

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), and 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 very 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. ([CLR] problem 32-1)

- a) Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications.
- b) Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound n that run in time $\Theta(n^{\log_3 3})$. The first should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is odd or even.

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. a) Consider two vectors $\mathbf{a} = \{a_0, a_1, \dots, a_{n-1}\}$ and $\mathbf{b} = \{a_0, a_1, \dots, a_{n-1}\}$. The function $\hat{\otimes}$ is defined as $\mathbf{c} = \mathbf{a} \hat{\otimes} \mathbf{b}$, where $\mathbf{c} = \{c_0, c_1, \dots, c_{n-1}\}$, such that

$$c_i = \sum_{j=0}^{n-1} a_j b_{i-j \bmod n}.$$

For example, $c_0 = a_0 b_0 + a_1 b_{n-1} + \dots + a_{n-1} b_1$. Demonstrate that $\mathbf{a} \hat{\otimes} \mathbf{b} = \text{FFT}_n^{-1}(\text{FFT}_n(\mathbf{a}) \cdot \text{FFT}_n(\mathbf{b}))$, where $\text{FFT}_n(\mathbf{a})$ denotes the vector $\mathbf{d} = \{d_0, d_1, \dots, d_{n-1}\}$, where $d_i = a_0 + a_1 \omega_n^i + a_2 \omega_n^{2i} \dots + a_{n-1} \omega_n^{(n-1)i}$.

b) A *left-shift* matrix is an $n \times n$ matrix where each row is identical to the previous row shifted one place to the left (with wrap-around). Thus, for example, the following is a 3×3 left-shift matrix:

$$\begin{pmatrix} 12 & 3 & 5 \\ 3 & 5 & 12 \\ 5 & 12 & 3 \end{pmatrix}$$

Design an algorithm that multiplies two $n \times n$ left-shift matrices in time $O(n \log n)$. Note that you can represent an $n \times n$ left-shift matrix as a vector of length n by describing the first row of the matrix.

4. In this question, we study the following optimization problem, which we shall refer to as the maximum vertex weight matching problem.

INPUT: An undirected graph $G = (V, E)$, and a weight function $W : V \rightarrow \mathbb{R}^+$. In other words, each vertex has a positive weight associated with it.

OUTPUT: A matching M in G such that the sum of the weights of the vertices covered by M is maximum. Reminder: a matching is a set of edges such that no two edges in the set touch the same vertex.

a) Describe a subset system (V, \mathcal{I}) such that the solution to the maximum vertex weight matching problem is the maximum weight element of \mathcal{I} , and such that (V, \mathcal{I}) is a matroid. Prove that (V, \mathcal{I}) is a matroid. Hint: the set V should be the vertex set, and not the edge set.

b) Design a greedy algorithm for the maximum vertex weight matching problem, and prove that it works correctly. For this portion of the problem, you can assume that you have a “black box” for testing whether a subset $U \subseteq V$ is such that $U \in \mathcal{I}$.

c) For the case where the graph G is bipartite, design an algorithm for performing these black box tests. What is the running time of the resulting algorithm in terms of $|E|$ and $|V|$? Hint: in lecture, we discussed the concept of an augmenting path - this concept is useful here.