



Algorithm Theory

Exercise Sheet 3

Due: Friday, 31st of October 2025, 10:00 am

Assumption: You may assume that calculations with real numbers can be performed with arbitrary precision in constant time.

Exercise 1: Faster Polynomial Multiplication (12 Points)

Let $p(x) := -3x^2 + x + 6$ and $q(x) := 2x^2 + 4$. The goal is to compute $p(x) \cdot q(x)$ with the help of the FFT algorithm. Please, make use of the following sketch:

1. Illustrate the **divide** procedure of the algorithm (for both functions p and q). More precisely, for the i -th divide step (with focus on $p(x)$), write down all the polynomials p_{ij} for $j \in \{0, \dots, 2^i - 1\}$ that you obtain from further dividing the polynomials from the previous divide step $i - 1$ (we define $p_{00} := p$, and the first split is into p_{10} and p_{11} and so on...).
2. Illustrate the **combine** procedure of the algorithm (for both functions p and q). That is, starting with the polynomials of the smallest degree as base cases, compute the DFT of p_{ij} (respectively q_{ij}) bottom up with the recursive formula given in the lecture. The recursion stops when $DFT_8(p_{00})$ (respectively $DFT_8(q_{00})$) is computed i.e., we know the function's values at the (8-th) roots of unity.
3. **Multiply** the polynomials. More specific, give the point value representation of $p(x) \cdot q(x)$, i.e., $(w_8^0, y_0), (w_8^1, y_1), \dots, (w_8^7, y_7)$.
4. Use the **inverse** DFT procedure from the lecture to get the final coefficients for $p(x) \cdot q(x)$. To do that efficiently, first compute the $DFT_8(f)$ where $f(x) := y_0 + y_1 \cdot x + \dots + y_7 \cdot x^7$ and then compute the coefficients a_k for $k \in \{0, 1, \dots, 7\}$ of $p(x) \cdot q(x)$ (using that $a_k = 1/8 \cdot f(w_8^{-k})$).

Write down all intermediate results to get partial points in the case of a typo.

Exercise 2: Strings and Polynomials (8 Points)

You are given a string S consisting of only the characters 'A' and 'B'. For each integer k between 1 and $n - 1$ (where n is the length of the string), we define a k -inversion as a pair of indices (i, j) with $1 \leq i < j \leq n$ where $j - i = k$ such that $S[i] = 'B'$, $S[j] = 'A'$.

In other words, a k -inversion is a pair of indices (i, j) such that the character at position i is 'B', the character at position $i + k$ is 'A'. For each $k \in \{1, 2, \dots, n - 1\}$, your task is to compute the number of all k -inversions in the string S .

Example

If the input is BABA (i.e., $n = 4$), the output is $[2, 0, 1]$, as for $k = 1$ there are two valid inversions $((1, 2)$ and $(3, 4))$, for $k = 2$ there is no inversion at all and for $k = 3$ we have 1 inversion at $(1, 4)$.

Task

A naive solution would involve iterating through all pairs of indices, leading to a time complexity of $O(n^2)$. Your task is to come up with an algorithm that improves the time complexity to $O(n \log n)$.

Hint: The idea is to construct 2 polynomials, say $p_A(x)$ and $p_B(x)$, such that from the coefficients of $p_A(x) \cdot p_B(x)$ you can deduce a solution to the problem. A potential choice for $p_A(x)$ could be for example $p_A(x) := \sum_{j \in I_A} x^j$, where I_A is the set of indices j where $S[j] = 'A'$, e.g. for BABA we would have $I_A = \{2, 4\}$. However, defining $p_B(x)$ in the same way would not directly solve the task. Try to adapt!

Remark: According to the definition given in the lecture, polynomials do not have negative integer powers. However, it is still possible to multiply "polynomials" with negative powers of x efficiently using FFT by multiplying the polynomial with some x^m (for some large enough $m > 0$) such that all powers become non-negative. Now, one can apply FFT and afterwards get rid of this additional x^m terms.

Exercise 3: FFT Application for the exercise session (0 Points)

Let A, B be two sets of integers between 0 and n i.e., $A, B \subseteq \{0, 1, 2, \dots, n\}$. We define two random variables X_A and X_B , where X_A is obtained by choosing a number uniformly at random from A and X_B is obtained by choosing a number uniformly at random from B . We further define the random variable $Z := X_A + X_B$. Note that Z can take values in the range $0, \dots, 2n$.

Give an $O(n \log n)$ algorithm to compute the distribution of Z . Hence, the algorithm should compute the probability $P(Z = z)$ for all $z \in \{0, \dots, 2n\}$. Note that $\sum_{z=0}^{2n} P(Z = z) = 1$. You can use the algorithms of the lecture as a black box. State the correctness of your algorithm and also explain the runtime!