Polynomials and the Fast Fourier Transform (FFT)

Algorithm Design and Analysis
(Week 7)

Battle Plan

- Polynomials
  - Algorithms to add, multiply and evaluate polynomials
  - Coefficient and point-value representation
- Fourier Transform
  - Discrete Fourier Transform (DFT) and inverse DFT to translate between polynomial representations
  - "A Short Digression on *Complex Roots of Unity*"
  - Fast Fourier Transform (FFT) is a divide-and-conquer algorithm based on properties of complex roots of unity

Polynomials

- A **polynomial** in the variable x is a representation of a function
  $$A(x) = a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$$
  as a formal sum $A(x) = \sum_{j=0}^{n-1} a_j x^j$.
- We call the values $a_0, a_{1,} \dots, a_{n-1}$ the **coefficients** of the polynomial
- $A(x)$ is said to have **degree** k if its highest nonzero coefficient is a_k.
- Any integer strictly greater than the degree of a polynomial is a **degree-bound** of that polynomial

Examples

- $A(x) = x^3 - 2x - 1$
  - $A(x)$ has degree 3
  - $A(x)$ has degree-bounds $4, 5, 6, \dots$ or all values $>$ degree
  - $A(x)$ has coefficients $(-1, -2, 0, 1)$
- $B(x) = \ x^3 + x^2 + 1$
  - $B(x)$ has degree 3
  - $B(x)$ has degree bounds $4, 5, 6, \dots$ or all values $>$ degree
  - $B(x)$ has coefficients $(1, 0, 1, 1)$

Coefficient Representation

- A **coefficient representation** of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree-bound n is a vector of coefficients $a = (a_0, a_1, \dots, a_{n-1})$.
- More examples
  - $A(x) = 6x^3 + 7x^2 - 10x + 9$ $(9, -10, 7, 6)$
  - $B(x) = -2x^3 + 4x - 5$ $(-5, 4, 0, -2)$
- The operation of **evaluating** the polynomial $A(x)$ at point x_0 consists of computing the value of $A(x_0)$.
- Evaluation takes time $\Theta(n)$ using Horner's rule

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))\cdots))$$

Adding Polynomials

- **Adding** two polynomials represented by the coefficient vectors $a = (a_0, a_1, \dots, a_{n-1})$ and $b = (b_0, b_1, \dots, b_{n-1})$ takes time $\Theta(n)$.
- Sum is the coefficient vector $c = (c_0, c_1, \dots, c_{n-1})$, where $c_j = a_j + b_j$ for $j = 0,1, \dots, n-1$.
- Example

$$\begin{array}{rrrrrl}
A(x) = & 6x^3 & + 7x^2 & - 10x & + 9 & (9, -10, 7, 6) \\
B(x) = & -2x^3 & & + 4x & - 5 & (-5, 4, 0, -2) \\
\hline
C(x) = & 4x^3 & + 7x^2 & - 6x & + 4 & (4, -6, 7, 4)
\end{array}$$

Multiplying Polynomials

- For **polynomial multiplication**, if $A(x)$ and $B(x)$ are polynomials of degree-bound n, we say their **product** $C(x)$ is a polynomial of degree-bound $2n-1$.
- Example

$$
\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -30x^3 & -35x^2 & +50x & -\ 45 \\
 & & 24x^4 & +28x^3 & -40x^2 & +36x & \\
-12x^6 & -14x^5 & +20x^4 & -18x^3 & & & \\
\hline
-12x^6 & -14x^5 & +44x^4 & -20x^3 & -75x^2 & +86x & -\ 45
\end{array}
$$

Multiplying Polynomials

- **Multiplication** of two degree-bound n polynomials $A(x)$ and $B(x)$ takes time $\Theta(n^2)$, since each coefficient in vector a must be multiplied by each coefficient in vector b.
- Another way to express the product C(x) is $\sum_{j=0}^{2n-1} c_j\, x^j$, where $c_j = \sum_{k=0}^{j} a_k b_{j-k}$.
- The resulting coefficient vector $c\ =\ (c_0, c_1, \dots c_{2n-1})$ is also called the **convolution** of the input vectors a and b, denoted as $c = a \otimes b$.

Point-Value Representation

- A **point-value representation** of a polynomial $A(x)$ of degree-bound n is a set of n **point-value pairs**
$$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$
such that all of the x_k are distinct and $y_k = A(x_k)$ for $k = 0, 1, \dots, n-1$.
- Example $A(x) = x^3 - 2x + 1$

$$\left.\begin{array}{ll} x_k & 0, 1, 2, 3 \\ A(x_k) & 1, 0, 5, 22 \end{array}\right\} \{(0,1), (1,0), (2,5), (3,22)\}$$

- Using Horner's method, ***n*-point evaluation** takes time $\Theta(n^2)$.

Point-Value Representation

- The inverse of evaluation is called **interpolation**
  - determines coefficient form of polynomial from point-value representation
  - For any set $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ of n point-value pairs such that all the x_k values are distinct, there is a **unique** polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \dots, n-1$.
- Lagrange's formula

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

- Using Lagrange's formula, interpolation takes time $\Theta(n^2)$.

Example

- Using Lagrange's formula, we interpolate the point-value representation $\{(0,1),(1,0),(2,5),(3,22)\}$.
  - $1\frac{(x-1)(x-2)(x-3)}{(0-1)(0-2)(0-3)} = \frac{x^3-6x^2+11x-6}{-6} = \frac{-x^3+6x^2-11x+6}{6}$
  - $0\frac{(x-0)(x-2)(x-3)}{(1-0)(1-2)(1-3)} = 0$
  - $5\frac{(x-0)(x-1)(x-3)}{(2-0)(2-1)(2-3)} = 5\frac{x^3-4x^2+3x}{-2} = \frac{-15x^3+60x^2-45x}{6}$
  - $22\frac{(x-0)(x-1)(x-2)}{(3-0)(3-1)(3-2)} = 22\frac{x^3-3x^2+2x}{6} = \frac{22x^3-66x^2+44x}{6}$
  - $\frac{1}{6}(6x^3+0x^2-12x+6)$
  - x^3-2x+1

Adding Polynomials

- In point-value form, addition $C(x) = A(x) + B(x)$ is given by $C(x_k) = A(x_k) + B(x_k)$ for any point x_k.
  - $A: \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$
  - $B: \{(x_0, y_0'), (x_1, y_1'), \dots, (x_{n-1}, y_{n-1}')\}$
  - $C: \{(x_0, y_0 + y_0'), (x_1, y_1 + y_1'), \dots, (x_{n-1}, y_{n-1} + y_{n-1}')\}$
- A and B are evaluated for the **same** n points.
- The time to add two polynomials of degree-bound n in point-value form is $\Theta(n)$.

Example

- We add $C(x) = A(x) + B(x)$ in point-value form
  - $A(x) = x^3 - 2x + 1$
  - $B(x) = x^3 + x^2 + 1$
  - $x_k = (0, 1, 2, 3)$
  - A: $\{(0,1), (1,0), (2, 5), (3,22)\}$
  - B: $\{(0,1), (1,3), (2,13), (3,37)\}$
  - C: $\{(0,2), (1,3), (2,18), (3,59)\}$

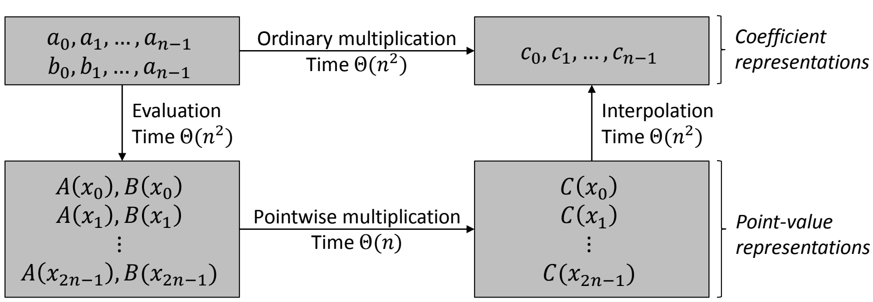
Multiplying Polynomials

- In point-value form, multiplication $C(x) = A(x)B(x)$ is given by $C(x_k) = A(x_k)B(x_k)$ for any point x_k.
- **Problem:** if A and B are of degree-bound n, then C is of degree-bound $2n$.
- Need to start with "extended" point-value forms for A and B consisting of $2n$ point-value pairs each.
  - A: $\{(x_0, y_0), (x_1, y_1), \dots, (x_{2n-1}, y_{2n-1})\}$
  - B: $\{(x_0, y'_0), (x_1, y'_1), \dots, (x_{2n-1}, y'_{2n-1})\}$
  - C: $\{(x_0, y_0 y'_0), (x_1, y_1 y'_1), \dots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\}$
- The time to multiply two polynomials of degree-bound n in point-value form is $\Theta(n)$.

Example

- We multiply $C(x) = A(x)B(x)$ in point-value form
  - $A(x) = x^3 - 2x + 1$
  - $B(x) = x^3 + x^2 + 1$
  - $x_k = (-3, -2, -1, 0, 1, 2, 3)$ *We need 7 coefficients!*
  - A: $\{(-3,-17), (-2,-3), (-1,1), (0,1), (1,0), (2,\ 5), (3,22)\}$
  - B: $\{(-3,-20), (-2,-3), (-1,2), (0,1), (1,3), (2,13), (3,37)\}$
  - C: $\{(-3,340), (-2,9), (-1,2), (0,1), (1,0), (2,65), (3,814)\}$

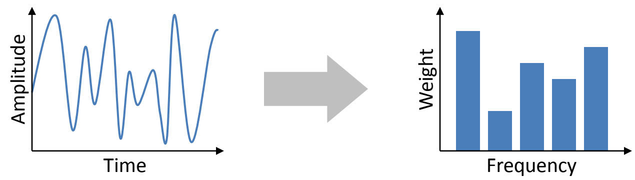
The Road So Far



- Can we do better?
  - Using Fast Fourier Transform (FFT) and its inverse, we can do evaluation and interpolation in time $\Theta(n \log n)$.
- The product of two polynomials of degree-bound n can be computed in time $\Theta(n \log n)$, with both the input and output in coefficient form.

Fourier Transform

- Fourier Transforms originate from **signal processing**
  - Transform signal from **time domain** to **frequency domain**



  - Input signal is a function mapping time to amplitude
  - Output is a weighted sum of phase-shifted sinusoids of varying frequencies

Fast Multiplication of Polynomials

- Using complex roots of unity
  - Evaluation by taking the Discrete Fourier Transform (DFT) of a coefficient vector
  - Interpolation by taking the “inverse DFT” of point-value pairs, yielding a coefficient vector
  - Fast Fourier Transform (FFT) can perform DFT and inverse DFT in time $\Theta(n \log n)$
- Algorithm
  1. Add n higher-order zero coefficients to $A(x)$ and $B(x)$
  2. Evaluate $A(x)$ and $B(x)$ using FFT for $2n$ points
  3. Pointwise multiplication of point-value forms
  4. Interpolate $C(x)$ using FFT to compute inverse DFT

Complex Roots of Unity

- A **complex $\boldsymbol{n}^{\text{th}}$ root of unity** (1) is a complex number ω such that $\omega^n = 1$.
- There are exactly n complex n^{th} root of unity

$$e^{2\pi ik/n} \text{ for } k = 0, 1, \ldots, n-1$$

  where $e^{iu} = \cos(u) + i\sin(u)$. Here u represents an angle in **radians**.
- Using $e^{2\pi ik/n} = \cos(2\pi k/n) + i\sin(2\pi k/n)$, we can check that it is a root

$$\left(e^{2\pi ik/n}\right)^n = e^{2\pi ik} = \underbrace{\cos(2\pi k)}_{1} + i\ \underbrace{\sin(2\pi k)}_{0} = 1$$

Examples

- The complex 4^{th} roots of unity are

$$1, -1, i, -i$$

  where $\sqrt{-1} = i$.
- The complex 8^{th} roots of unity are all of the above, plus four more

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

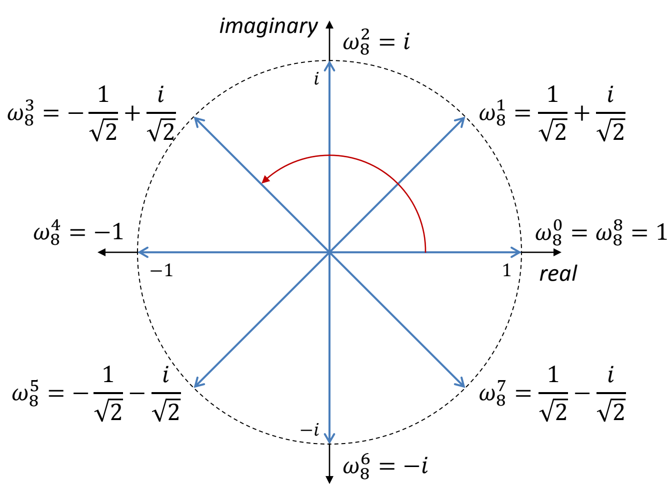
- For example

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

Principal n^{th} Root of Unity

- The value
$$\omega_n = e^{2\pi i/n}$$
is called the **principal n^{th} root of unity**.
- All of the other complex n^{th} roots of unity are powers of ω_n.
- The n complex n^{th} roots of unity, $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$, form a group under multiplication that has the same structure as $(\mathbb{Z}_n, +)$ modulo n.
- $\omega_n^n = \omega_n^0 = 1$ implies
  - $\omega_n^j \omega_n^k = \omega_n^{j+k} = \omega_n^{(j+k) \bmod n}$
  - $\omega_n^{-1} = \omega_n^{n-1}$

Visualizing 8 Complex 8^{th} Roots of Unity



Cancellation Lemma

- For any integers $n \geq 0$, $k \geq 0$, and $b > 0$,
$$\omega_{dn}^{dk} = \omega_n^k.$$
- Proof
$$\omega_{dn}^{dk} = \left(e^{2\pi i/dn}\right)^{dk} = \left(e^{2\pi i/n}\right)^{k} = \omega_n^k$$
- For any even integer $n > 0$, $\omega_n^{n/2} = \omega_2 = -1$.
- Example $\omega_{24}^6 = \omega_4$
  - $\omega_{24}^6 = \left(e^{2\pi i/24}\right)^6 = e^{2\pi i \frac{6}{24}} = e^{2\pi i/4} = \omega_4$

Halving Lemma

- If $n > 0$ is even, then the squares of the n complex n^{th} roots of unity are the $n/2$ complex $n/2^{\text{th}}$ roots of unity.
- Proof
  - By the cancellation lemma, we have $\left(\omega_n^k\right)^2 = \omega_{n/2}^k$ for any nonnegative integer k.
- If we square all of the complex n^{th} roots of unity, then each $n/2^{\text{th}}$ root of unity is obtained exactly twice
  - $\left(\omega_n^{k+n/2}\right)^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = \left(\omega_n^k\right)^2$
  - Thus, ω_n^k and $\omega_n^{k+n/2}$ have the **same square**

Summation Lemma

- For any integer $n \geq 1$ and nonzero integer k not divisible by n, $\sum_{j=0}^{n-1} \left(\omega_n^k\right)^j = 0$.
- Proof
  - Geometric series $\sum_{j=0}^{n-1} x^j = \frac{x^n - 1}{x - 1}$
  - $\sum_{j=0}^{n-1} \left(\omega_n^k\right)^j = \frac{\left(\omega_n^k\right)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0$
  - Requiring that k not be divisible by n ensures that the denominator is not 0, since $\omega_n^k = 1$ only when k is divisible by n

Discrete Fourier Transform (DFT)

- Evaluate a polynomial $A(x)$ of degree-bound n at the n complex n^{th} roots of unity, $\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$.
  - assume that n is a power of 2
  - assume A is given in coefficient form $a = (a_0, a_1, \dots, a_{n-1})$
- We define the results y_k, for $k = 0, 1, \dots, n-1$, by

$$y_k = A\left(\omega_n^k\right) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} .$$

- The vector $y = (y_0, y_1, \dots, y_{n-1})$ is the **Discrete Fourier Transform (DFT)** of the coefficient vector $a = (a_0, a_1, \dots, a_{n-1})$, denoted as $y = \mathrm{DFT}_n(a)$.

Fast Fourier Transform (FFT)

- **Fast Fourier Transform (FFT)** takes advantage of the special properties of the complex roots of unity to compute $\mathrm{DFT}_n(\mathrm{a})$ in time $\Theta(n \log n)$.
- Divide-and-conquer strategy
  - define two new polynomials of degree-bound $n/2$, using even-index and odd-index coefficients of $A(x)$ separately
  - $A^{[0]}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$
  - $A^{[1]}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$
  - $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

Fast Fourier Transform (FFT)

- The problem of evaluating $A(x)$ at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ reduces to
  1. evaluating the degree-bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$
  2. combining the results by $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$
- Why bother?
  - The list $(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2$ does not contain n distinct values, but $n/2$ complex $n/2^{\text{th}}$ roots of unity
  - Polynomials $A^{[0]}$ and $A^{[1]}$ are recursively evaluated at the $n/2$ complex $n/2^{\text{th}}$ roots of unity
  - Subproblems have exactly the same form as the original problem, but are half the size

Example

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ of degree-bound 4
  - $A(\omega_4^0) = A(1) = a_0 + a_1 + a_2 + a_3$
  - $A(\omega_4^1) = A(i) = a_o + \ a_1 i - a_2 - a_3 i$
  - $A(\omega_4^2) = A(-1) = a_0 - a_1 + a_2 - a_3$
  - $A(\omega_4^3) = A(-i) = a_0 - a_1 i + a_2 + a_3 i$
- Using $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$
  - $A(x) = a_0 + a_2 x^2 + x(a_1 + a_3 x^2)$
  - $A(\omega_4^0) = A(1) = a_0 + a_2 + 1(a_1 + a_3)$
  - $A(\omega_4^1) = A(i) = a_0 - a_2 + i(a_1 - a_3)$
  - $A(\omega_4^2) = A(-1) = a_0 + a_2 - 1(a_1 + a_3)$
  - $A(\omega_4^3) = A(-i) = a_0 - a_2 - i(a_1 - a_3)$

Recursive FFT Algorithm

RECURSIVE-FFT(a)

- $n \leftarrow length[a]$ — n is a power of 2
- **if** $n = 1$
  - **then return** a — basis of recursion
- $\omega_n \leftarrow e^{2\pi i/n}$ — ω_n is principal n^{th} root of unity
- $\omega \leftarrow 1$
- $a^{[0]} \leftarrow (a_0, a_2, \dots, a_{n-2})$
- $a^{[1]} \leftarrow (a_1, a_3, \dots, a_{n-1})$
- $y^{[0]} \leftarrow$ RECURSIVE-FFT$(a^{[0]})$ — $y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) = A^{[0]}(\omega_n^{2k})$
- $y^{[1]} \leftarrow$ RECURSIVE-FFT$(a^{[1]})$ — $y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) = A^{[1]}(\omega_n^{2k})$
- **for** $k \leftarrow 0$ **to** $n/2 - 1$
  - **do** $y_k \leftarrow y_k^{[0]} + \omega y_k^{[1]}$
  - $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega y_k^{[1]}$ — since $-\omega_n^k = \omega_n^{k+(n/2)}$
  - $\omega \leftarrow \omega \omega_n$ — compute ω_n^k iteratively
- **return** y

Why Does It Work?

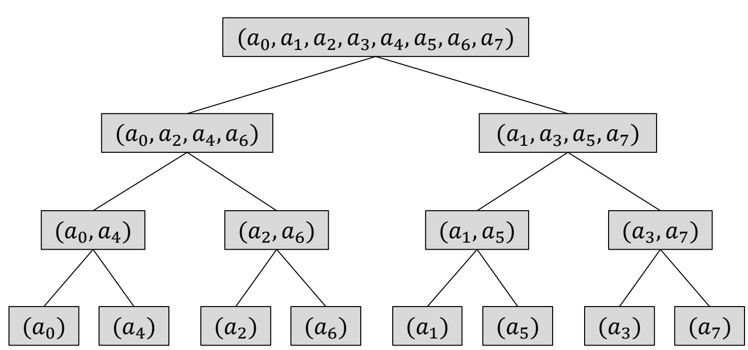
- For $y_0, y_1, \dots y_{n/2-1}$ (line 11)

$$
\begin{aligned}
y_k &= y_k^{[0]} + \omega_n^k y_k^{[1]} \\
&= A^{[0]}\left(\omega_n^{2k}\right) + \omega_n^k A^{[1]}\left(\omega_n^{2k}\right) \\
&= A\left(\omega_n^k\right)
\end{aligned}
$$

- For $y_{n/2}, y_{n/2+1}, \dots, y_{n-1}$ (line 12)

$$
\begin{aligned}
y_{k+n/2} &= y_k^{[0]} - \omega_n^k y_k^{[1]} \\
&= y_k^{[0]} + \omega_n^{k+(n/2)} y_k^{[1]} && \text{since } -\omega_n^k = \omega_n^{k+(n/2)} \\
&= A^{[0]}\left(\omega_n^{2k}\right) + \omega_n^{k+(n/2)} A^{[1]}\left(\omega_n^{2k}\right) \\
&= A^{[0]}\left(\omega_n^{2k+n}\right) + \omega_n^{k+(n/2)} A^{[1]}\left(\omega_n^{2k+n}\right) \\
& && \text{since } \omega_n^{2k+n} = \omega_n^{2k} \\
&= A\left(\omega_n^{k+(n/2)}\right)
\end{aligned}
$$

Input Vector Tree of RECURSIVEFFT(a)



Interpolation

- Interpolation by computing the inverse DFT, denoted by $\mathrm{a} = \mathrm{DFT}_n^{-1}(y)$.
- By modifying the FFT algorithm, we can compute DFT_n^{-1} in time $\Theta(n \log n)$.
  - switch the roles of a and y
  - replace ω_n by ω_n^{-1}
  - divide each element of the result by n