CS 3510: Design and Analysis of Algorithms, Summer 2023
Instructor: He Jia, Georgia Institute of Technology.

Lecture 4, 5/25/23. Scribed by Abdullah AlJasser, Andrew McBurnett, Robert Terpin.

1 Poll Questions

1.1 Questions

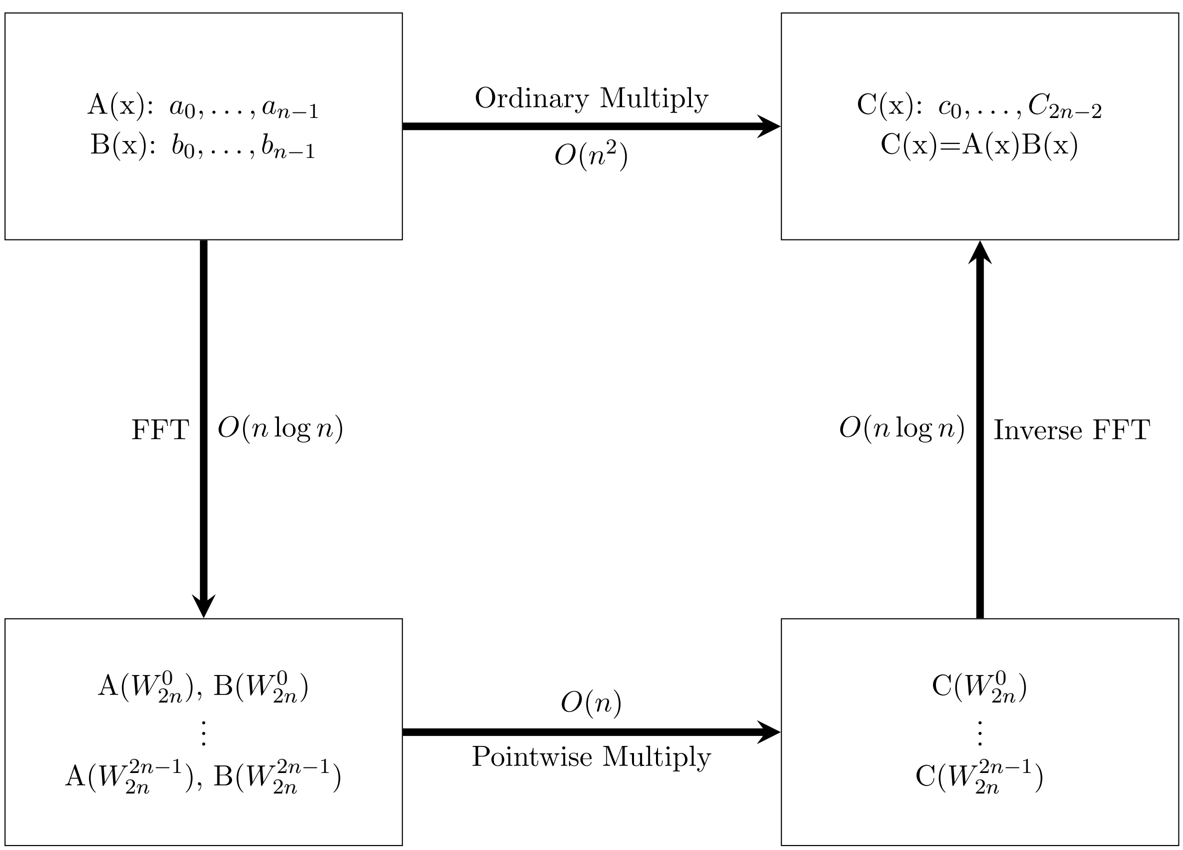
- How many distinct points are needed to uniquely determine a degree-k polynomial?
  - 2
  - k-1
  - k
  - k+1
- How many coefficients are needed to uniquely determine a degree-k polynomial?
  - 2
  - k-1
  - k
  - k+1
- What is the degree of the product of two polynomials of degree k?
  - k
  - k+1
  - 2k
  - k^2
- How many complex roots does $z^n = 1$ have?
  - 1
  - 1 or 2
  - n
  - n+1

1.2 Answers

- How many distinct points are needed to uniquely determine a degree-k polynomial?
  - k+1
- How many coefficients are needed to uniquely determine a degree-k polynomial?
  - k+1
- What is the degree of the product of two polynomials of degree k?
  - 2k
- How many complex roots does $z^n = 1$ have?
  - n

2 Recap

2.1 Polynomial Multiplication Process



In this problem, we take the coefficients of two polynomials A(x) and B(x) each with a degree of $n-1$ at the most, and we want to multiply them to get C(x). Instead of using ordinary multiplication which is $O(n^2)$, we can use the FFT polynomial multiplication algorithm to get a better efficiency.

2.2 Algorithm: Polynomial-Multiplication-FFT

Input:
$a = (a_0, ..., a_{n-1})$
$b = (b_0, ..., b_{n-1})$

Output:
$c = (c_0, ..., c_{2n-2})$

Algorithm Steps:

1. Run FFT(a, ω_{2n}) and FFT(b, ω_{2n}) to get $A(x)$ and $B(x)$ at the $(2n)^{th}$ roots of unity.

2. Multiply to get $C(x) = A(x)B(x)$ at the $(2n)^{th}$ roots of unity.

3. Run InverseFFT(C) to get $c = (c_0, ..., c_{2n-2})$.

2.3 Algorithm: FFT

Input:
$a = (a_0, ..., a_{n-1})$
ω, an n^{th} root of unity

Output:
$A(\omega^0), A(\omega^1), ..., A(\omega^{n-1})$

Algorithm Steps:

1. if $\omega = 1$, return $A(1)$

2. Let $a_{even} = (a_0, a_2, ..., a_{n-2})$ and $a_{odd} = (a_1, a_3, ..., a_{n-1})$

3. $(s_0, s_1, ..., s_{\frac{n}{2}-1}) = FFT(a_{even}, \omega^2)$

4. $(t_0, t_1, ..., t_{\frac{n}{2}-1}) = FFT(a_{odd}, \omega^2)$

5. For $j = 0 \to \frac{n}{2} - 1$

   - $r_j = s_j + \omega^j t_j$
   - $r_{\frac{n}{2}+j} = s_j - \omega^j t_j$

6. Return $(r_0, r_1, ..., r_{n-1})$

NOTE: in the above algorithm, r_j represents $A(\omega^j)$, and $r_{\frac{n}{2}+j}$ represents $A(\omega^{j+\frac{n}{2}})$

3 Matrix View of FFT

For a polynomial A(x), when we apply $FFT(a, \omega_n)$, we get $A(\omega_n^0), A(\omega_n^1), ..., A(\omega_n^{n-1})$. The following is a matrix representation of the FFT algorithm.

1. For points $x_0, x_1, ..., x_{n-1}$

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

2. $x_j = \omega_n^j$ for $j = 0, ..., n-1$

$$\begin{bmatrix} A(\omega_n^0) \\ A(\omega_n^1) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$A = M_n(\omega_n)a$$

3. Inverse FFT: $a = (M_n(\omega_n))^{-1}A$

<u>Lemma</u>

$$(M_n(\omega_n))^{-1} = \frac{1}{n}M_n(\omega_n^{-1})$$

$$\omega_n = e^{\frac{2\pi i}{n}}$$
$$\omega_n^{-1} = \omega_n^{n-1} = e^{\frac{2\pi i}{n}(n-1)}$$

$$m_n(\omega_n^{-1}) =$$

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{bmatrix}$$