

Notes: *On homework assignments, you are allowed to discuss the questions with a small number of other people in the course. However, the emphasis of such discussions should be obtaining a solid understanding of the solutions to the assigned problem. Thus, you must destroy any notes from your discussions, and then write up the solutions on your own. For each problem, you must also list anyone you discussed that problem with (even briefly). You also must describe any other references you used.*

The homeworks are due at the beginning of class on the due date. Late submissions will be accepted only with special permission. Also, please take the time to write clear and concise answers. Credit will be reduced if answers are unclear or long winded.

All questions count for the same amount of credit, although some are harder than others. Some of the questions may require quite a bit of thought, so I encourage you to start early.

1. (Based on [CLRS], Problem 2-4.) Given a sequence of n distinct numbers $A = a_1, a_2, \dots, a_n$, we say that a_i and a_j are inverted if $i < j$ but $a_i > a_j$. The number of inversions in the sequence A , denoted $\nu(A)$, is the number of pairs a_i and a_j that are inverted. $\nu(A)$ provides a measure of how close A is to being sorted in increasing order. For example, if A is already sorted in increasing order, then $\nu(A)$ is 0. At the other extreme, if A is sorted in decreasing order, every pair is inverted, and thus $\nu(A)$ is $\binom{n}{2}$.

Describe a divide and conquer algorithm that finds $\nu(A)$ in time $\Theta(n \log n)$. You should assume that the sequence A is given to us in an array such that we can test the value of a_i in unit time, for any i . Note that the obvious algorithm is to test all pairs a_i and a_j ; that algorithm runs in $\Theta(n^2)$ time.

Hint: you should modify merge sort to also compute $\nu(A)$.

2. In this question, we shall obtain a more exact bound on the running time of matrix multiplication, and use this to determine at what value of n Strassen's algorithm starts to outperform the naïve matrix multiplication algorithm. In order to do so, we shall use the fact that the solution to the recurrence relation

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + cn^\alpha \quad (n > 1); \\ T(1) &= d \end{aligned}$$

is

$$T(n) = n^\beta \left(d + \frac{cb^\alpha}{a - b^\alpha} \right) - n^\alpha \left(\frac{cb^\alpha}{a - b^\alpha} \right),$$

where $\beta = \log_b a$.

- a) Let $S(n)$ be the running time of Strassen's algorithm for multiplying two $n \times n$ matrices, where the entries in the matrix are integers, adding two integers requires 1 time unit, and multiplying two integers requires m time units.

Give an exact recurrence relation (i.e., without any Θ or O expressions) for $S(n)$. Use the formula above to derive an explicit expression for $S(n)$ in terms of m and n . You should assume that n is a power of 2.

- b) Let $N(n)$ be the running time of the naïve matrix multiplication algorithm in the same model. Give an exact recurrence relation for $N(n)$, and derive an explicit expression for $N(n)$ in terms of m and n .

- c) For the case where $m = 10$, determine n_0 , the smallest power of 2 such that $S(n) \leq N(n)$ for all $n > n_0$. Do the same for the case where $m = 1$.
3. Say we are given a set of n non-vertical lines in the plane, where ℓ_i , the i^{th} line, is described by real numbers a_i and b_i such that ℓ_i consists of the points (x, y) such that $y = a_i x + b_i$. A line ℓ *dominates* at t , if the y -coordinate of ℓ is larger at $x = t$ than the y -coordinate of any other line at $x = t$. Intuitively, line ℓ is the first line encountered when moving from $y = +\infty$ to $y = -\infty$ along the vertical line $x = t$. We say that line ℓ is *visible* if there is some t , such that ℓ dominates at t .
- Describe an algorithm that takes as input n lines, and outputs a list of which lines are visible. Argue that your algorithm returns the correct answer and runs in time $\Theta(n \log n)$. You can assume that no three of the lines all meet at a single point.
4. In lecture, we saw a divide and conquer algorithm to multiply A and B , two n -bit integers, using $\Theta(n^{\log_2 3})$ bit operations. To do so, we reduced the multiplication problem to three multiplication sub-problems, each of size $\frac{n}{2}$.

- (a) Show that we can reduce the the multiplication problem to **five** multiplication sub-problems, each of size approximately $\frac{n}{3}$, rather than the nine sub-problems required by the obvious divide and conquer algorithm that breaks the problem up into integers of size $\frac{n}{3}$. Show that the resulting algorithm requires $o(n^{\log_2 3})$ bit operations.

Hint: while a fairly straightforward extension of the technique shown in class results in six sub-problems, it is more difficult to obtain the five required here. The idea is to write the integer A as $A_1 2^{2n/3} + A_2 2^{n/3} + A_3$, and the integer B as $B_1 2^{2n/3} + B_2 2^{n/3} + B_3$. Then, let $A(x)$ be the polynomial $A_1 x^2 + A_2 x + A_3$ and let $B(x)$ be the polynomial $B_1 x^2 + B_2 x + B_3$. If you knew the coefficients of the polynomial $C(x) = A(x) \cdot B(x)$, then it would be fairly easy to compute the value $C = A \cdot B$. Keep in mind that for this problem, the running time is defined as the number of bit operations required.

- (b) Generalize the algorithm from part (a) to show that for any integer $k \geq 1$ the multiplication of two n -bit integers can be reduced to $2k - 1$ multiplication sub-problems, each of size approximately $\frac{n}{k}$.
- (c) Use part (b) to show that for every real number $\alpha > 1$, there exists an algorithm \mathcal{A}_α that can multiply two n -bit integers using $O(n^\alpha)$ bit operations.