TUTORIAL 4

In all the exercises, K is a commutative field of characteristic not equal to 2 (the FFT is quite tricky to work out in characteristic 2 – no nice roots of unity). We assume that all operations in K cost $O(1)$. We denote $M(n)$ for the complexity of multiplying two polynomials of degree n.

1 FFT as a particular multipoint evaluation

1. Let $n = 2^k \in \mathbb{N}$, and P and Q be two polynomials of $K[X]$ with degree at most $n/2 - 1$. Explain why the FFT algorithm is a particular case of the fast multipoint evaluation algorithm.
2. Recall the complexity of multiplying P and Q using the FFT algorithm. What is the general complexity of fast multi-point evaluation at n points? Why is the complexity of the FFT algorithm better than in the general fast multipoint evaluation algorithm?

2 Deterministic factorization

In this exercise we develop Strassen's factorization method (sometimes called Pollard-Strassen factorization algorithm). This method *deterministically* finds the prime factorization of a positive integer N in time $O(N^{1/4+\varepsilon})$. Up to $poly(\log N)$ factors, this is the fastest method known so far.

1. Consider the simplest case when N is a product of two primes, namely, $N = p \cdot q$ (assume, $p < q$). Let $d = \lceil N^{1/4} \rceil$. Show how to compute a non-trivial factor of N knowing $(d^2)! \bmod N$ in time $O(M(\log N) \log \log N)$.
2. Consider the polynomial
$$f(x) = (x+1)(x+2) \cdot \ldots \cdot (x+d) \in (\mathbb{Z}/N\mathbb{Z})[x].$$
Show how to compute $(d^2)!$ using multipoint evaluation of f in time $O(M(d) \log d)$, where $M(d)$ is time needed to multiply two polynomials of degree d. Conclude on the running time for factoring N.
3. Now assume $N = p \cdot q \cdot r$. What can go wrong in the above algorithm? Suggest a method that solves this problem.

3 Determinant

Let $M \in \mathcal{M}_n(\mathbb{K}[X])$. Assume that all the entries of M have degree at most d. Give an evaluation-interpolation algorithm for computing $\det(M)$. What is its complexity ?

4 Fast CRT

1. Recall (any version of) the Chinese Remainder Theorem.

   Let $P_i \in K[X]$ for $i \in \{0, \dots, k-1\}$ be pairwise coprime polynomials, with $d_i := \deg P_i$. Let $N = \prod_{i=0}^{k-1} P_i$ and $n := \sum_{i=0}^{k-1} d_i = \deg N$ and k – a power-of-two.

   Note some useful properties of $M(n)$: $\sum_{i=0}^{k-1} M(d_i) \leq M(n)$ (M is superlinear) and $M(2n) = O(M(n))$.

2. Let $u_0, \dots, u_{k-1}$ be polynomials with $\deg u_i < d_i$. Give an algorithm of complexity $O(M(n) \log n \log k)$ to compute a polynomial x of degree $< n$ such that

$$x = u_i \mod P_i \ \forall i \in [k]. \tag{1}$$

   (Bonus: Note that your algorithm works in the integer case (if P_i and u_i are integers).)

3. Prove that one can compute all the polynomials $R_i := N \bmod P_i^2$ in time $O(M(n) \log k)$ (generalize fast multipoint evaluation).

4. Define $S_i = (R_i/P_i)^{-1} \bmod P_i$. Show that S_i is well defined (i.e., R_i/P_i is invertible modulo P_i) and that one can compute all the S_i's in time $O(M(n) \log n)$.

5. Prove that $x = \sum_{i=0}^{k-1} c_i N/P_i$ with $c_i = u_i S_i \bmod P_i$ is a solution to question 2, and explain how to compute x in time $O(M(n) \log n)$.