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 $< n$):

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

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

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

↓ **Point-wise multiplication**

$2n$ 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

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

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

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

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

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

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

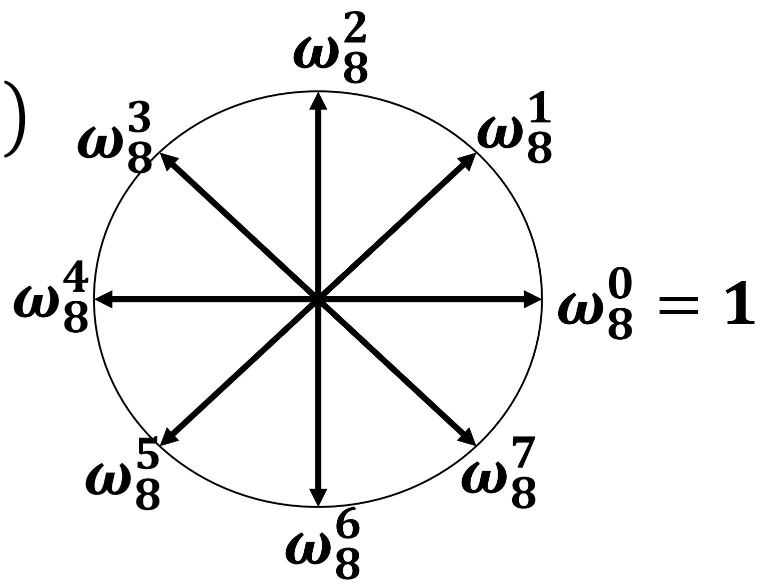
Choice of X

- 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: $\boldsymbol{\omega_N = e^{2\pi i/N}}$

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



Powers of $\boldsymbol{\omega_n}$ (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 $X = \left\{\omega_{2k}^{i} : i \in \{0, \dots, 2k-1\}\right\}$, we have

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

Analysis

New recurrence formula:

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

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 points $\omega_{2n}^0, \omega_{2n}^1, \dots, \omega_{2n}^{2n-1}$

$2 \times 2n$ 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

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

Interpolation

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

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):

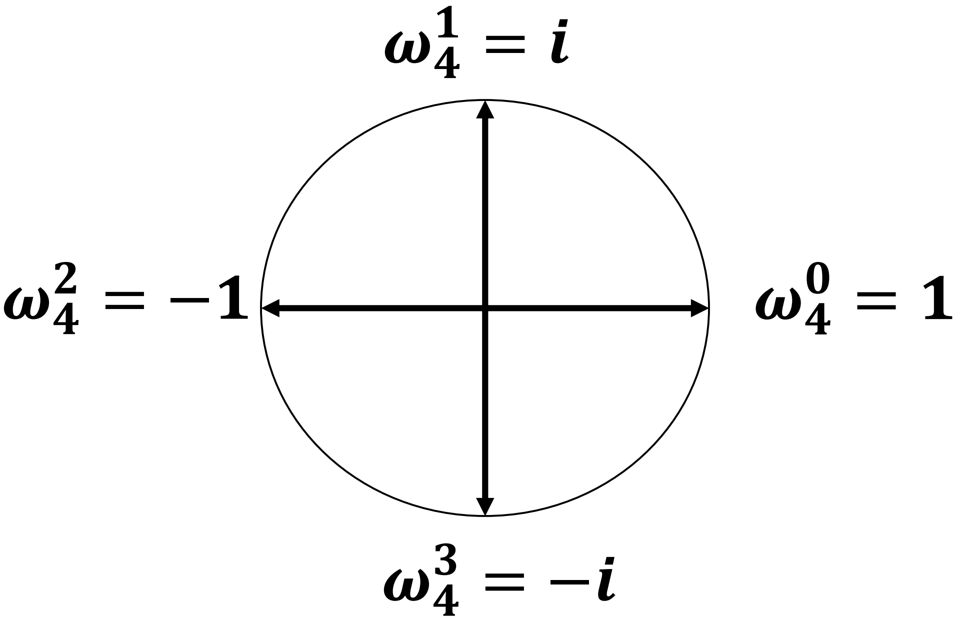
- Assume $a = (a_0, \dots, a_{N-1})$ is the coefficient vector of poly. p

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

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

Example

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

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

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

$$\mathbf{DFT_4}(\boldsymbol{a}) = (\mathbf{6}, \mathbf{15} + \mathbf{15}\boldsymbol{i}, -\mathbf{36}, \mathbf{15} - \mathbf{15}\boldsymbol{i})$$

DFT: Recursive Structure

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

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

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

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

$$\mathrm{DFT}_N(a) = \left(p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right), p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$
$$+\left(\omega_N^0 p_0(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_0\left(\omega_{N/2}^{N/2-1}\right), \omega_N^{N/2} p_0(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$

Example

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

$$\mathrm{DFT}_N(a) = \left(p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right), p_0(\omega_{N/2}^0), \dots, p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$
$$+\left(\omega_N^0 p_0(\omega_{N/2}^0), \dots, \omega_N^{N/2-1} p_0\left(\omega_{N/2}^{N/2-1}\right), \omega_N^{N/2} p_0(\omega_{N/2}^0), \dots, \omega_N^{N-1} p_0\left(\omega_{N/2}^{N/2-1}\right)\right)$$

$N = 4$:

$$p(\omega_4^0) = p_0(\omega_2^0) + \omega_4^0 p_1(\omega_2^0)$$
$$p(\omega_4^1) = p_0(\omega_2^1) + \omega_4^1 p_1(\omega_2^1)$$
$$p(\omega_4^2) = p_0(\omega_2^0) + \omega_4^2 p_1(\omega_2^0)$$
$$p(\omega_4^3) = p_0(\omega_2^1) + \omega_4^3 p_1(\omega_2^1)$$

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

(DFTs of coefficient vectors of p_0 and p_1)

Summary: Computation of DFT_N

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

1. Divide

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

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

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 $\mathrm{DFT}_{N/2}(p_1)$

Small Improvement

Polynomial p of degree $N-1$:

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

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

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 $N = 8$

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

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

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

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

$$p(\omega_8^3) = p_0(\omega_4^0) - \omega_8^0 \cdot p_1(\omega_4^0)$$

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

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

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

Fast Fourier Transform (FFT) Algorithm

Algorithm FFT(a)

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

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

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

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

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

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

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

$\quad d_k \coloneqq d_k^{[0]} + x$; $d_{k+N/2} \coloneqq d_k^{[0]} - x$;

$\quad \omega \coloneqq \omega \cdot \omega_N$

end;

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

Example

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

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**

$2 \times 2n$ 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

$2n$ 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: $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ 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

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

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

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 $\boldsymbol{x_i = \omega_n^i}$:

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

Interpolation

- Linear system:

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

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

Claim:

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

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

DFT Matrix Inverse

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

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 $\boldsymbol{i = j}$:

DFT Matrix Inverse

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

Case $i \neq j$:

Inverse DFT

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

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

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

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

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

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

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

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

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

- 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**

$2 \times 2n$ 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**

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

↓ **Interpolation** using **FFT**

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

Convolution

- More generally, the 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 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):

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

- 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!

DFT in Signal Processing

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

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

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

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

- 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