Chapter 1
Divide and Conquer

Algorithm Theory
WS 2017/18

Fabian Kuhn

Divide-And-Conquer Principle

- Important algorithm design method

- Examples from basic alg. & data structures class (Informatik 2):
  - Sorting: Mergesort, Quicksort
  - Binary search

- Further examples
  - Median
  - Compairing orders
  - Convex hull / Delaunay triangulation / Voronoi diagram
  - Closest pairs
  - Line intersections
  - Polynomial multiplication / FFT
  - ...

$O(n \log n)$

Formulation of the D&C principle

Divide-and-conquer method for solving a problem instance of size n:

1. Divide

$n \leq c$: Solve the problem directly.

$n > c$: Divide the problem into k subproblems of sizes $n_1, \dots, n_k < n$ ($k \geq 2$).

2. Conquer

Solve the k subproblems in the same way (recursively).

recursion

3. Combine

Combine the partial solutions to generate a solution for the original instance.

Running Time

Recurrence relation:

$$T(n) = 2 \cdot T(n/2) + c \cdot n, \qquad T(1) = a$$

divide + combine

Solution:

- Same as for computing number of number of inversions, merge sort (and many others…)

$$T(n) = O(n \cdot \log n)$$

Recurrence Relations: Master Theorem

Recurrence relation

$$\boldsymbol{T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \qquad T(n) = O(1) \text{ for } n \leq n_0}$$

Cases

- $f(n) = O(n^c),\ c < \log_b a$

$$\boldsymbol{T(n) = \Theta\left(n^{\log_b a}\right)}$$

- $f(n) = \Omega(n^c),\ c > \log_b a$

$$\boldsymbol{T(n) = \Theta\left(f(n)\right)}$$

- $f(n) = \Theta\left(n^c \cdot \log^k n\right),\ c = \log_b a$

$$\boldsymbol{T(n) = \Theta\left(n^c \cdot \log^{k+1} n\right)}$$

Polynomials

Real polynomial p in one variable x: $a_i \in \mathbb{R}$

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

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)

Divide-&-Conquer Polynomial Multiplication

- Multiplication is slow $\left(\Theta(n^2)\right)$ when using the standard coefficient representation
- 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: $q(x) = q_1(x) \cdot x^{n/2} + q_0(x)$

Divide-&-Conquer Polynomial Multiplication

- **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:
  $$T(n) = 4T(n/2) + O(n)$$

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

Karatsuba Algorithm

- Recursive multiplication:

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

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

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

$$T(n) = 3T\left(n/2\right) + O(n)$$

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

$\Downarrow$

$O(n \log n)$

Representation of Polynomials

Coefficient representation:

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

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

- Coefficient vector $\boldsymbol{a} = (a_0, a_1, \dots a_{n-1})$

- 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 $n-1$ is given by n point-value pairs:

$$p = \{(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{n-1}, p(x_{n-1}))\}$$

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

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

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:

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

- Naive solution: Need to compute product $a_i b_j$ for all $0 \leq i, j \leq n$
- Time: $O(n^2)$ $\quad O(n^{1.58\ldots})$

Operations: Point-Value Representation

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

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

Addition:

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

- Time: $O(n)$

Multiplication:

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

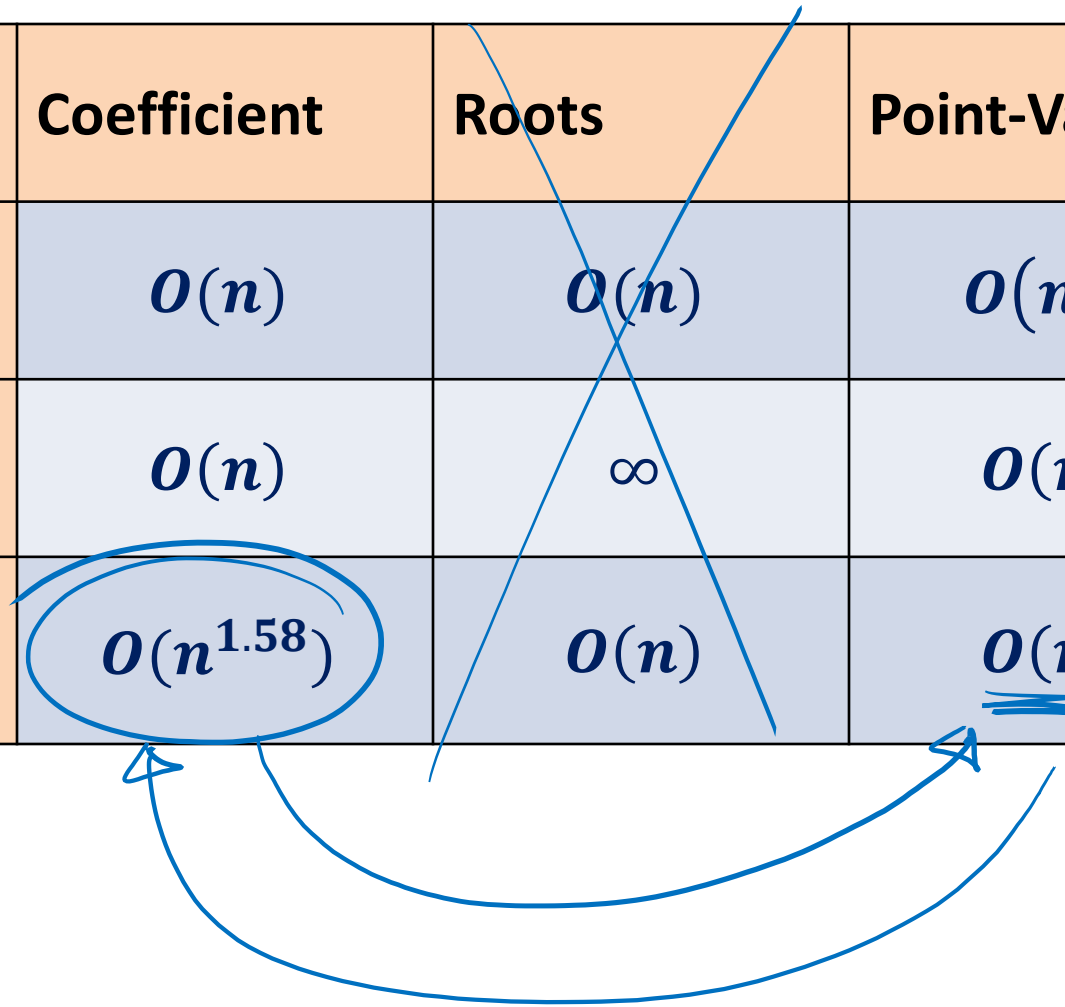
- Time: $O(n)$

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

Operations on Polynomials

Cost depending on representation:

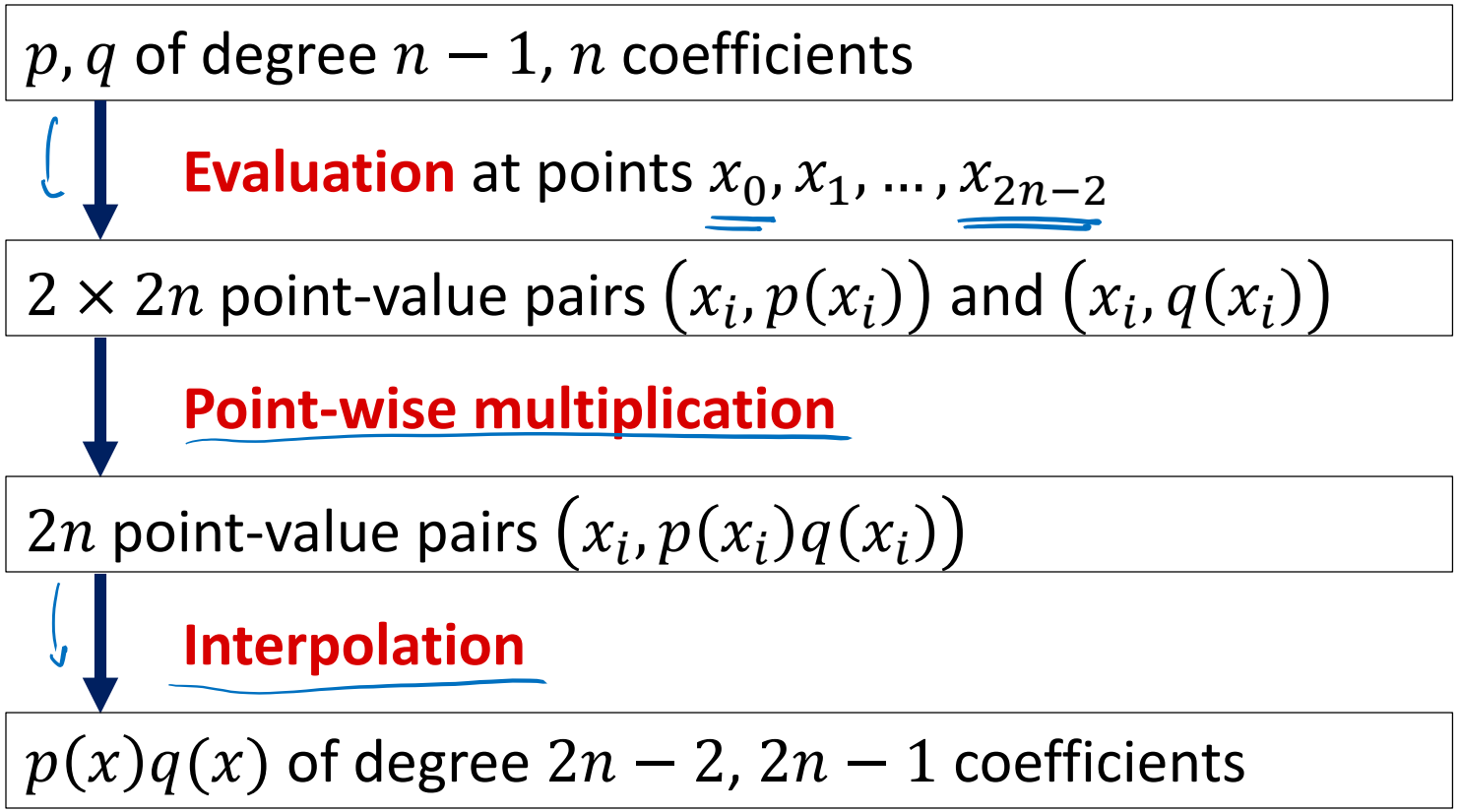
	Coefficient	Roots	Point-Value
Evaluation	$O(n)$	$O(n)$	$O(n^2)$
Addition	$O(n)$	∞	$O(n)$
Multiplication	$O(n^{1.58})$	$O(n)$	$O(n)$



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

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

$\{(x, p(x)) : x \in X\}$

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.)

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

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 $|X^2| = |X|$

Analysis

$|X^2| \leq |X|$

Recurrence formula for the given algorithm:

$$T(n, |X|) = 2 \cdot T(n/2, |X^2|) + O(n + |X|)$$

$$\leq 2 \cdot T(n/2, |X|) + O(n + |X|)$$

at start: $|X| = \Theta(n)$

$$T(n, n) = O(n^2)$$

Faster Algorithm?

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

best we can hope for: $|X^2| = \frac{|X|}{2}$

$T(n,|X|) = 2\,T(\frac{n}{2}, |X^2|) + O(n+|X|)$

$= 2\,T(\frac{n}{2}, \frac{|X|}{2}) + O(n+|X|)$

$T'(n/2)$

$|X| = n$

$\Rightarrow T(n) = 2\,T(n/2) + O(n)$

$\Rightarrow T(n) = O(n \log n)$

$T(n,|X|) = T'(n)$

$\{1\}$

$\{-1, 1\}$

$\{-1, 1, -i, i\}$

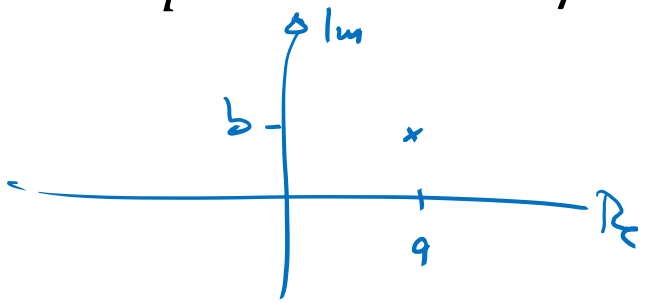
$\{-1, 1, -i, i, \frac{1}{\sqrt{2}}+\frac{i}{\sqrt{2}}, \frac{1}{\sqrt{2}}-\frac{i}{\sqrt{2}}, -\frac{1}{\sqrt{2}}+\frac{i}{\sqrt{2}}, -\frac{1}{\sqrt{2}}-\frac{i}{\sqrt{2}}\}$

$\left(\frac{1}{\sqrt{2}}+\frac{i}{\sqrt{2}}\right)^2 = \frac{1}{2} - \frac{1}{2} + \frac{2i}{2} = i$

Choice of X

$a + b \cdot i$

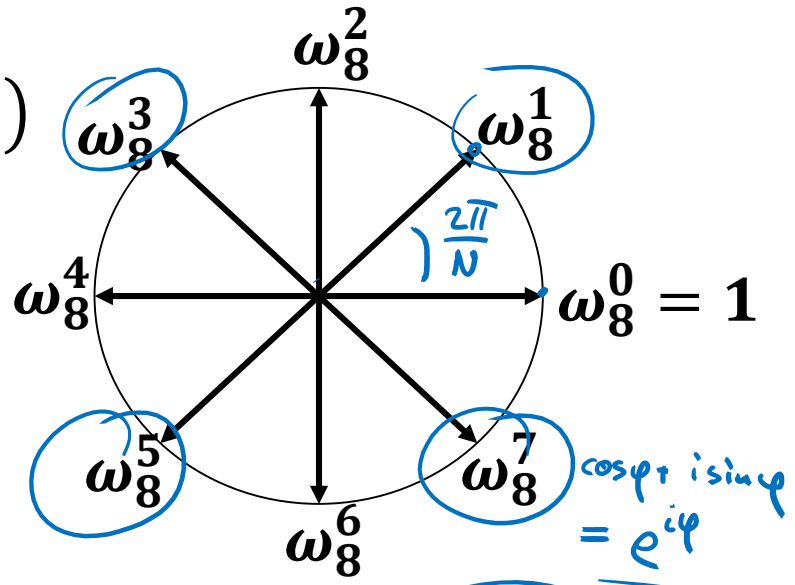
- Select points $x_0, x_1, \dots, x_{N-1}$ to evaluate p and q in a clever way



Consider the N complex roots of unity:

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

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



Powers of ω_n (roots of unity):

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

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

Properties of the Roots of Unity

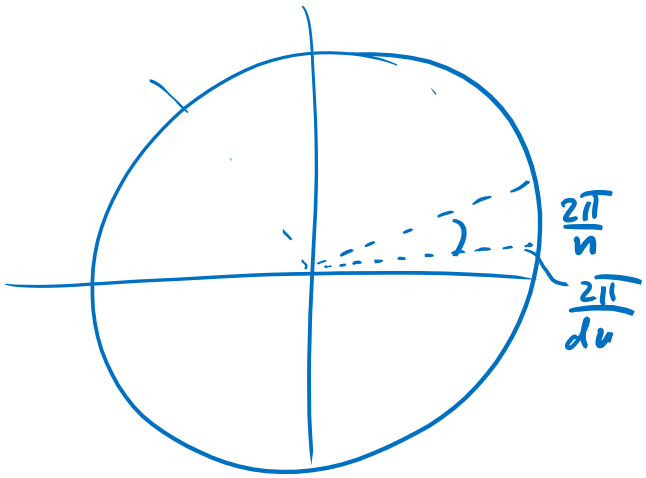
- **Cancellation Lemma:**

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

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

- **Proof:**

$$\omega_n^k = e^{\frac{2\pi i}{n} \cdot k} = e^{\frac{2\pi i}{d \cdot n} d \cdot k}$$



Properties of the Roots of Unity

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

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

$|X| = 2k$

$\Downarrow$

$|X^2| = k$

$x \in X \to x^2 \in X^2$

$(\omega_{2k}^i)^2 = \omega_{2k}^{2i} = \omega_k^i$

Analysis

New recurrence formula:

$$T(N, |X|) \leq 2 \cdot T\left(N/2, |X|/2\right) + O(N + |X|)$$

$\hookrightarrow O(N \log N)$

if $|X| = N$ (or $|X| = O(N)$)

Faster Polynomial Multiplication?

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

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

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

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

$O(n)$ ↓ **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

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):

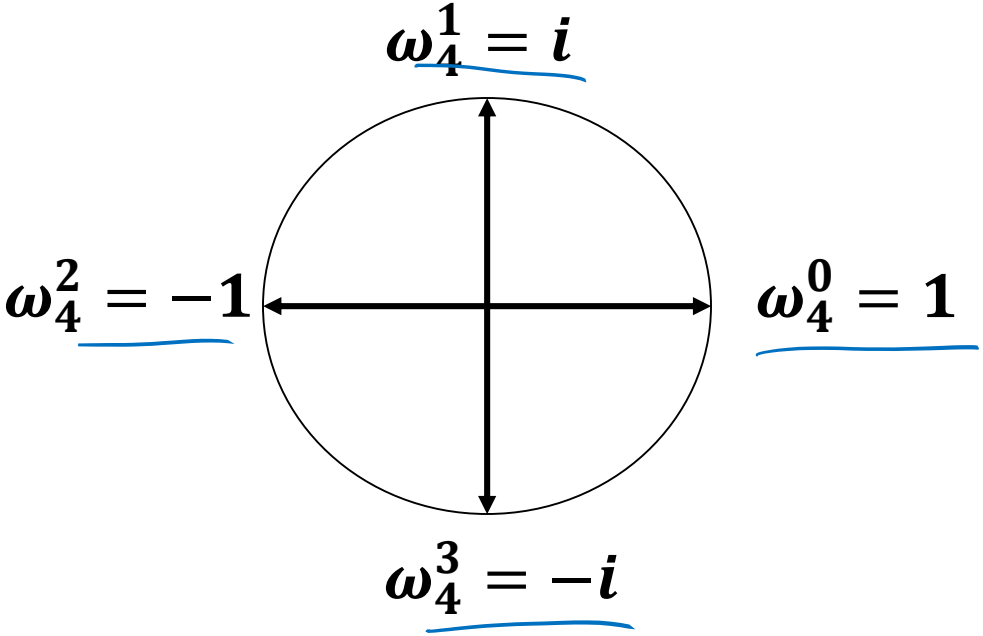
- Assume $a = (a_0, \dots, a_{N-1})$ is the coefficient vector of poly. p

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

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

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

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

DFT: Recursive Structure

Evaluation for $k = 0, \dots, N-1$: $p(x) = p_0(x^2) + x\,p_1(x^2)$

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

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

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

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

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; // $a = [a_0]$

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

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

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

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

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

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: $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ 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 $a_0, \dots, a_{n-1}$