Chapter 1
Divide and Conquer

Part 2: Polynomial Multiplication

Algorithm Theory
WS 2012/13

Fabian Kuhn

Representation of Polynomials

Coefficient representation:

- Polynomial $p(x) \in \mathbb{R}[x]$ of degree n is given by its $n+1$ coefficients $a_0, \dots, a_n$:

$$p(x) = a_n x^n + \cdots + a_1 x + a_0$$

- Example:

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

- The most typical (and probably most natural) representation of polynomials

Representation of Polynomials

Point-value representation:

- Polynomial $p(x) \in \mathbb{R}[x]$ of degree n is given by $n+1$ point-value pairs:

$$p = \{(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_n, p(x_n))\}$$

where $x_i \neq x_j$ for $i \neq j$.

- Example: The polynomial

$$p(x) = 3x(x-2)(x-3)$$

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

Operations: Coefficient Representation

Deg.-n polynomials $p(x) = a_n x^n + \cdots + a_0$, $q(x) = b_n x^n + \cdots + b_0$

Addition:

$$p(x) + q(x) = (a_n + b_n)x^n + \cdots + (a_0 + b_0)$$

- Time: $O(n)$

Multiplication:

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

- Naive solution: Need to compute product $a_i b_j$ for all $0 \leq i, j \leq n$
- Time: $O(n^2)$

Operations Point-Value Representation

Degree-n polynomials

$$p = \{(x_0, p(x_0)), \dots, (x_n, p(x_n))\}, q = \{(x_0, q(x_0)), \dots, (x_n, q(x_n))\}$$

- Note: we use the same points $x_0, \dots, x_n$ for both polynomials

Addition:

$$p + q = \{(x_0, p(x_0) + q(x_0)), \dots, (x_n, p(x_n) + q(x_n))\}$$

- Time: $O(n)$

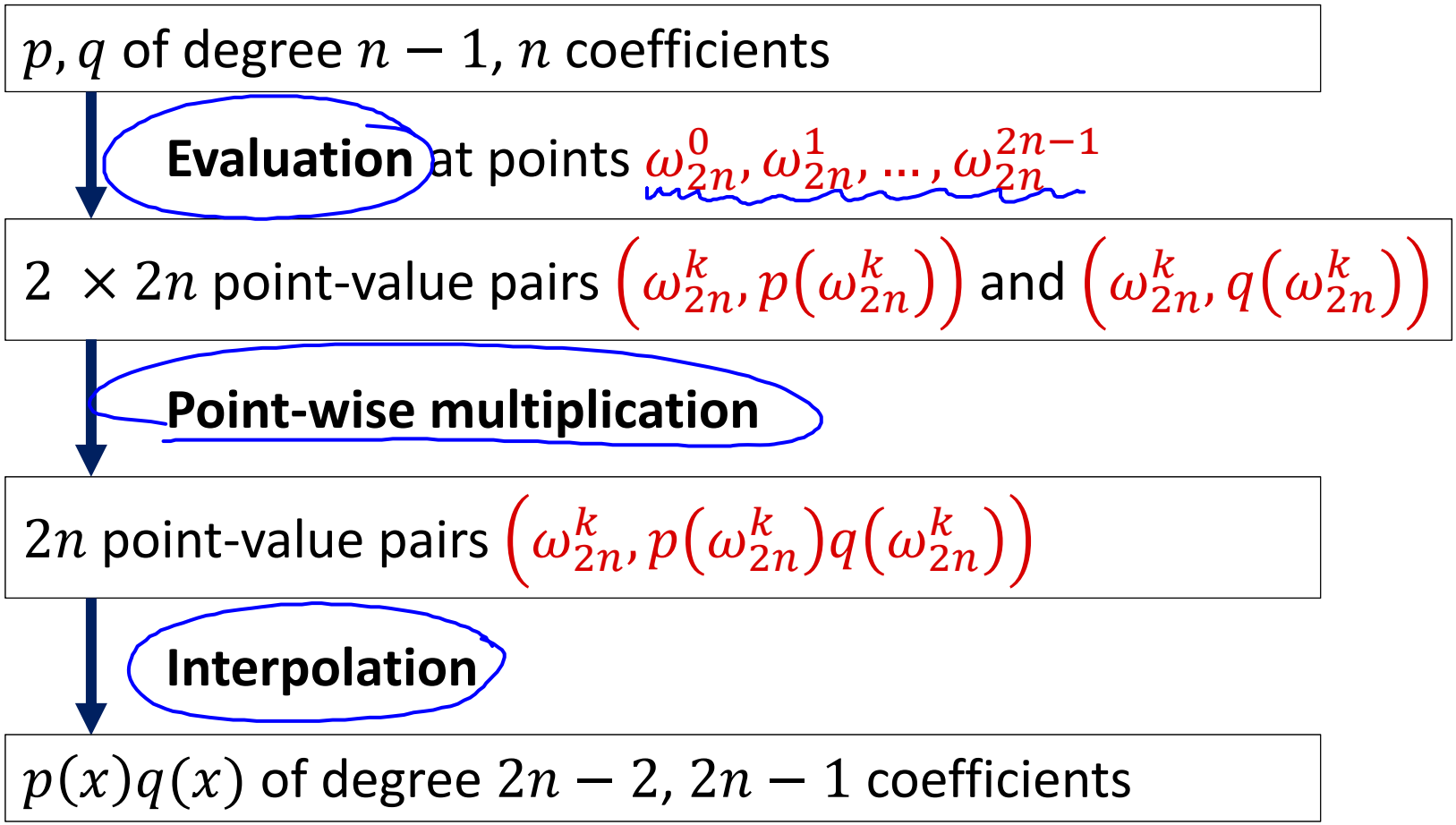
Multiplication:

$$p \cdot q = \{(x_0, p(x_0) \cdot q(x_0)), \dots, (x_n, p(x_n) \cdot q(x_n))\}$$

- Time: $O(n)$

Faster Polynomial Multiplication?

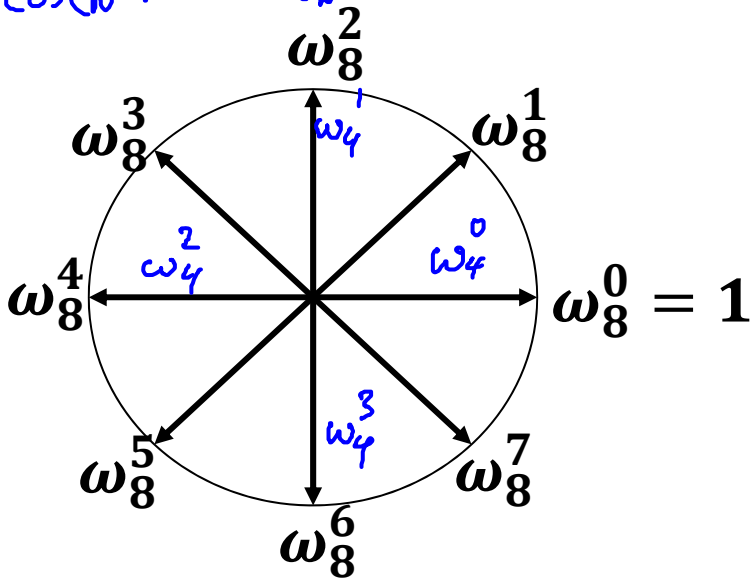
Idea to compute $p(x) \cdot q(x)$ (for polynomials of degree $< n$):



Properties of the Roots of Unity

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

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



Powers of ω_n (roots of unity):

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

Cancellation Lemma:

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

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

$\omega_n^n = 1$

Discrete Fourier Transform

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

Discrete Fourier Transform (DFT):

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

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

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

Computing the DFT

Polynomial p of degree $N-1$:

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

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

where

divide:

$$p_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{N-2} x^{N/2-1} \quad \text{(even coeff.)}$$

$$p_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{N-1} x^{N/2-1} \quad \text{(odd coeff.)}$$

Requires computing $p_0(\omega_{N/2}^k)$ and $\omega_N^k \cdot p_1(\omega_{N/2}^k)$ for $0 \leq k < N/2$.

Fast Fourier Transform (FFT) Algorithm

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; // $a = [a_0]$

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

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

compute $\mathrm{DFT}_{N/2}(a_0)$, $\mathrm{DFT}_{N/2}(a_1)$ recursively

$d^{[0]} = (P_0(\omega_{N/2}^0), P_0(\omega_{N/2}^1), \dots, P_0(\omega_{N/2}^{N/2-1}))$

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

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

$d_k = P(\omega_N^k)$

Example

$p_0(x) = -15x + 0$ $\quad p_1(x) = 3x + 18$

$\omega_2^0 = 1$, $\omega_2^1 = -1$

- $p(x) = 3x^3 - 15x^2 + 18x + 0$, $a = [0, 18, -15, 3]$

$\omega_4^0 = 1$
$\omega_4^1 = i$ ←
$\omega_4^2 = -1$
$\omega_4^3 = -i$

$DFT_4(0, 18, -15, 3)$

$DFT_2(0, -15)$

$p_0(1) = -15$, $p_0(-1) = 15$

$DFT_2(0, -15) = (-15, 15)$

$DFT_2(18, 3)$

$p_1(1) = 21$, $p_1(-1) = 15$

$DFT_2(18, 3) = (21, 15)$

$$DFT_4(a) = (p(\omega_4^0), p(\omega_4^1), p(\omega_4^2), p(\omega_4^3))$$

$$= (p_0(\omega_2^0) + 1 \cdot p_1(\omega_2^0), p_0(\omega_2^1) + i \cdot p_1(\omega_2^1),$$
$$p_0(\omega_2^0) - p_1(\omega_2^0), p_0(\omega_2^1) - i \cdot p_1(\omega_2^1))$$

$$= (-15 + 21, 15 + i \cdot 15, -15 - 21, 15 - i \cdot 15)$$

$$= (6, 15 + i \cdot 15, -36, 15 - i \cdot 15)$$

$DFT_4(0, 18, -15, 3)$

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** $O(n \log n)$

$2 \times 2n$ 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** $O(n)$

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

↓ **Interpolation** ?

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

Interpolation

Convert point-value representation into coefficient representation

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

Output:

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

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

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

Interpolation

$x_i = \omega_n^i$

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 $x_i = \omega_n^i$:

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

Interpolation

- Linear system:

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

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

row i, col. j of W^{-1}

Claim:

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

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

DFT Matrix Inverse

$W_{ij} = \omega_n^{ij}$ claim: $W_{ij}^{-1} = \frac{\omega^{-ij}}{n}$

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

$$(W^{-1}W)_{ij} = \frac{1}{n}\cdot 1 + \frac{\omega_n^{-i}\omega_n^{j}}{n} + \frac{\omega_n^{-2i}\omega_n^{2j}}{n} + \cdots + \frac{\omega_n^{-(n-1)i}\omega_n^{(n-1)j}}{n}$$

$$= \frac{1}{n}\left(1 + \omega_n^{j-i} + \omega_n^{2(j-i)} + \cdots + \omega_n^{(n-1)(j-i)}\right)$$

$$= \frac{1}{n}\cdot\sum_{k=0}^{n-1}\omega_n^{k(j-i)}$$

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 $i = j$:

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

✓

DFT Matrix Inverse

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

Case $i \neq j$:

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

$$= \frac{1}{n} \sum_{\ell=0}^{n-1} q^\ell = \frac{1}{n} \cdot 1 \cdot \frac{q^n - 1}{q-1}$$

$$= \frac{1}{n} \cdot \frac{\omega_n^{n(i-j)} - 1}{\omega_n^{i-j} - 1} = \frac{1}{n} \cdot \frac{0}{\omega_n^{i-j} - 1} = 0$$

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

✓

Inverse DFT

- $W^{-1} = \begin{pmatrix} \frac{1}{n} & \frac{\omega_n^{-k}}{n} & \cdots & \frac{\omega_n^{-(n-1)k}}{n} \\ & & \vdots & \\ & & \cdots & \end{pmatrix}$ ← row k
- We get $\boldsymbol{a} = W^{-1} \cdot \boldsymbol{y}$ and therefore

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

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

$q(\omega_n^{-k})$

$y_k = p(\omega_n^k) = \sum_{j=0}^{n-1} a_j \cdot \omega_n^{kj}$

$q(x) = y_0 + y_1 x + \ldots + y_{n-1} x^{n-1}$

DFT and Inverse DFT

Inverse DFT:

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

- Define polynomial $q(x) = y_0 + y_1 x + \cdots + y_{n-1} x^{n-1}$:

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

DFT:

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

$$y_k = p(\omega_n^k)$$

DFT and Inverse DFT

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

- Therefore:

$\omega_n^k = \omega_n^{k+n}$

$$(a_0, a_1, \dots, a_{n-1})$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^{-0}), q(\omega_n^{-1}), q(\omega_n^{-2}), \dots, q\left(\omega_n^{-(n-1)}\right)\right)$$

$$= \frac{1}{n} \cdot \left(q(\omega_n^0), q(\omega_n^{n-1}), q(\omega_n^{n-2}), \dots, q(\omega_n^1)\right)$$

- Recall:

$$\mathrm{DFT}_n(\boldsymbol{y}) = \left(q(\omega_n^0), q(\omega_n^1), q(\omega_n^2), \dots, q(\omega_n^{n-1})\right)$$

$$= n \cdot (a_0, a_{n-1}, a_{n-2}, \dots, a_2, a_1)$$

DFT and Inverse DFT

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

$$a_i = \begin{cases} (\mathrm{DFT}_n(\boldsymbol{y}))_0 & \text{if } i = 0 \\ (\mathrm{DFT}_n(\boldsymbol{y}))_{n-i} & \text{if } i \neq 0 \end{cases}$$

- 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** $O(n \log n)$

$2 \times 2n$ 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** $O(n)$

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

↓ **Interpolation** using **FFT** $O(n \log n)$

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

Convolution

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

$$\boldsymbol{a} = (a_0, a_1, \dots, a_{m-1})$$
$$\boldsymbol{b} = (b_0, b_1, \dots, b_{n-1})$$

$$\boldsymbol{a} * \boldsymbol{b} = (c_0, c_1, \dots, c_{m+n-2}),$$
$$\text{where } c_k = \sum_{\substack{(i,j): i+j=k \\ i<m, j<n}} a_i b_j$$

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

More Applications of Convolutions

Signal Processing Example:

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

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

- New vector $\boldsymbol{a}'$ is the convolution of $\boldsymbol{a}$ and the weight vector

$$\frac{1}{Z} \cdot \left(e^{-m^2}, e^{-(m-1)^2}, \dots, e^{-1}, 1, e^{-1}, \dots, e^{-(m-1)^2}, e^{-m^2}\right)$$

- Might need to take care of boundary points…

More Applications of Convolutions

Combining Histograms:

- Vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ represent two histograms
- E.g., annual income of all men & annual income of all women

- Goal: Get new histogram $\boldsymbol{c}$ representing combined income of all possible pairs of men and women:

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

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