MCS 401 – Computer Algorithms I

Fall 2023

Problem Set 3

Lev Reyzin

Due: 10/13/23 by the beginning of class

1. [10 pts] Consider the problem of taking positive integers x and n and computing x^n using as few multiplications as possible. Here, we are not concerned with the cost of performing the multiplications, but only with how many multiplications are done. Naively, $O(n)$ suffice by starting with 1 and multiplying x with the result n times. But observe that

$$x^n = \left\{ \begin{array}{ll} (x^{n/2})(x^{n/2}), & \text{for } n \text{ even} \\ (x^{(n-1)/2})(x^{(n-1)/2})x, & \text{for } n \text{ odd} \end{array} \right\}.$$

Use this observation to come up with a divide and conquer approach to computing x^n that uses asymptotically fewer multiplications than linear in n.

2. [10 pts] Consider multiplying two n by n matrices A and B to get an n by n matrix $C = AB$. Each entry of C for $1 \le i, j \le n$ is defined as $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$.

- **a**. What is the running time, as a function of n, to compute matrix $C = AB$ when A and B are given as input using the formula above? Justify your answer.

A different approach uses divide and conquer to split all these matrices into four equally sized $n/2$ by $n/2$ contiguous blocks (known as "block matrices") as follows:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, \quad C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix},$$

and recursively computes the blocks of C as

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}.$$

- **b**. Give the recurrence resulting from this approach and solve the recurrence explicitly.[1]

An improvement to the approach above was given by Strassen, who observed that one can compute 7 matrices $M_1 \dots M_7$, properly defined[2], using 1 matrix multiplication of two $n/2 \times n/2$ matrices

[1] Adding/subtracting two n by n matrices trivially takes $O(n^2)$ time. ($C = A + B$ has $c_{i,j} = a_{i,j} + b_{i,j}$.)

[2] This is done as: $M_1 = (A_{11}+A_{22})(B_{11}+B_{22})$, $M_2 = (A_{21}+A_{22})B_{11}$, $M_3 = A_{11}(B_{12}-B_{22})$, $M_4 = A_{22}(B_{21}-B_{11})$, $M_5 = (A_{11}+A_{12})B_{22}$, $M_6 = (A_{21}-A_{11})(B_{11}+B_{12})$, $M_7 = (A_{12}-A_{22})(B_{21}+B_{22})$, but worrying about these details is not important for solving this problem.

for each of the M_is. It turns out that C can be computed as:

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix},$$

again with the multiplications to produce $M_1 \ldots M_7$ also done recursively.

c. Give the recurrence resulting from this improved approach and solve it explicitly.

3. [10 pts] You are given a one dimensional array that may contain both positive and negative integers. Give an $O(n \log n)$ algorithm to find the sum of contiguous (ie. next to one another) subarray of numbers which has the largest sum. For example, if the given array is $[-2, -5, \mathbf{6}, \mathbf{-2}, \mathbf{-3}, \mathbf{1}, \mathbf{5}, -6]$, then the maximum subarray sum is 7 (the subarray is marked in boldface). Argue that your algorithm is correct.

4. [10 pts] You are given two arrays, A and B, each of which contains n integers. The elements in each array are guaranteed to already be in sorted order in the input, i.e.

$$A[0] \le A[1] \le \ldots \le A[n-1] \text{ and also } B[0] \le B[1] \le \ldots \le B[n-1].$$

Give as fast an algorithm as you can for finding the *median* value of all the $2n$ numbers in both A and B. (We define the median of $2n$ numbers to be the average of the nth smallest and nth largest values.) Argue that your algorithm is correct and give its running time.

5. [10 pts] You are given an array X of n elements. A majority element of X is any element occurring in more than $n/2$ positions. The only access you have to the array is to compare any two of its elements for equality; hence you cannot sort the array, nor add up its values, etc. Design an $O(n \log n)$ divide-and-conquer algorithm to find a majority element in X (or determine that no majority element exists).

6. [10 pts] You are given a $2^k \times 2^k$ board with one missing cell. (See Figure 1 below.)

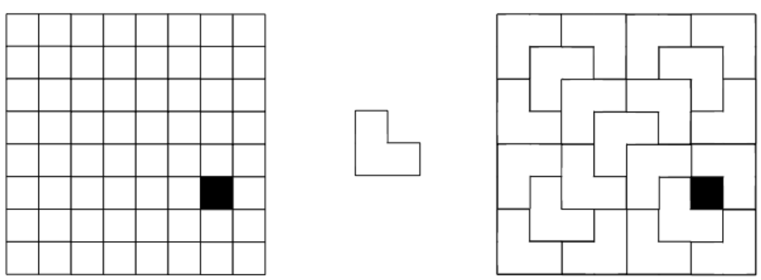

Figure 1: On the left is an example grid with a missing cell, with $k = 3$. In the middle is the "L-shaped" tile, to be used for tiling. On the right is an example solution.

Give an $O(2^{2k})$-time algorithm for filling the board with "L-shaped" tiles.