universität freiburg

Algorithm Theory – WS 2024/25

Chapter 1 : Divide and Conquer Algorithms
(Multiplication of Polynomials)

Fabian Kuhn
Dept. of Computer Science
Algorithms and Complexity

Polynomials

Real polynomial p in one variable x:

$$p(x) = a_{n-1}x^{n-1} + \ldots + a_1x^1 + a_0$$

Coefficients of p: $a_0, a_1, \ldots, a_{n-1} \in \mathbb{R}$

Degree of p: largest power of x in p ($n - 1$ in the above case)

Example:

$$p(x) = 3x^3 - 15x^2 + 18x$$

Set of all real-valued polynomials in x: $\mathbb{R}[x]$ (polynomial ring)

Operations on Polynomials

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

- How expensive are basic operations on these polynomials?
  - **Evaluation:** What is $p(x_0)$ for a given value $x_0 \in \mathbb{R}$?
  - **Addition:** Compute the polynomial $p(x) + q(x)$
  - **Multiplication:** Compute the polynomial $p(x) \cdot q(x)$

We will focus on multiplication.

- **Computational Models**
  - **RAM (random access machine):** standard model for algorithm analysis
    - Reading / writing one memory cell costs 1 time unit
    - Basic arithmetic op. on integers cost 1 time unit (if integers fit in a mem. cell)
  - **Real RAM:**
    - Also basic arithmetic operations on real numbers cost 1 time unit
    - We will now use this assumption

Operations on Polynomials : Evaluation

- Given: Polynomial $p \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

- **Horner's method** for evaluation at specific value x_0:

$$p(x_0) = (\ldots((a_{n-1}x_0 + a_{n-2})x_0 + a_{n-3})x_0 + \cdots + a_1)x_0 + a_0$$

$$(\;\ldots ((a_{n-1} \cdot x_0 + a_{n-2}) \cdot x_0 + a_{n-3}) \cdot x_0 + a_{n-4} \quad) \cdot x_0 + a_0$$

- Pseudo-code:

  $p \coloneqq a_{n-1};\ i \coloneqq n-1;$
  while $(i > 0)$ **do**
  $\quad i \coloneqq i - 1;$
  $\quad p \coloneqq p \cdot x_0 + a_i$

- Running time: $O(n)$

Operations on Polynomials : Addition

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

- Compute sum $p(x) + q(x)$:

$$\begin{aligned} & p(x) + q(x) \\ & \quad = (a_{n-1}x^{n-1} + \cdots + a_0) + (b_{n-1}x^{n-1} + \cdots + b_0) \\ & \quad = (a_{n-1} + b_{n-1})x^{n-1} + \cdots + (a_1 + b_1)x + (a_0 + b_0) \end{aligned}$$

- Running time: $O(n)$

Operations on Polynomials : Multiplication

- Given: Polynomials $p, q \in \mathbb{R}[x]$ of degree $n-1$

$$p(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$
$$q(x) = b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$$

- Product $p(x) \cdot q(x)$:

$$p(x) \cdot q(x) = (a_{n-1}x^{n-1} + \cdots + a_0) \cdot (b_{n-1}x^{n-1} + \cdots + b_0)$$
$$= c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \cdots + c_1 x + c_0$$

- Obtaining c_k: what products of monomials have degree i?

$$\text{For } 0 \le k \le 2n-2\text{: } c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

  where $a_i = b_i = 0$ for $i \ge n$.

- Running time naïve algorithm: $O(n^2)$

Operations on Polynomials : Faster Multiplication?

- Multiplication is slow $\left(\Theta(n^2)\right)$
- Try divide-and-conquer to get a faster algorithm
- Assume: degree is $n-1$, n is even
- Divide polynomial $p(x) = a_{n-1}x^{n-1} + \cdots + a_0$ into 2 polynomials of degree ${}^{n}/_{2} - 1$:

$$p_0(x) = a_{n/2-1}x^{n/2-1} + \cdots + a_0$$
$$p_1(x) = a_{n-1}x^{n/2-1} + \cdots + a_{n/2}$$
$$p(x) = p_1(x) \cdot x^{n/2} + p_0(x)$$

- Similarly: $q(x) = q_1(x) \cdot x^{n/2} + q_0(x)$

Polynomial Multiplication : Divide-And-Conquer

- **Divide:**

$$p(x) = p_1(x) \cdot x^{n/2} + p_0(x), \qquad q(x) = q_1(x) \cdot x^{n/2} + q_0(x)$$

- **Multiplication:**

$$p(x)q(x) = p_1(x)q_1(x) \cdot x^n + (p_0(x)q_1(x) + p_1(x)q_0(x)) \cdot x^{n/2} + p_0(x)q_0(x)$$

- 4 multiplications of degree ${}^{n}/_{2} - 1$ polynomials:

$$T(n) = 4T({}^{n}/_{2}) + O(n)$$

$$4^{\log_2 n} = 2^{2 \cdot \log_2 n} = n^2$$

- Leads to $T(n) = \Theta(n^2)$ like the naive algorithm…
  - follows immediately by using the master theorem

Polynomial Multiplication : More Clever Recursive Solution

- **Recall that**

$$p(x)q(x) = p_1(x)q_1(x) \cdot x^n + (p_0(x)q_1(x) + p_1(x)q_0(x)) \cdot x^{n/2} + p_0(x)q_0(x)$$

- Compute $r(x) = (p_0(x) + p_1(x)) \cdot (q_0(x) + q_1(x))$:

$$r(x) = p_0(x)q_0(x) + p_0(x)q_1(x) + p_1(x)q_0(x) + p_1(x)q_1(x)$$

Algorithm:

- Compute (recursively): $p_0(x) \cdot q_0(x)$ $\quad p_1(x) \cdot q_1(x)$ $\quad r(x) = (p_0(x) + p_1(x)) \cdot (q_0(x) + q_1(x))$

- $p(x)q(x) = \square \cdot x^n + (\square - \square - \square) \cdot x^{n/2} + \square$

Polynomial Multiplication : Karatsuba Algorithm

- **Recursive multiplication:**

$$r(x) = \left(p_0(x) + p_1(x)\right) \cdot \left(q_0(x) + q_1(x)\right)$$

$$\begin{aligned} p(x)q(x) &= p_1(x) \cdot q_1(x) \cdot x^n \\ &+ \left(r(x) - p_0(x)q_0(x) - p_1(x)q_1(x)\right) \cdot x^{n/2} \\ &+ p_0(x) \cdot q_0(x) \end{aligned}$$

- Recursively do 3 multiplications of degree $(n/2 - 1)$-polynomials

$$T(n) = 3T(n/2) + O(n)$$

$$3^{\log_2 n} = \left(2^{\log_2 3}\right)^{\log_2 n} = 2^{\log_2 n \cdot \log_2 3} = \left(2^{\log_2 n}\right)^{\log_2 3} = n^{\log_2 3}$$

- Gives: $T(n) = O\left(n^{\log_2 3}\right) = O(n^{1.58496\ldots})$ (see Master theorem)

- Can we do even better?

Representation of Polynomials

Coefficient Representation: Polynomial of degree $n-1$ defined by coefficients $a_0, \ldots, a_{n-1}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

Point-value Representation: Polynomial $p(x)$ of degree $n-1$ is given by n point-value pairs:

$$p = \{(x_0, p(x_0)), (x_1, p(x_1)), \ldots, (x_{n-1}, p(x_{n-1}))\},$$ where $x_i \neq x_j$ for $i \neq j$.

Example: The polynomial

$$p(x) = 3x^3 - 15x^2 + 18x = 3x(x-2)(x-3)$$

is uniquely defined by the four point-value pairs $(0,0), (1,6), (2,0), (3,0)$.

Operations: Coefficient Representation

$$\boldsymbol{p(x) = a_{n-1}x^{n-1} + \cdots + a_0}, \qquad \boldsymbol{q(x) = b_{n-1}x^{n-1} + \cdots + b_0}$$

Evaluation: Horner's method: Time $O(n)$

Addition:

$$p(x) + q(x) = (a_{n-1} + b_{n-1})x^{n-1} + \cdots + (a_0 + b_0)$$

- Time: $O(n)$

Multiplication:

$$p(x) \cdot q(x) = c_{2n-2}x^{2n-2} + \cdots + c_0, \qquad \text{where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

- Naïve solution: Need to compute product $a_i b_j$ for all $0 \leq i, j \leq n$
- Time: Naïve alg. $O(n^2)$ Karatsuba Alg. $O(n^{1.58496\ldots})$

Operations: ~~Coefficient~~ Point-Value Representation

$$\boldsymbol{p} = \{(\boldsymbol{x_0}, \boldsymbol{p}(\boldsymbol{x_0})), \dots, (\boldsymbol{x_{n-1}}, \boldsymbol{p}(\boldsymbol{x_{n-1}}))\}, \qquad \boldsymbol{q} = \{(\boldsymbol{x_0}, \boldsymbol{q}(\boldsymbol{x_0})), \dots, (\boldsymbol{x_{n-1}}, \boldsymbol{q}(\boldsymbol{x_{n-1}}))\}$$

- Note: We use the **same points $\boldsymbol{x_0, \dots, x_{n-1}}$** for both polynomials.

Addition:

$$p + q = \{(x_0, p(x_0) + q(x_0)), \dots, (x_{n-1}, p(x_{n-1}) + q(x_{n-1}))\}$$

- Time: $O(n)$

Multiplication:

$$p \cdot q = \{(x_0, p(x_0) \cdot q(x_0)), \dots, (x_{2n-2}, p(x_{2n-2}) \cdot q(x_{2n-2}))\}$$

- Time: $O(n)$

- **Remark:** Need both polynomials at (the same) $2n - 1$ points.

Evaluation: Polynomial interpolation can be done in $O(n^2)$

Operations on Polynomials

Cost depending on representation:

	Coefficient	Point-Value
Evaluation	$O(n)$	$O(n^2)$
Addition	$O(n)$	$O(n)$
Multiplication	$O(n^{1.58})$	$O(n)$

default representation

Can we improve this?

Faster Multiplication of Polynomials?

Observation: Multiplication is fast when using the point-value representation

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at points $x_0, x_1, \ldots, x_{2n-2}$

We will consider this first.
Remark: *We can freely choose the points.*

$2 \times 2n$ point-value pairs $\big(x_i, p(x_i)\big)$ and $\big(x_i, q(x_i)\big)$

↓ **Point-wise multiplication**

Time: $O(n)$

$2n$ point-value pairs $\big(x_i, p(x_i)q(x_i)\big)$

↓ **Interpolation**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Coefficient to Point-Value Representation

$N \geq 2n - 1$

Given: Polynomial $p(x)$ by the coefficient vector $(a_0, a_1, \dots, a_{N-1})$

Goal: Compute $p(x)$ for all x in a given set $\boldsymbol{X}$

- Where X is of size $|X| = N$
- Assume that N is a power of 2

We will fix X later.

Divide and Conquer Approach

- Divide $p(x)$ of degree $N - 1$ (N is even) into 2 polynomials of degree ${}^{N}/_{2} - 1$ differently than in Karatsuba's algorithm

- $p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1}$ (even coefficients)
  $p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1}$ (odd coefficients)

Coefficient to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degree $N-1$ into 2 polynomials of degree $N/2 - 1$

$$p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1} \qquad \text{(even coefficients)}$$
$$p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1} \qquad \text{(odd coefficients)}$$

Let's first look at the "combine" step:

- We need to compute $p(x)$ for all $x \in X$ after recursive calls for polynomials p_0 and p_1:
- Plug $y = x^2$ into $p_0(y)$ and $p_1(y)$:

$$p_0(x^2) = a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{N-2} x^{N-2}$$
$$p_1(x^2) = a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_{N-1} x^{N-2}$$

$$\boldsymbol{p(x) = p_0(x^2) + x \cdot p_1(x^2)}$$

Coefficient to Point-Value Representation

Goal: Compute $p(x)$ for all x in a given set X of size $|X| = N$

- Divide $p(x)$ of degree $N-1$ into 2 polynomials of degree $N/2 - 1$

$$p_0(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{N-2} y^{N/2-1} \quad \text{(even coefficients)}$$
$$p_1(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{N-1} y^{N/2-1} \quad \text{(odd coefficients)}$$

Let's first look at the "combine" step:

$$\forall \boldsymbol{x} \in \boldsymbol{X} : \quad \boldsymbol{p}(\boldsymbol{x}) = \boldsymbol{p}_0(\boldsymbol{x}^2) + \boldsymbol{x} \cdot \boldsymbol{p}_1(\boldsymbol{x}^2)$$

- Goal: *recursively compute* $p_0(y)$ and $p_1(y)$ for all $y \in X^2$
  - Where $X^2 := \{x^2 : x \in X\}$
- Generally, we have $|X^2| = |X|$

Coefficient to Point-Value Representation: Analysis

Let's get a recurrence relation for the given algorithm:

Time for polynomial of degree N with set X: $T(N, |X|)$

$$T(N, |X|) = 2 \cdot T\left({}^{N}/_{2}, |X^2|\right) + O(N + |X|)$$

Assume that $|X^2| = |X| = N$:

$$T(N, N) \;=\; 2 \cdot T\left({}^{N}/_{2}, N\right) + O(N) \;=\; 4 \cdot T\left({}^{N}/_{4}, N\right) + O(N)$$

$$= \;\dots\; = N \cdot \left(T(1, N) + O(N)\right)$$

$$T(1, N) \;=\; O(N)$$

$O(N)$

We therefore get $T(N, |X|) = O(N^2)$.

$\Rightarrow$ **We need $|X^2| < |X|$ to get a faster algorithm!**

Faster Algorithm? Choice of X?

In order to have a faster algorithm, we need $|X^2| < |X|$:

$\implies |X^2| < |X|$ if X contains values x and x' such that $x \neq x'$, but $x^2 = x'^2$:

$$\boldsymbol{X = \{-1, +1\} \implies X^2 = \{+1\}}$$

We also need $|(X^2)^2| = |X^4| < |X^2|$:

- Can we get a set Y of size 4 such that $Y^2 = \{-1, +1\}$?

Complex numbers $\mathbb{C}$:

- Define imaginary constant i such that $i^2 = -1$
- Complex numbers: $\mathbb{C} = \{a + i \cdot b \mid a, b \in \mathbb{R}\}$

$$\boldsymbol{Y = \{-1, +1, -i, +i\} \implies Y^2 = \{-1, +1\}}$$

$\forall y \in \mathbb{C} \setminus \{0\}$, there are exactly 2 numbers $x_1, x_2 \in \mathbb{C}$ such that $x_1^2 = x_2^2 = x$

- and more generally exactly k solutions x to the equation $x^k = y$

Faster Algorithm? Choice of X?

For every $y \in \mathbb{C}$ and $c \in \mathbb{N}$, there are exactly c values $x \in \mathbb{C}$ for which $x^c = y$

- Choose N as a power of 2 (say $N = 2^\ell$) and the set X as

$$X := \{x \in \mathbb{C} : x^N = 1\}$$

known as the N^{th} complex roots of unity

- The set X has size $|X| = N = 2^\ell$
- Claim: The set X^2 can be defined as $X^2 = \{y \in \mathbb{C} : y^{N/2} = 1\}$
  - If $y \in X^2$, there is $x \in X$ s.t. $x^2 = y$ and thus $x^N = (x^2)^{N/2} = y^{N/2} = 1$
  - If $y^{N/2} = 1$, there is $x \in X$ s.t. $x^2 = y$, and thus $y^{N/2} = (x^2)^{N/2} = x^N = 1$

X^2 are the $({}^N/_2)^{th}$ complex roots of unity

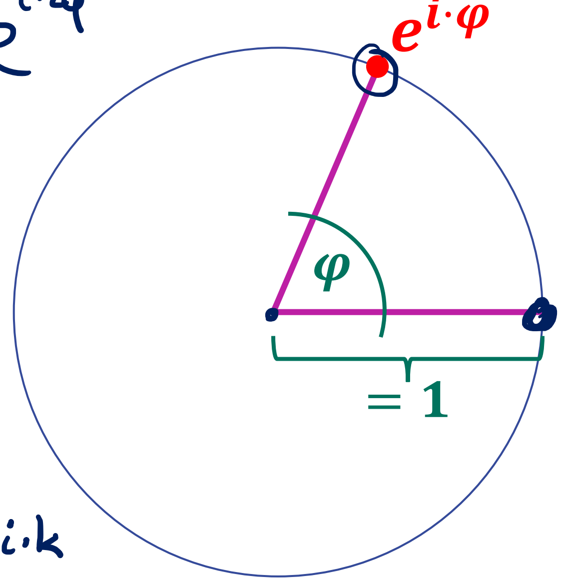
- With the same argumentation, we obtain that
  - For $k = 2^j$ and $j \in \{0, \dots, \ell\}$,we have $X^k = \{y \in \mathbb{C} : y^{N/k} = 1\}$ and thus $|X^k| = N/k$
  - Hence: $|X| = N,\ \ |X^2| = N/2\,,\ \ |X^4| = N/4\,,\ \ |X^8| = N/8\,,\ \ \dots\ ,\ \ |X^N| = 1$

Complex Roots of Unity

Polar Form of Complex Numbers

- A complex number can be written as

$$a + i \cdot b = r \cdot (\cos\varphi + i \cdot \sin\varphi),\ \text{where } r = \sqrt{a^2 + b^2}$$



Euler's Formula: $e^{i\cdot\varphi} = \cos\varphi + i \cdot \sin\varphi$

The N^{th} roots of unity $x_0, \dots, x_{N-1} \in \mathbb{C}$ such that $(x_k)^N = 1$:

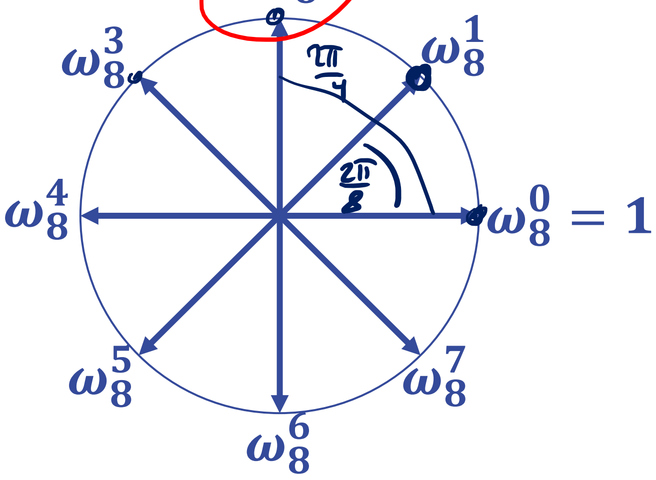
Define $\omega_N := e^{2\pi i/N}$ and $x_k := (\omega_N)^k = \omega_N^k = e^{2\pi i \cdot k/N}$

- We then have

$$(x_k)^N = \left(e^{2\pi i\cdot k/N}\right)^N = e^{2\pi i \cdot k/N \cdot N} = e^{2\pi k \cdot i}$$

$$= \cos(2\pi \cdot k) + i \cdot \sin(2\pi \cdot k) = 1$$

$$= \cos(0) + i \cdot \sin(0) = 1$$



Properties of the Roots of Unity

Cancellation Lemma: For all integers $n > 0$, $k \geq 0$, and $d > 0$, we have:

$$\boldsymbol{\omega_{dn}^{dk} = \omega_n^k}, \qquad \boldsymbol{\omega_n^{k+n} = \omega_n^k}$$

Proof: Recall that $\omega_n = e^{2\pi i/n}$, $e^{2\pi i} = 1$

$$\omega_{dn}^{dk} = (\omega_{dn})^{dk} = \left(e^{\frac{2\pi i}{dn}}\right)^{dk} = e^{\frac{2\pi i}{dn}\cdot dk} = e^{\frac{2\pi i}{n}\cdot k} = \omega_n^k$$

$$\omega_n^{k+n} = \left(e^{\frac{2\pi i}{n}}\right)^{k+n} = e^{\frac{2\pi i}{n}\cdot(k+n)} = e^{\frac{2\pi i}{n}\cdot k} \cdot e^{2\pi i} = \omega_n^k$$

Properties of the Roots of Unity

Claim: If $X = \left\{\omega_{2k}^{j} : j \in \{0, \dots, 2k-1\}\right\}$, we have

$$X^2 = \left\{\omega_{k}^{j} : j \in \{0, \dots, k-1\}\right\}, \qquad |X^2| = \frac{|X|}{2}$$

Proof: We just showed: $\omega_{dn}^{dk} = \omega_n^k$, $\omega_n^{k+n} = \omega_n^k$

- Consider some $x = \omega_{2k}^{j} \in X$:

$$x^2 = \left(\omega_{2k}^{j}\right)^2 = \omega_{2k}^{2j} = \omega_k^j$$

$$\text{If } j \geq k : \omega_k^j = \omega_k^{j-k}$$

- Clearly, $|X^2| = |X|/2 \quad (|X| = 2k, |X^2| = k)$.

Coefficient to Point-Value Representation: Analysis

Time for polynomial of degree $\boldsymbol{N}$ with set $\boldsymbol{X}$: $\boldsymbol{T(N, |X|)}$

$$T(N, |X|) = 2 \cdot T\left({}^{N}/_{2}, |X^2|\right) + O(N + |X|)$$

By choosing $\boldsymbol{X = \{\omega_N^0, \ldots, \omega_N^{N-1}\}}$:

- The number of points gets halved on all recursion levels

To compute $p(x)$ for the N points in X, we recursively compute $p_0(x^2)$ and $p_1(x^2)$ for all $x^2 \in X^2$

- p has degree $N-1$, p_0 and p_1 have degree ${}^{N}/_{2} - 1$, $|X^2| = {}^{|X|}/_{2}$
- Combine step: compute $p(x) = p_0(x^2) + x \cdot p_1(x^2)$ for all $x \in X$

- $|X| = N \implies T(N) \leq 2 \cdot T\left({}^{N}/_{2}\right) + O(N)$

$$\boldsymbol{T(N) = O(N \cdot \log N)}$$

Faster Multiplication of Polynomials?

Observation: Multiplication is fast when using the point-value representation

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at points $\omega_{2n}^0, \omega_{2n}^1, \ldots, \omega_{2n}^{2n-1}$ — Time: $O(n \cdot \log n)$

$2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p(\omega_{2n}^k)\right)$ and $\left(\omega_{2n}^k, q(\omega_{2n}^k)\right)$

↓ **Point-wise multiplication** — Time: $O(n)$

$2n$ point-value pairs $\left(\omega_{2n}^k, p(\omega_{2n}^k) \cdot q(\omega_{2n}^k)\right)$

↓ **Interpolation** — ???

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Discrete Fourier Transform

- The values $p(\omega_N^k)$ for $k = 0, \ldots, N-1$ uniquely define a polynomial p of degree $< N$.

Discrete Fourier Transform (DFT):

- Assume $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ is the coefficient vector of a polynomial p (of degree $\leq N-1$):

$$p(x) = a_{N-1}x^{N-1} + \cdots + a_1 x + a_0$$

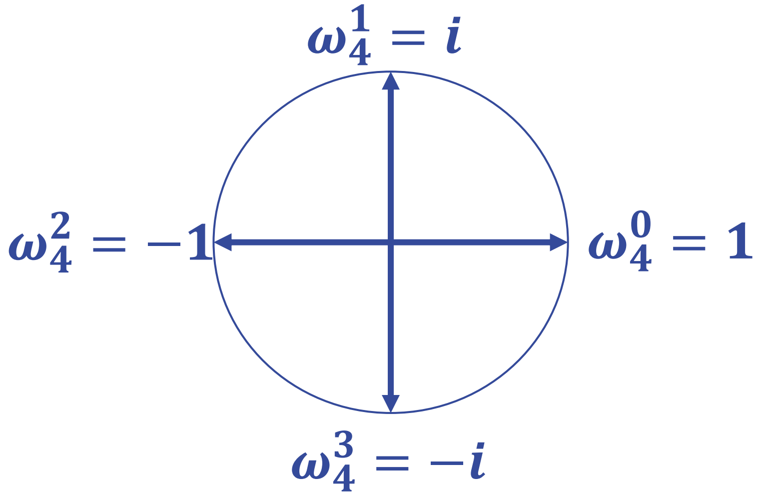
- Then, the Discrete Fourier Transform of the vector $\boldsymbol{a}$ is defined as

$$\mathbf{DFT}_N(\boldsymbol{a}) \coloneqq \left(\boldsymbol{p}(\boldsymbol{\omega}_N^0), \boldsymbol{p}(\boldsymbol{\omega}_N^1), \ldots, \boldsymbol{p}(\boldsymbol{\omega}_N^{N-1})\right)$$

We abuse notation and also write $\mathbf{DFT}_N(\boldsymbol{p})$

Discrete Fourier Transform : Example

Consider polynomial $p(x) = 3x^3 - 15x^2 + 18x$ and choose $N = 4$



Complex roots of unity:

- $\omega_4 = e^{2\pi i/4} = e^{i \cdot \pi/2} = i$
- $\omega_4^0 = 1,\ \omega_4^1 = i,\ \omega_4^2 = -1,\ \omega_4^3 = -i$

Evaluate $p(x)$ at $\omega_4^0, \dots, \omega_4^3$:

$$\left(\omega_4^0, p(\omega_4^0)\right) = \left(1, p(1)\right) = (1,6)$$
$$\left(\omega_4^1, p(\omega_4^1)\right) = \left(i, p(i)\right) = (i, 15 + 15i)$$
$$\left(\omega_4^2, p(\omega_4^2)\right) = \left(-1, p(-1)\right) = (-1, -36)$$
$$\left(\omega_4^3, p(\omega_4^3)\right) = \left(-i, p(-i)\right) = (-i, 15 - 15i)$$

- For $a = (0,18,-15,3)$: $\quad \mathbf{DFT_4}(\boldsymbol{a}) = (\mathbf{6, 15 + 15}\boldsymbol{i}\mathbf{, -36, 15 - 15}\boldsymbol{i})$

Computing of DFT : Summary

$\omega_N^{2k} = \omega_{N/2}^{k}$

Divide-and-conquer algorithm for $\mathrm{DFT}_N(p)$ for some poly. p with N coefficients $a_0, \dots, a_{N-1}$:

$p(x) = p_0(x^2) + x \cdot p_1(x^2)$

1. Divide

$N \leq 1$: $\mathrm{DFT}_N(p) = a_0$

$N > 1$: Divide p into p_0 (even coefficients) and p_1 (odd coefficients).

2. Conquer

Solve $\mathrm{DFT}_{N/2}(p_0)$ and $\mathrm{DFT}_{N/2}(p_1)$ recursively

3. Combine

Compute $\mathrm{DFT}_N(p)$ based on $\mathrm{DFT}_{N/2}(p_0)$ and $\mathrm{DFT}_{N/2}(p_1)$.

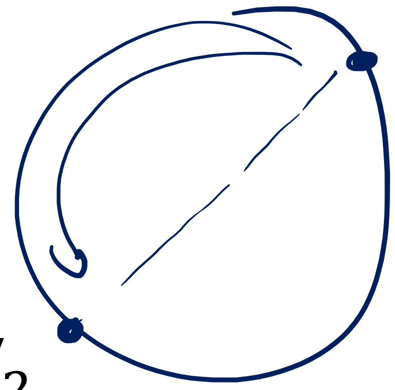
Evaluation for $k = 0, \dots, N-1$:

$$\mathrm{DFT}_N(p) = \left(p(\omega_N^0), p(\omega_N^1), \dots, p(\omega_N^{N-1})\right)$$

$$p(\omega_N^k) = p_0\left((\omega_N^k)^2\right) + \omega_N^k \cdot p_1\left((\omega_N^k)^2\right)$$

$$= \begin{cases} p_0(\omega_{N/2}^k) + \omega_N^k \cdot p_1(\omega_{N/2}^k) & \text{if } k < N/2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) + \omega_N^k \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq N/2 \end{cases}$$

Small Constant Improvement



Polynomial p of degree $N-1$:

$$p(\omega_N^k) = \begin{cases} p_0(\omega_{N/2}^k) + \omega_N^k \cdot p_1(\omega_{N/2}^k) & \text{if } k < {}^N\!/_2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) + \omega_N^k \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq {}^N\!/_2 \end{cases}$$

$$= \begin{cases} p_0(\omega_{N/2}^k) + \omega_N^k \cdot p_1(\omega_{N/2}^k) & \text{if } k < {}^N\!/_2 \\ p_0\left(\omega_{N/2}^{k-N/2}\right) - \omega_N^{k-N/2} \cdot p_1\left(\omega_{N/2}^{k-N/2}\right) & \text{if } k \geq {}^N\!/_2 \end{cases}$$

- $\omega_N^{k-N/2} = e^{\frac{2\pi i}{N}\cdot(k-{}^N\!/_2)} = e^{\frac{2\pi i}{N}\cdot k} \cdot e^{-\frac{2\pi i}{N}\cdot\frac{N}{2}} = \omega_N^k \cdot e^{-\pi i} = -\omega_N^k$

Need to compute $p_0(\omega_{N/2}^k)$ and $\omega_N^k \cdot p_1(\omega_{N/2}^k)$ for $0 \leq k < {}^N\!/_2$.

Example $N = 8$

$$p(\omega_8^0) = p_0(\omega_4^0) + \omega_8^0 \cdot p_1(\omega_4^0)$$

$$p(\omega_8^1) = p_0(\omega_4^1) + \omega_8^1 \cdot p_1(\omega_4^1)$$

$$p(\omega_8^2) = p_0(\omega_4^2) + \omega_8^2 \cdot p_1(\omega_4^2)$$

$$p(\omega_8^3) = p_0(\omega_4^3) + \omega_8^3 \cdot p_1(\omega_4^3)$$

$$p(\omega_8^4) = p_0(\omega_4^0) - \omega_8^0 \cdot p_1(\omega_4^0)$$

$$p(\omega_8^5) = p_0(\omega_4^1) - \omega_8^1 \cdot p_1(\omega_4^1)$$

$$p(\omega_8^6) = p_0(\omega_4^2) - \omega_8^2 \cdot p_1(\omega_4^2)$$

$$p(\omega_8^7) = p_0(\omega_4^3) - \omega_8^3 \cdot p_1(\omega_4^3)$$

$$p_0(\omega_4^0) + \omega_8^4 \cdot p_1(\omega_4^0)$$

Fast Fourier Transform (FFT) Algorithm

Divide-and-conquer algorithm to compute the Discrete Fourier Transform

- A highly relevant algorithm in practice with many applications

a is coefficient vector of polynomial p of degree $N-1$

Algorithm FFT(a) (input: array a of length N, where N is a power of 2, output: $\mathrm{DFT}_N(a)$)

if $n = 1$ **then return** a_0 // $a = [a_0]$

$d^{[0]} \coloneqq \mathrm{FFT}([a_0, a_2, \dots, a_{N-2}]);$ // *recursive computation of* $\mathrm{DFT}_{N/2}(a_{even})$

$d^{[1]} \coloneqq \mathrm{FFT}([a_1, a_3, \dots, a_{N-1}]);$ // *recursive computation of* $\mathrm{DFT}_{N/2}(a_{odd})$

$\omega_N \coloneqq e^{2\pi i/N}; \ \omega \coloneqq 1;$ // *initialize* ω to $\omega = \omega_N^0 = 1$

for $k = 0$ **to** $N/2 - 1$ **do**

  $x \coloneqq \omega \cdot d_k^{[1]};$

  $d_k \coloneqq d_k^{[0]} + x; \ d_{k+N/2} \coloneqq d_k^{[0]} - x;$ // *compute* $d_k = p(\omega_N^k)$ *and* $d_{k+N/2} = p\left(\omega_N^{k+N/2}\right)$

  $\omega \coloneqq \omega \cdot \omega_N$ // *update* ω to $\omega = \omega_N^k$

return $d = [d_0, d_1, \dots, d_{N-1}];$

Faster Multiplication of Polynomials?

Observation: Multiplication is fast when using the point-value representation

Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):

p, q of degree $n-1$, n coefficients

↓ **Evaluation** at points $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ — Time: $O(n \cdot \log n)$

$2 \times 2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right)\right)$ and $\left(\omega_{2n}^k, q\left(\omega_{2n}^k\right)\right)$

↓ **Point-wise multiplication** — Time: $O(n)$

$2n$ point-value pairs $\left(\omega_{2n}^k, p\left(\omega_{2n}^k\right) \cdot q\left(\omega_{2n}^k\right)\right)$

↓ **Interpolation**

$p(x)q(x)$ of degree $2n-2$, $2n-1$ coefficients

Interpolation

Goal: Convert point-value representation into coefficient representation

Input: $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ with $x_i \neq x_j$ for $i \neq j$

Output:

Degree-$(n-1)$ polynomial with coefficients $a_0, \ldots, a_{n-1}$ such that

$$
\begin{aligned}
p(x_0) &= a_0 + a_1 \cdot x_0 + a_2 \cdot x_0^2 + \cdots + a_{n-1} \cdot x_0^{n-1} = y_0 \\
p(x_1) &= a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \cdots + a_{n-1} \cdot x_1^{n-1} = y_1 \\
&\vdots \\
p(x_{n-1}) &= a_0 + a_1 \cdot x_{n-1} + a_2 \cdot x_{n-1}^2 + \cdots + a_{n-1} \cdot x_{n-1}^{n-1} = y_{n-1}
\end{aligned}
$$

→ linear system of equations for $a_0, \ldots, a_{n-1}$

Interpolation

Matrix Notation:

$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^{n-1} \\ 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

- System of equations solvable iff $x_i \neq x_j$ for all $i \neq j$

Special Case $x_i = \omega_n^i$:

$$\underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1^{2\cdot 0} & {\omega_n^2}^{2\cdot 1} & {\omega_n^4}^{2\cdot 2} & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}}_{W} \cdot \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}}_{y}$$

Interpolation

Linear system:

$$W \cdot \boldsymbol{a} = \boldsymbol{y} \quad \Longrightarrow \quad \boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$$

$$W_{i,j} = \omega_n^{ij}, \qquad \boldsymbol{a} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}, \qquad \boldsymbol{y} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Claim:

$$W_{i,j}^{-1} = \frac{\omega_n^{-ij}}{n}$$

Proof: Need to show that $W^{-1}W = I_n$

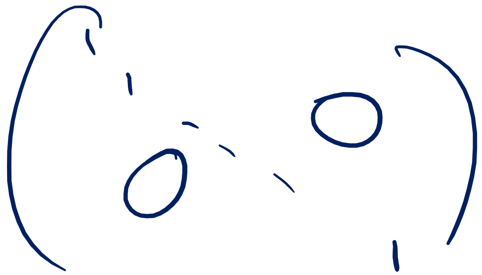
DFT Matrix Inverse

$$W^{-1}W = \begin{pmatrix} & & \cdots & \\ \frac{1}{n} & \frac{\omega_n^{-i}}{n} & \cdots & \frac{\omega_n^{-(n-1)i}}{n} \\ & & \vdots & \\ & & \cdots & \end{pmatrix} \cdot \begin{pmatrix} \cdots & 1 & \cdots \\ \cdots & \omega_n^{j} & \cdots \\ \cdots & \omega_n^{2j} & \cdots \\ & \vdots & \\ \cdots & \omega_n^{(n-1)j} & \cdots \end{pmatrix}$$

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{-\ell i} \cdot \omega_n^{\ell j}}{n} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

We need to show that

- $(W^{-1}W)_{i,j} = 1$ for $i = j$
- $(W^{-1}W)_{i,j} = 0$ for $i \neq j$



DFT Matrix Inverse

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Need to show $(W^{-1}W)_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Case $\boldsymbol{i = j}$:

$$(W^{-1}W)_{i,i} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(i-i)}}{n} = \sum_{\ell=0}^{n-1} \frac{\omega_n^0}{n} = n \cdot \frac{1}{n} = 1$$

DFT Matrix Inverse

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n}$$

Need to show $(W^{-1}W)_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Case $i \neq j$:

$$\omega_n^{nk} = \omega_1^k = 1$$

$$(W^{-1}W)_{i,j} = \sum_{\ell=0}^{n-1} \frac{\omega_n^{\ell(j-i)}}{n} = \frac{1}{n} \cdot \sum_{\ell=0}^{n-1} \left(\omega_n^{j-i}\right)^{\ell} = \frac{1 - \omega_n^{n(j-i)}}{1 - \omega_n^{j-i}} = 0$$

$$\neq 1$$

$q = \omega_n^{j-i}$

Geometric series: $\sum_{\ell=0}^{n-1} q^{\ell} = \frac{1 - q^n}{1 - q}$

Inverse Discrete Fourier Transform

$$W^{-1} = \begin{pmatrix} & & \cdots & \\ \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \cdots & \frac{\omega_n^{-(n-1)k}}{n} \\ & & \vdots & \\ & & \cdots & \end{pmatrix}$$

We get $\boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$ and therefore

$$a_k = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \cdots & \frac{\omega_n^{-(n-1)k}}{n} \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j = \frac{1}{n} \cdot \sum_{j=0}^{n-1} y_j \cdot \left(\omega_n^{-k}\right)^j$$

$q(z) = \sum y_j z^j$