holds for c and n sufficiently large.)