



Chapter 1

Divide and Conquer

Algorithm Theory
WS 2018/19

Fabian Kuhn

Polynomials

Real polynomial p in one variable x :

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

Coefficients of p : $a_0, a_1, \dots, a_n \in \mathbb{R}$

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

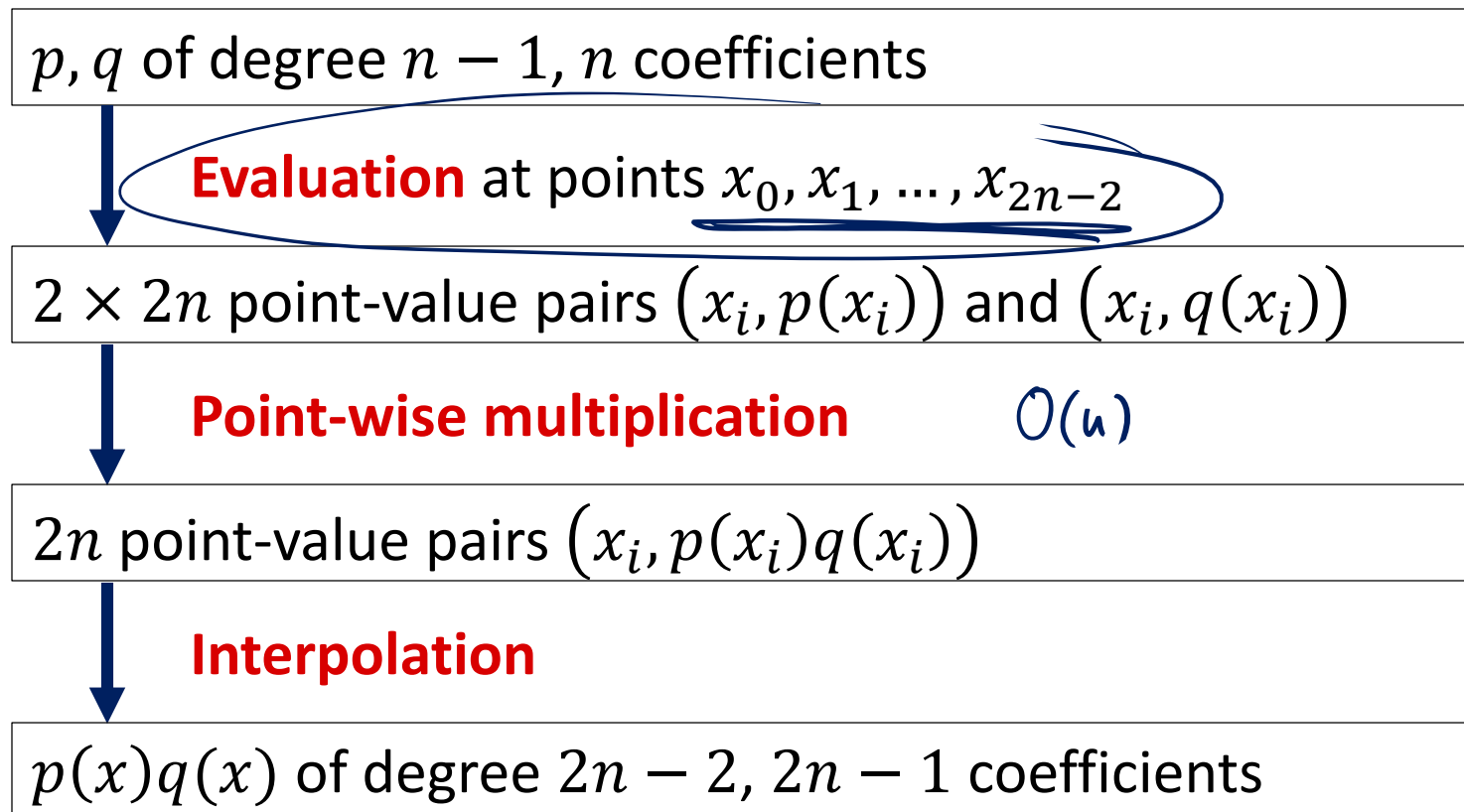
Cost depending on representation:

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

Faster Polynomial Multiplication?

Multiplication is fast when using the **point-value representation**

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



Coefficients to Point-Value Representation



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 X

$$N = 2^{\ell}$$

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

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

Coefficients to Point-Value Representation



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

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

$$p_0(y) = a_0 + a_2y + a_4y^2 + \dots + a_{N-2}y^{N/2-1} \quad (\text{even coeff.})$$

$$p_1(y) = a_1 + a_3y + a_5y^2 + \dots + a_{N-1}y^{N/2-1} \quad (\text{odd coeff.})$$

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

$$\forall x \in X : \underline{p(x)} = \underline{p_0(x^2)} + x \cdot \underline{p_1(x^2)}$$

- Recursively compute $\underline{p_0(y)}$ and $\underline{p_1(y)}$ for all $y \in \underline{X^2}$
 - Where $X^2 := \{x^2 : x \in X\}$

$$|X^2| = |X|$$

$$|X^2| < |X|$$

$$|X^2| = \frac{|X|}{2}$$

- Generally, we have $|X^2| = |X|$

Choice of X

- Select points x_0, x_1, \dots, x_{N-1} to evaluate p and q in a clever way

Consider the N complex roots of unity:

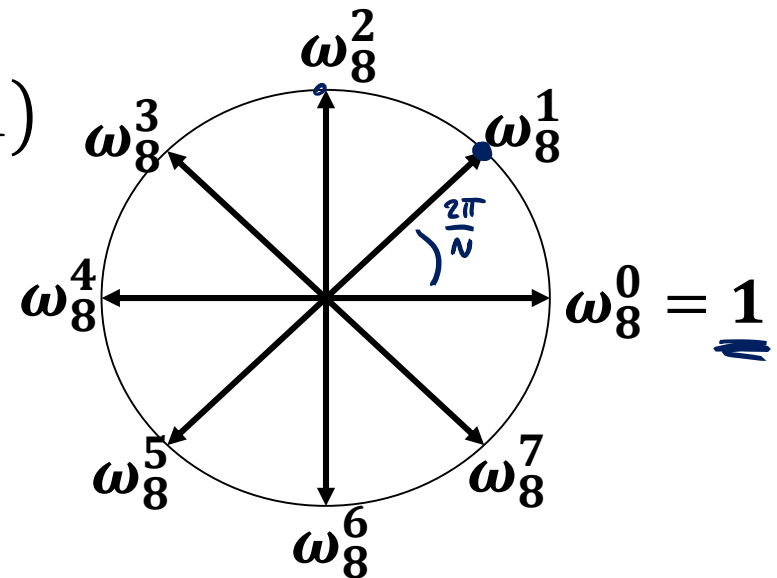
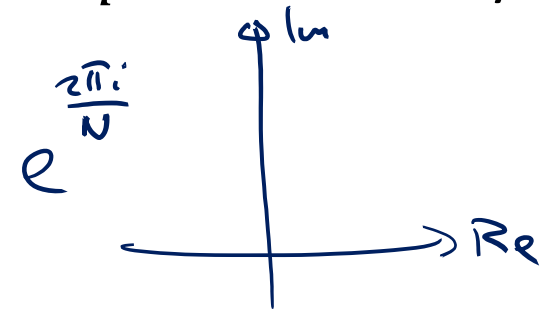
Principle root of unity: $\omega_N = e^{2\pi i/N}$

$$(i = \sqrt{-1}, \quad e^{2\pi i} = 1)$$

Powers of ω_n (roots of unity):

$$1 = \omega_N^0, \omega_N^1, \dots, \omega_N^{N-1}$$

Note: $\omega_N^k = e^{2\pi i k/N} = \cos \frac{2\pi k}{N} + i \cdot \sin \frac{2\pi k}{N}$



Properties of the Roots of Unity

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

$$X^2 = \{\omega_k^i : i \in \{0, \dots, k - 1\}\}, \quad \underline{\underline{|X^2| = \frac{|X|}{2}}}$$

N: $X = \{\omega_N^i \mid i = 0, \dots, N-1\}$

$$|X^2| = \frac{|X|}{2}$$

$$X^2 = \{\omega_{N/2}^i \mid i = 0, \dots, \frac{N}{2} - 1\}$$

Analysis

New recurrence formula:

$$T(N, |X|) \leq 2 \cdot T\left(\frac{N}{2}, \frac{|X|}{2}\right) + \underline{\underline{O(N + |X|)}}$$

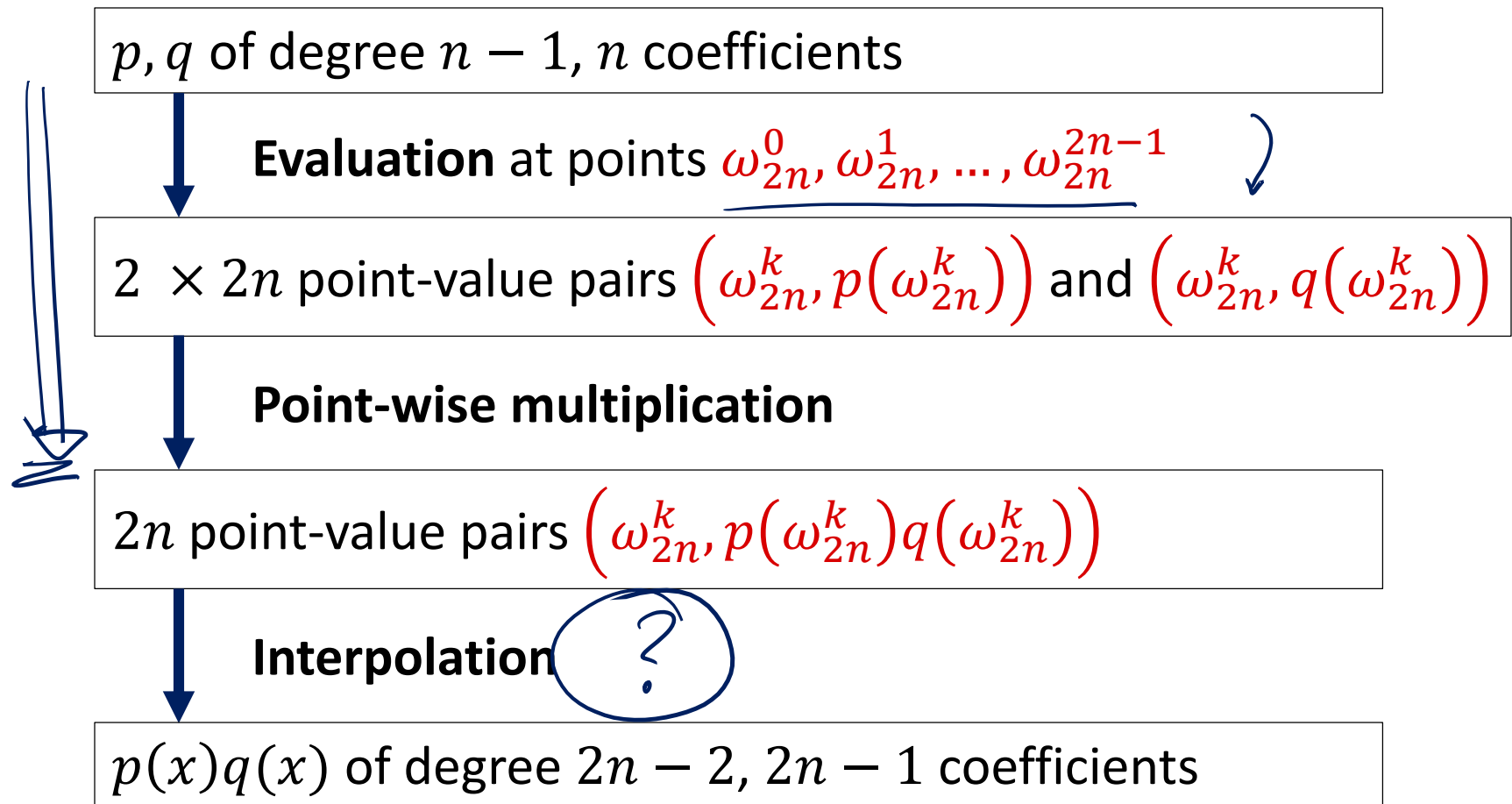
$$|X| = N$$

$$T(N) \leq 2T(N/2) + O(N)$$

$$\implies T(N) = O(N \cdot \log N)$$

Faster Polynomial Multiplication?

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



Discrete Fourier Transform

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

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

Discrete Fourier Transform (DFT):

- Assume $a = (a_0, \dots, a_{N-1})$ is the coefficient vector of poly. p

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

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

Example

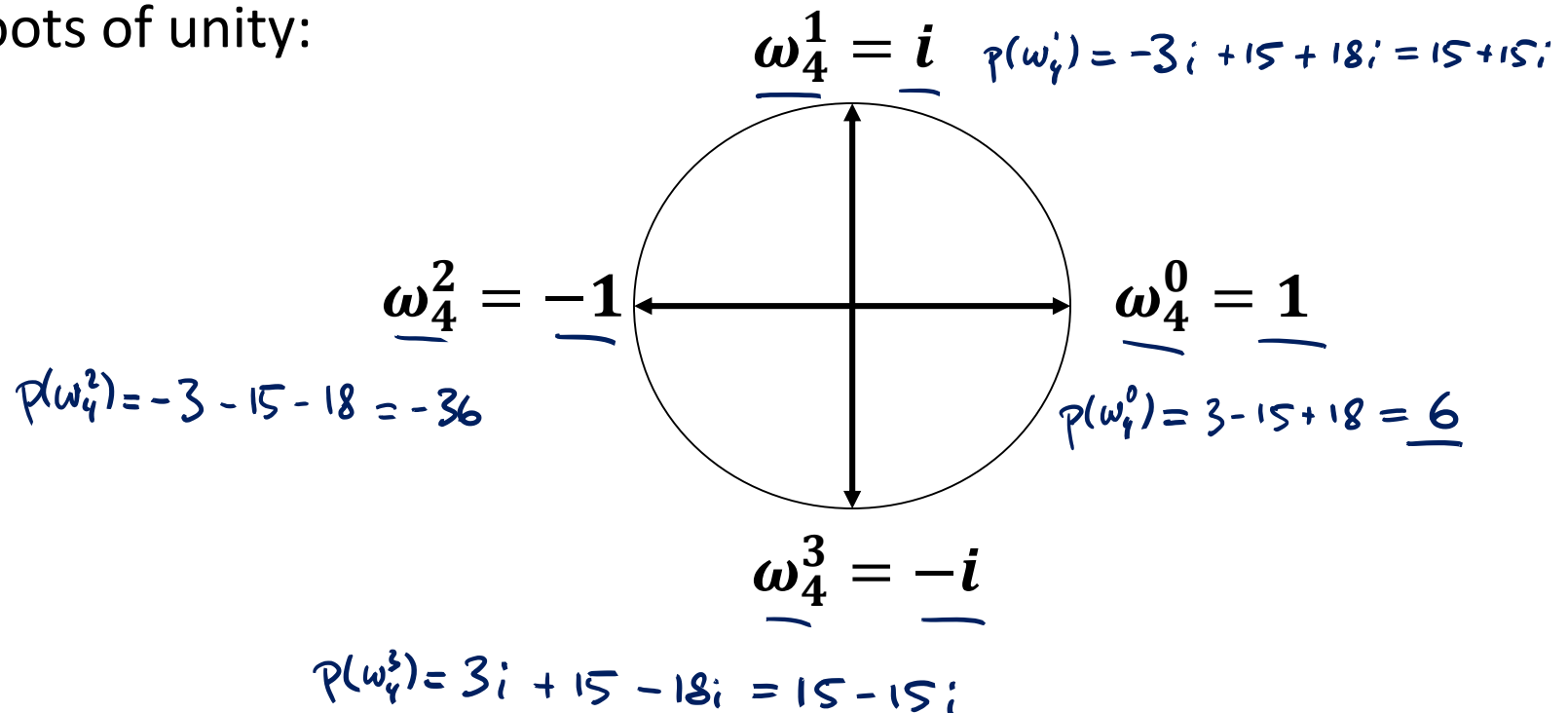
- Consider polynomial $p(x) = 3x^3 - 15x^2 + 18x + 0$

$$a = (0, 18, -15, 3)$$

- Choose $N = 4$

$$\begin{aligned} \text{DFT}_4(a) &= (p(\omega_4^0), p(\omega_4^1), p(\omega_4^2), p(\omega_4^3)) \\ &= \underline{(6, 15+15i, -36, 15-15i)} \end{aligned}$$

- Roots of unity:



Example

- Consider polynomial $p(x) = 3x^3 - 15x^2 + 18x$
- $N = 4$, roots of unity: $\omega_4^0 = 1, \omega_4^1 = i, \omega_4^2 = -1, \omega_4^3 = -i$
- Evaluate $p(x)$ at ω_4^k :

$$\left(\omega_4^0, p(\omega_4^0)\right) = (1, p(1)) = (1, 6)$$

$$\left(\omega_4^1, p(\omega_4^1)\right) = (i, p(i)) = (i, 15 + 15i)$$

$$\left(\omega_4^2, p(\omega_4^2)\right) = (-1, p(-1)) = (-1, -36)$$

$$\left(\omega_4^3, p(\omega_4^3)\right) = (-i, p(-i)) = (-i, 15 - 15i)$$

- For $a = (0, 18, -15, 3)$:

$$\mathbf{DFT}_4(a) = (6, 15 + 15i, -36, 15 - 15i)$$

DFT: Recursive Structure

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



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

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

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

For the coefficient vector a of $p(x)$:

$$\begin{aligned} \text{DFT}_N(a) &= \left(p_0(\omega_{N/2}^0), \dots, p_0(\omega_{N/2}^{N/2-1}), p_0(\omega_{N/2}^0), \dots, p_0(\omega_{N/2}^{N/2-1}) \right) \\ &+ \left(\omega_N^0 p_1(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_1(\omega_{N/2}^{N/2-1}), \omega_N^{N/2} p_1(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_1(\omega_{N/2}^{N/2-1}) \right) \end{aligned}$$

Example

For the coefficient vector a of $p(x)$:

$$\begin{aligned} \text{DFT}_N(a) = & \left(p_0(\omega_{N/2}^0), \dots, p_0(\omega_{N/2}^{N/2-1}), p_0(\omega_{N/2}^0), \dots, p_0(\omega_{N/2}^{N/2-1}) \right) \\ & + \left(\omega_N^0 p_0(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_0(\omega_{N/2}^{N/2-1}), \omega_N^{N/2} p_0(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_0(\omega_{N/2}^{N/2-1}) \right) \\ & \quad \left(p(\omega_4^0), p(\omega_4^1), p(\omega_4^2), p(\omega_4^3) \right) \end{aligned}$$

$N = 4$:

$$\begin{aligned} p(\omega_4^0) &= p_0(\omega_2^0) + \omega_4^0 p_1(\omega_2^0) \\ p(\omega_4^1) &= p_0(\omega_2^1) + \omega_4^1 p_1(\omega_2^1) \\ p(\omega_4^2) &= p_0(\omega_2^0) + \omega_4^2 p_1(\omega_2^0) \\ p(\omega_4^3) &= p_0(\omega_2^1) + \omega_4^3 p_1(\omega_2^1) \end{aligned}$$

Need: $(p_0(\omega_2^0), p_0(\omega_2^1))$ and $(p_1(\omega_2^0), p_1(\omega_2^1))$

(DFTs of coefficient vectors of p_0 and p_1)

Summary: Computation of DFT_N

- Divide-and-conquer algorithm for $\text{DFT}_N(p)$:

1. Divide

$$N \leq 1: \text{DFT}_1(p) = a_0$$

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

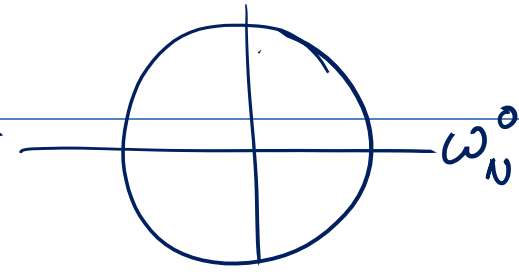
2. Conquer

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

3. Combine

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

Small Improvement

Polynomial p of degree $N - 1$: $-1 = \omega_N^{N/2}$ 

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

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

$$\omega_N^k = -(-1) \cdot \omega_N^k = -(\omega_N^{N/2}) \cdot \omega_N^k = -(\omega_N^{-N/2}) \cdot \omega_N^k = -\omega_N^{k-N/2}$$

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 $N = 8$

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

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

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

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

$$\underline{p(\omega_8^3)} = p_0(\omega_4^0) + \omega_8^4 \cdot p_1(\omega_4^0) - \underline{\omega_8^0 \cdot p_1(\omega_4^0)}$$

$$p(\omega_8^3) = p_0(\omega_4^1) - \omega_8^1 \cdot p_1(\omega_4^1)$$

$$p(\omega_8^3) = p_0(\omega_4^2) - \omega_8^2 \cdot p_1(\omega_4^2)$$

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

Fast Fourier Transform (FFT) Algorithm

Algorithm FFT(a)

- Input: Array a of length N , where N is a power of 2
- Output: $\text{DFT}_N(a)$

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

$d^{[0]} := \text{FFT}([a_0, a_2, \dots, a_{N-2}]);$

$d^{[1]} := \text{FFT}([a_1, a_3, \dots, a_{N-1}]);$

$\omega_N := e^{2\pi i/N}$; $\omega := 1$;

for $k = 0$ **to** $N/2 - 1$ **do** // $\omega = \omega_N^k$

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

$d_k := d_k^{[0]} + x$; $d_{k+N/2} := d_k^{[0]} - x$;

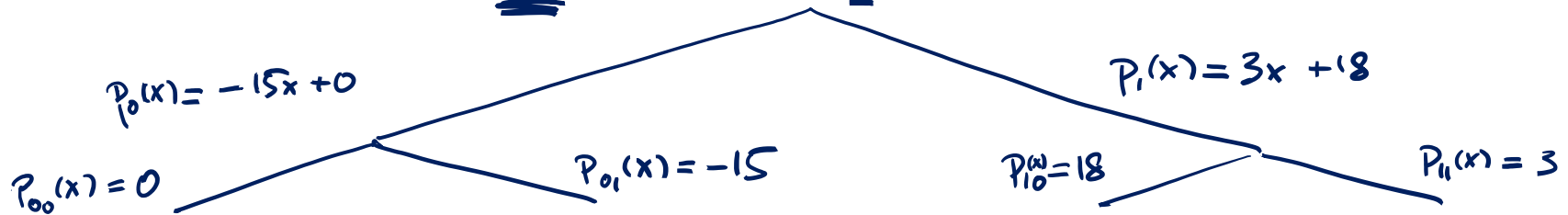
$\omega := \omega \cdot \omega_N$

end;

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

Example

$$p(x) = 3x^3 - \underline{15x^2} + 18x + \underline{0}, \quad a = [0, 18, -15, 3]$$



$$P_0(\omega_2^0) = P_{00}(\omega_1^0) + \omega_2^0 \cdot P_{01}(\omega_1^0) = -15 \quad P_1(\omega_2^0) = P_{10}(\omega_1^0) + \omega_2^0 P_{11}(\omega_2^0) = 21$$

$$P_0(\omega_2^1) = P_{00}(\omega_1^1) - \omega_2^0 P_{01}(\omega_1^0) = +15 \quad P_1(\omega_2^1) = P_{10}(\omega_1^1) - \omega_2^0 P_{11}(\omega_2^0) = 15$$

$$P(\omega_4^0) = P_0(\omega_2^0) + \omega_4^0 \cdot P_1(\omega_2^0) = -15 + 21 = \underline{6}$$

$$P(\omega_4^1) = P_0(\omega_2^1) + \omega_4^1 \cdot P_1(\omega_2^1) = \underline{15 + 15i}$$

$$P(\omega_4^2) = P_0(\omega_2^0) - \omega_4^0 P_1(\omega_2^0) = -15 - 21 = \underline{-36}$$

$$P(\omega_4^3) = P_0(\omega_2^1) - \omega_4^1 P_1(\omega_2^1) = \underline{15 - 15i}$$

Faster Polynomial Multiplication?

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

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



Evaluation at $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ using **FFT** $\mathcal{O}(n \log n)$

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



Point-wise multiplication $\mathcal{O}(n)$

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



Interpolation

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

$$y_i = P(x_i)$$

Convert point-value representation into coefficient representation

Input: $(\underline{x}_0, y_0), \dots, (x_{n-1}, y_{n-1})$ with $x_i \neq x_j$ for $i \neq j$

$$x_i = \omega_n^i$$

Output:

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

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

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

Interpolation

Matrix Notation:

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

known unknown known

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

Special Case $x_i = \omega_n^i$:

$$W_{i,j} = \omega_n^{i \cdot j} \quad i, j = 0, \dots, n-1$$

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

Interpolation

- Linear system:

$$W \cdot \mathbf{a} = \mathbf{y} \quad \Rightarrow \quad \mathbf{a} = \underline{W^{-1}} \cdot \mathbf{y}$$

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

Claim:

$$\underline{W_{ij}^{-1}} = \frac{\omega_n^{-ij}}{n}$$

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

DFT Matrix Inverse

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

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

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

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 $i = j$:

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

DFT Matrix Inverse

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

$\omega_n^{n(j-i)} = 1$

Case $i \neq j$:

$$(W^{-1}W)_{i,j} = \frac{1}{n} \underbrace{\sum_{\ell=0}^{n-1} (\omega_n^{j-i})^{\ell}}_{\text{geometric series}} = \frac{1}{n} \frac{(\omega_n^{j-i})^n - 1}{\underbrace{\omega_n^{j-i} - 1}_{\neq 0}} = 0 \quad \checkmark$$

$$\sum_{\ell=0}^{n-1} q^{\ell} = \frac{q^n - 1}{q - 1} \quad (\text{also works if } q \in \mathbb{C})$$

Inverse DFT

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

- We get \mathbf{a} = W^{-1} · \mathbf{y} and therefore

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

$$= \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j$$

DFT and Inverse DFT

Inverse DFT:

$$a_k = \frac{1}{n} \cdot \sum_{j=0}^{n-1} \omega_n^{-kj} \cdot y_j = \frac{1}{n} \sum_{\ell \Rightarrow j}^{n-1} y_j \cdot z^j$$

$\omega_n^{-k} =: z$

- Define polynomial $q(x) = y_0 + y_1z + \dots + y_{n-1}z^{n-1}$:

$$\underline{a_k} = \frac{1}{n} \cdot \underline{q(\omega_n^{-k})} = \frac{1}{n} \cdot q(\omega_n^{n-k})$$

DFT:

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

$$\underline{y_k} = \underline{p(\omega_n^k)}$$

DFT and Inverse DFT

$$q(x) = \underline{y_0} + \underline{y_1}x + \cdots + \underline{y_{n-1}}x^{n-1}, \quad \underline{a_k = \frac{1}{n} \cdot q(\omega_n^{-k})}:$$

- Therefore:

$$\begin{aligned} & \underline{(a_0, a_1, \dots, a_{n-1})} \\ &= \frac{1}{n} \cdot \left(q(\omega_n^{-0}), q(\omega_n^{-1}), q(\omega_n^{-2}), \dots, q(\omega_n^{-(n-1)}) \right) \\ &= \frac{1}{n} \cdot \left(\underline{q(\omega_n^0)}, \underline{q(\omega_n^{n-1})}, q(\omega_n^{n-2}), \dots, q(\omega_n^1) \right) \end{aligned}$$

- Recall:

$$\begin{aligned} \underline{\text{DFT}_n(\mathbf{y})} &= \underline{(q(\omega_n^0), q(\omega_n^1), q(\omega_n^2), \dots, q(\omega_n^{n-1}))} \\ &= \underline{n \cdot (a_0, a_{n-1}, a_{n-2}, \dots, a_2, a_1)} \end{aligned}$$

DFT and Inverse DFT

- We have $\text{DFT}_n(\mathbf{y}) = n \cdot (a_0, a_{n-1}, a_{n-2}, \dots, a_2, a_1)$:

$$\underline{a_i} = \begin{cases} \frac{1}{n} \cdot \underline{(\text{DFT}_n(\mathbf{y}))_0} & \text{if } i = 0 \\ \frac{1}{n} \cdot (\text{DFT}_n(\mathbf{y}))_{n-i} & \text{if } i \neq 0 \end{cases}$$

- DFT and inverse DFT can both be computed using FFT algorithm in $O(n \log n)$ time.
- 2 polynomials of degr. $< n$ can be multiplied in time $O(n \log n)$.

Faster Polynomial Multiplication?

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

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



Evaluation at $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$ using **FFT** $\mathcal{O}(n \log n)$

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



Point-wise multiplication $\mathcal{O}(n)$

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



Interpolation using **FFT** $\mathcal{O}(n \log n)$

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

Convolution

- More generally, the polynomial multiplication algorithm computes the convolution of two vectors:

$$\mathbf{a} = (a_0, a_1, \dots, a_{m-1})$$

$$\mathbf{b} = (b_0, b_1, \dots, b_{n-1})$$

$$\mathbf{a} * \mathbf{b} = (c_0, c_1, \dots, c_{m+n-2}),$$

$$\text{where } c_k = \sum_{\substack{(i,j):i+j=k \\ i < m, j < n}} a_i b_j$$

- c_k is exactly the coefficient of x^k in the product polynomial of the polynomials defined by the coefficient vectors \mathbf{a} and \mathbf{b}

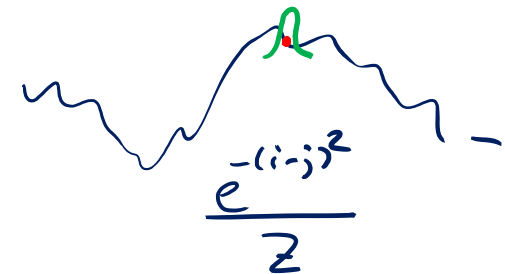
More Applications of Convolutions

Signal Processing Example:



- Assume $\mathbf{a} = (a_0, \dots, a_{n-1})$ represents a sequence of measurements over time
- Measurements might be noisy and have to be smoothed out
- Replace a_i by weighted average of nearby last m and next m measurements (e.g., Gaussian smoothing):

$$\underline{a'_i} = \frac{1}{Z} \cdot \sum_{j=i-m}^{i+m} \underline{a_j} \frac{e^{-(i-j)^2}}{Z}$$



- New vector \mathbf{a}' is the convolution of \mathbf{a} and the weight vector $\frac{1}{Z} \cdot (e^{-m^2}, e^{-(m-1)^2}, \dots, e^{-1}, 1, e^{-1}, \dots, e^{-(m-1)^2}, e^{-m^2})$
- Might need to take care of boundary points...

More Applications of Convolutions

Combining Histograms:

- Vectors \mathbf{a} and \mathbf{b} represent two histograms
- E.g., annual income of all men & annual income of all women
- Goal: Get new histogram \mathbf{c} representing combined income of all possible pairs of men and women:

$$\mathbf{c} = \mathbf{a} * \mathbf{b}$$

Also, the DFT (and thus the FFT alg.) has many other applications!

DFT in Signal Processing

Assume that $\underline{y(0), y(1), y(2), \dots, y(T - 1)}$ are measurements of a time-dependent signal.

Inverse DFT_N of $(y(0), \dots, y(T - 1))$ is a vector $\underline{(c_0, \dots, c_{N-1})}$ s.t.

$$\underline{y(t)} = \sum_{k=0}^{N-1} c_k \cdot e^{\frac{2\pi i \cdot k}{N} \cdot t}$$

$c_0 + c_1 x + c_2 x^2 + \dots$
 $x = \omega_N^i$

$$= \sum_{k=0}^{T-1} c_k \cdot \left(\cos\left(\frac{2\pi \cdot k}{N} \cdot t\right) + i \sin\left(\frac{2\pi \cdot k}{N} \cdot t\right) \right)$$

$e^{i\varphi} = \cos\varphi + i \sin\varphi$

- Converts signal from time domain to frequency domain
- Signal can then be edited in the frequency domain
 - e.g., setting some $c_k = 0$ filters out some frequencies