TUTORIAL 3

1 Alternative FFT algorithm

Let P be a polynomial of degree at most $2^k - 1$, and write $P = P_h X^{2^{k-1}} + P_l$. Let ω be a primitive 2^k-th root of 1.

1. Prove that $P(\omega^{2i}) = P_h(\omega^{2i}) + P_l(\omega^{2i})$ and $P(\omega^{2i+1}) = -P_h(\omega^{2i+1}) + P_l(\omega^{2i+1})$
2. Deduce an alternative FFT algorithm. You will need to introduce the polynomial

$$Q(X) = P_l(\omega X) - P_h(\omega X).$$

2 The "binary splitting" method computation of $n!$

We want to compute $n!$. We denote by $M(k)$ the cost (in terms of elementary operations) of the multiplication of two k-bit numbers, and we assume $2M(k/2) \leqslant M(k)$ (we recall some typical values: $M(k) = O(k^2)$ with naive multiplication, $O(k^{\log(3)/\log(2)})$ with Karatsuba multiplication and $O(k \log k \log\log k)$ with the FFT-in finite ring variant of the Schönhage & Strassen algorithm). Use the fact that $\log n! \sim n \log n$.

1. What is the cost of multiplying $O(n)$-digit integer by a $O(1)$-digit integer by the naive algorithm. Argue that it is essentially optimal.
2. We first consider the simplest approach: $x_1 = 1,\ x_2 = 2x_1,\ x_3 = 3x_2,\ \ldots,\ x_n = nx_{n-1}$. Show that the cost of this approach is $O(n^2(\log n)^2)$.
3. We define

$$p(a, b) = (a + 1)(a + 2) \cdots (b - 1)b = \frac{b!}{a!}.$$

Suggest a recursive method to compute $n!$ with cost $O(\log n M(n \log n))$. Conclude on the complexity of your method under different values of $M(k)$.

3 Computing square root of $F \bmod X^n$

In class, you have seen how to compute $\sqrt{F} \bmod X^n$ for a polynomial F of degree $< n$ in time $3M(n)$ using Newton's iteration. In this exercise, we develop an algorithm to compute the square root of $F \bmod X^n$. For simplicity, assume $n = 2^k$.

1. Consider the polynomial

$$\Phi(y) = y^2 - F.$$

Give an algorithm to compute $\sqrt{F} \bmod X^n$ and argue about its complexity (expressing this complexity as $c \cdot M(n) + \Theta(n)$, we are mostly interested to determine c).

Using $\Phi(y)$, the algorithm will be a recursive algorithm of the form

- *If $n = 1$, compute $\sqrt{F_0}$*
- *Else*
  1. *Call the algorithm recursively to compute $G = \sqrt{F} \bmod X^{n/2}$*
  2. *Return $G - \frac{1}{2}\left(G - \frac{F}{G}\right) \bmod X^n$*

Complexity, in class, we've seen how to perform inversion $(1/G \bmod X^n)$ in 3 multiplications $M(n)$. Multiplying the result by F takes another $M(n)$ time, giving in total $4M(n) + \Theta(n)$ time to compute Step 2. Hence, the recursion will be of the form

$$T(n) = T(n/2) + 4M(n) + \Theta(n).$$

Applying the master theorem and using the fact that $M(1) = \Theta(1)$ and $2M(n/2) \leq M(n)$, we obtain $T(n) = 8M(n) + \Theta(n)$.

The correctness has been shown during the TD.

2. Can you improve your algorithm by considering

$$\Phi(y) = \frac{1}{y^2} - F \ ?$$

Give an algorithm and argue about its complexity.

Step 2 of the above algorithm now will be

$$\textit{Return } G + \frac{1}{2}(G - G^3 F).$$

Note that at the end of the recursion we obtain the inverse of the square root, i.e., $1/\sqrt{f}$. So at the end we should multiply the result by F and obtain $f \cdot \frac{1}{\sqrt{F}} = \sqrt{F} \bmod X^n$.

The complexity of Step 2 now is $M(n/2)$ to compute $g^2 \bmod x^n, \deg g = n/2$, $+2M(n)$ to compute $fg^2 \bmod x^n$ and $g \cdot (fg^2 \bmod x^n) \bmod x^n$. That is, the recurrence is of the form $T(n) = T(n/2) + \frac{5}{2}M(n) + \Theta(n)$, leading to the complexity $T(n) = 5M(n) + \Theta(n)$. Final multiplication add another $M(n)$ to the complexity, giving $6M(n) + \Theta(n)$ in total.