Assignement 2, COSC 3P03, Algorithms, Winter, 2016

Due: Feb. 10, Wed, 5:00 PM.

1. (10) The running time of an algorithm A is described by the recurrence $t(n) = 7t(n/2) + n^2$. A competing algorithm A' has a running time of $T(n) = aT(n/4) + n^2$. What is the largest integer value for a such that A' is asymptotically faster than A?

   For algorithm A, by the masters method,

   $$t(n) = O(n^{log_2 7}).$$

   For A', we can assume that $a > 16$ since if $a = 16$, by masters method,

   $$T(n) = \Theta(n^2 \log n),$$

   which is already better than $t(n)$. For $a > 16$, since $n^2 = O(n^{\log_4 a - \epsilon})$ for some $\epsilon > 0$, the solution for $T(n)$ is (Case 1)

   $$T(n) = O(n^{\log_4 a}).$$

   To find the largest a such that $\log_4 a < \log_2 7$, namely, $\log_2 a / \log_2 4 < \log_2 7$, we have $\log_2 a < \log_2 7^2$, i.e., $a < 49$, therefore, $a = 48$.

2. (40) For Fibonacci numbers defined as follows:

   $$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2),\ n \geq 2 \end{aligned}$$

   (a) Prove by induction that for any $n \geq 0$,

   $$\begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

   Proof: When $n = 0$, by definition. $f(0) = 0$ and $f(1) = 1$, we have

   $$\begin{pmatrix} f(0) \\ f(1) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^0 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

   Assume that

   $$\begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

   Then for $n + 1$,

   $$\begin{aligned} \begin{pmatrix} f(n+1) \\ f(n+2) \end{pmatrix} &= \begin{pmatrix} f(n+1) \\ f(n) + f(n+1) \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} f(n) \\ f(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^{n+1} \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned}$$

   (b) Using the recursive definition of $f(n)$, design and implement a recursive algorithm that computes $f(n)$; Then using the just proved formula to design and implement a recusive algorithm that computes $f(n)$ in $O(\log n)$ time. Compare the running times (real or asymptotic) of these two algorithms.

Recursive algorithm $A(n)$ that given n, returns $f(n)$:

$A(n)$

if $n \leq 1$

    return n

else

    return $(A(n-1) + A(n-2))$

with running time $O((\frac{1+\sqrt{5}}{2})^n)$.

A more efficient algorithm can be designed as follows: We first compute

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$$

which takes $O(\log n)$ time (see the 3rd algorithm from Q3 of A1). We then multiply the resulting 2×2 matrix with vector $\binom{0}{1}$.

3. (10) Given n arbitrary numbers, how can you use our linear-time selection algorithm to find the k smallest numbers in $O(n)$ time? For example, if we have 4, 3, 3, 9, 10, 2, 3, then the two smallest numbers are 2 and 3 (or 3 and 2. Your answer does not have to be sorted). Similarly, the four smallest numbers are 3, 3, 2, 3.

   Since the input contains arbitrary numbers, some elements could appear multiple times, which makes the problem non-trivial. Otherwise, one can simply do the following:

   - select the kth smallest element A and output A;
   - scan the input: if $x_i < A$, output x_i

   The problem with this approach when the input contains arbitrary numbers is that it could output more than k elements. Here is a correct algorithm that output exactly k smallest elements.

   - select the kth smallest element A ($O(n)$ time);
   - scan the input to find elements $\leq A$ such that

     S_1: all elements $< A$
     S_2: all elements $= A$ (note that $|S_1| + |S_2| \geq k$)
   - put $k - |S_1|$ copies of A into S_1. Now $|S_1| = k$ and S_1 contains k smallest elements.

4. (10) Will we still have a linear selection algorithm if elements are grouped into groups of 7? Prove your answer. What about 3?

   If the group size is 7, the recurrence becomes

   $$t(n) = t(n/7) + t(3n/4) + c'n$$

   which has a solution $t(n) = O(n)$ (since $1/7+3/4 < 1$). So yes, we will still have a linear-time selction algorithm if the group size is 7

   When the group size is 3, we would have

   $$t(n) = t(n/3) + t(2n/3) + n.$$

   which has a solution $t(n) = \Omega(n \log n)$ (since $1/3 + 2/3 = 1$).

5. (10) Quicksort has a worst case running time of $O(n^2)$ and a best and average case running time $O(n \log n)$. How can you modify the quicksort so that its worst case running time is $O(n \log n)$?

   Use the linear selection algorithm to find the median and use the median to partition. The running time is

   $$\begin{aligned} t(n) &= 2t(n/2) + cn \\ &= O(n \log n), \end{aligned}$$

   where cn is the time used to find the median and to partition.

6. (20) (a) What is the largest k such that if you can multiply 3×3 matrices using k multiplications, then you can multiply $n \times n$ matrices in time $o(n^{\log 7})$? What would the running time of this algorithm be? You can assume that n is a power of 3.

   Let k be the number of multiplications required for a certain algorithm. The recursive algorithm has a time $t(n) = kt(n/3) + cn^2$, which has a solution $O(n^{\log_3 k})$. Traditionally, $k = 27$, resulting in an $O(n^3)$ algorithm.

   If we want an $o(n^3)$ algorithm, we need to have $\log_3 k < \log 7$, implying that the largest k is 21.

   (b) V. Pan has discovered a way of multiplying 68×68 matrices using 132,464 multiplications, a way of multiplying 70×70 matrices using 143,640 multiplications, and a way of multiplying 72×72 matrices using 155,424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?

   Similar to Q2, we have (using Maple evalf(log[a](X))):

$$\log_{68} 132464 = 2.795128488$$
$$\log_{70} 143640 = 2.795122690$$
$$\log_{72} 155424 = 2.795147392$$

   while $\log_2 7 = 2.807354922$. So they are all better than the Strassen's, with the second one being the best (70 by 70 case).