Fast Fourier Transform (FFT)

CSEN 1038

German University in Cairo

April 9, 2025

Outline

Polynomials

- A polynomial is a mathematical expression of the form:

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

- It can also be written in summation form:

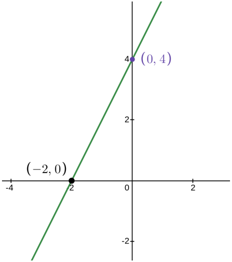
$$P(x) = \sum_{k=0}^{n-1} a_k x^k$$

- The polynomial can be represented in **coefficient vector** form:

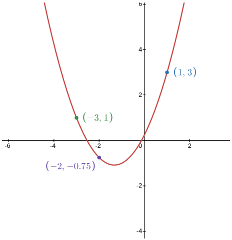
$$\mathrm{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Polynomial Degree

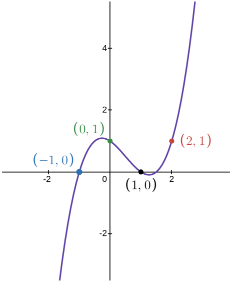
- The **degree** of a polynomial is the highest exponent of x with a nonzero coefficient.
- A polynomial of degree $n-1$ is uniquely determined by n points.



$P(x) = 4 + 2x$



$P(x) = \frac{1}{4} + 2x + \frac{3}{4}x^2$



$P(x) = 1 - \frac{2}{3}x - x^2 + \frac{2}{3}x^3$

Polynomial Operations

- When working with polynomials, we are primarily interested in:
  - **Addition:** $(P_1 + P_2)(x)$
  - **Multiplication:** $(P_1 \cdot P_2)(x)$
  - **Evaluation:** Computing $P(x)$ at specific values
  - **Interpolation:** Reconstructing $P(x)$ from given points (not covered in this lecture)

Polynomial Addition

Polynomial Addition

- Given two polynomials:

$$P_1(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$P_2(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{m-1}x^{m-1}$$

- Their sum is obtained by adding corresponding coefficients:

$$R(x) = P_1(x) + P_2(x) = \sum_k (a_k + b_k)x^k$$

- Example:

$$P_1(x) = 1 + 2x + x^2, \quad P_2(x) = 2 + 3x$$

$$R(x) = (1 + 2x + x^2) + (2 + 3x) = 3 + 5x + x^2$$

Polynomial Multiplication

Polynomial Multiplication

- Consider the same two polynomials:

$$P_1(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$P_2(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{m-1}x^{m-1}$$

- Their product is computed by multiplying each term of $P_1(x)$ with each term of $P_2(x)$:

$$R(x) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} a_i b_j x^{i+j}$$

Polynomial Multiplication Example

- Example polynomials:

$$P_1(x) = 1 + 2x + x^2, \quad P_2(x) = 1 - 2x + x^2$$

- Multiply each term:

$$(1 + 2x + x^2) \cdot (1 - 2x + x^2)$$

- Expanding:

$$1(1 - 2x + x^2) + 2x(1 - 2x + x^2) + x^2(1 - 2x + x^2)$$
$$= 1 - 2x + x^2 + 2x - 4x^2 + 2x^3 + x^2 - 2x^3 + x^4$$

- Simplify:

$$R(x) = 1 - x^2 + x^4$$

Polynomial Evaluation

Polynomial Evaluation

- Evaluating a polynomial $P(x)$ means computing its value at a given x.
- Given:

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

- Example: Let $P(x) = 2x^2 + 3x + 1$, evaluate at $x = 2$:

$$P(2) = 2(2^2) + 3(2) + 1 = 8 + 6 + 1 = 15$$

- Evaluation is fundamental for polynomial operations like interpolation and multiplication.

Multiplication via Point Evaluations

From Coefficients to Evaluations

- To multiply polynomials more efficiently, we represent them by their values at distinct input points instead of their coefficients.
- For a degree-$n-1$ polynomial $P(x)$, it's enough to know its values at n distinct points:

$$\{(x_0, P(x_0)), (x_1, P(x_1)), \ldots, (x_{n-1}, P(x_{n-1}))\}$$

- **Example:** For $P(x) = 2x + 1$, sampling at $x_0 = 0$, $x_1 = 1$ gives:

$$\{(0, 1), (1, 3)\}$$

Pointwise Polynomial Multiplication

- When multiplying $P_1(x)$ of degree $n-1$ with $P_2(x)$ of degree $m-1$, the result $R(x) = P_1(x) \cdot P_2(x)$ has degree $n+m-2$.
- To fully determine $R(x)$, we need its values at $n+m-1$ distinct points.
- **Pointwise multiplication process:**
  - **1.** Evaluate P_1 and P_2 at $n+m-1$ shared input points.
  - **2.** Multiply the corresponding outputs to get values of $R(x)$.
  - **3.** Reconstruct $R(x)$ from these values (via interpolation).

Example: Pointwise Polynomial Multiplication

- Let:

$$P_1(x) = 1 + 2x + x^2, \quad P_2(x) = 1 - 2x + x^2$$

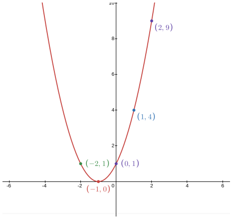
- Their product is:

$$R(x) = P_1(x) \cdot P_2(x) = 1 - 2x^2 + x^4$$

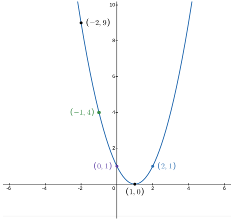
- Since $\deg(R) = 4$, we need evaluations at 5 distinct points.

	$x = -2$	$x = -1$	$x = 0$	$x = 1$	$x = 2$
$P_1(x)$	1	0	1	4	9
$P_2(x)$	9	4	1	0	1
$R(x)$	9	0	1	0	9

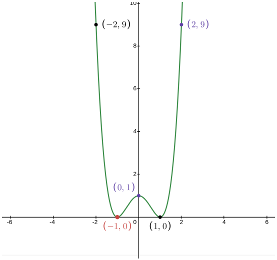
Polynomial Multiplication - Visualization



$P_1(x) = 1 + 2x + x^2$



$P_2(x) = 1 - 2x + x^2$



$R(x) = 1 - 2x^2 + x^4$

Time Complexity

Coefficient Representation

$$P_1(x) = 1 + 2x + x^2$$
$$P_2(x) = 1 - 2x + x^2$$

$$\Big\downarrow \; O(n^2)$$

$$R(x) = P_1(x) \cdot P_2(x)$$
$$R(x) = 1 - 2x^2 + x^4$$

Pointwise Representation

$$(x_0, P_1(x_0))\ldots$$
$$(x_0, P_2(x_0))\ldots$$

$$\Big\downarrow \; O(n)$$

$$(x_0, P_1(x_0).P_2(x_0))\ldots$$
$$(x_0, R(x_0))\ldots$$

Time Complexity

Coefficient Representation | Pointwise Representation

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\begin{array}{c} P_1(x) = 1 + 2x + x^2 \\ P_2(x) = 1 - 2x + x^2 \end{array}} \arrow[r, dashed, "O(n^2)"] \arrow[d, "O(n^2)"'] & \boxed{\begin{array}{c} (x_0, P_1(x_0))\ldots \\ (x_0, P_2(x_0))\ldots \end{array}} \arrow[d, "O(n)"] \\
\boxed{\begin{array}{c} R(x) = P_1(x) \cdot P_2(x) \\ R(x) = 1 - 2x^2 + x^4 \end{array}} & \boxed{\begin{array}{c} (x_0, P_1(x_0).P_2(x_0))\ldots \\ (x_0, R(x_0))\ldots \end{array}} \arrow[l, dashed, "O(n^2)"]
\end{tikzcd}
$$

Time Complexity

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\begin{array}{rl} P_1(x) &= 1 + 2x + x^2 \\ P_2(x) &= 1 - 2x + x^2 \end{array}} \arrow[r, dashed, "\text{FFT: } O(n\log n)"] \arrow[d, "O(n^2)"'] & \boxed{\begin{array}{c} (x_0, P_1(x_0))\ldots \\ (x_0, P_2(x_0))\ldots \end{array}} \arrow[d, "O(n)"] \\
\boxed{\begin{array}{rl} R(x) &= P_1(x) \cdot P_2(x) \\ R(x) &= 1 - 2x^2 + x^4 \end{array}} & \boxed{\begin{array}{c} (x_0, P_1(x_0)P_2(x_0))\ldots \\ (x_0, R(x_0))\ldots \end{array}} \arrow[l, dashed, "\text{IFFT: } O(n\log n)"]
\end{tikzcd}$$

Fast Polynomial Multiplication

FFT Polynomial Multiplication

- **Problem:** Given two polynomials $P_1(x)$ and $P_2(x)$, we want to multiply them efficiently.

- **Preprocessing:** First, we pad both polynomials with zeros so their length becomes a power of 2:

$$n \geq \deg(P_1) + \deg(P_2) + 1$$

- Next, we apply the Fast Fourier Transform (FFT) to evaluate each polynomial at n points, then we perform pointwise multiplication.

FFT: Fast Polynomial Evaluation

Problem: Given a polynomial $A(x)$ of degree $n-1$, where $n = 2^m$, evaluate it at n distinct points $X = \{x_0, x_1, \ldots, x_{n-1}\}$.

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

FFT: A Divide and Conquer Algorithm

The Fast Fourier Transform (FFT) is a classic divide and conquer algorithm with three main steps:

- **Divide:** Split the polynomial into 2 parts.
- **Conquer:** Recursively evaluate the smaller sub-polynomials.
- **Combine:** Merge the results from these smaller parts.

FFT: Divide Step

Step 1: Split the Polynomial

For a polynomial $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ where $n = 2^m$, divide it into:

- **Even-indexed terms:**

$$A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{n-2}x^{n/2-1}$$

- **Odd-indexed terms:**

$$A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{n/2-1}$$

FFT: Divide Step

Step 1: Split the Polynomial

For a polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$ where $n = 2^m$, divide it into:

- **Even-indexed terms:**

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$$

- **Odd-indexed terms:**

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$$

Step 2: Recursive Structure

$$A(x) = A_{\text{even}}(x^2) + x \cdot A_{\text{odd}}(x^2)$$

Conquer and Combine Steps

Conquer Step: Recursive Evaluation

- Obtain the squared set $X^2 = \{x_k^2 \mid x_k \in X\}$.
- Recursively evaluate the polynomials on this new set:
  - Compute $A_{\text{even}}(x_k^2)$ for each $x_k^2 \in X^2$.
  - Compute $A_{\text{odd}}(x_k^2)$ for each $x_k^2 \in X^2$.

Conquer and Combine Steps

Conquer Step: Recursive Evaluation

- Obtain the squared set $X^2 = \{x_k^2 \mid x_k \in X\}$.
- Recursively evaluate the polynomials on this new set:
  - Compute $A_{\text{even}}(x_k^2)$ for each $x_k^2 \in X^2$.
  - Compute $A_{\text{odd}}(x_k^2)$ for each $x_k^2 \in X^2$.

Combine Step: Merging Results

- Using results from the recursive step, compute the final evaluations:

$$A(x_k) = A_{\text{even}}(x_k^2) + x_k \cdot A_{\text{odd}}(x_k^2), \quad 0 \leq k < n.$$

Conquer and Combine Steps

Conquer Step: Recursive Evaluation

- Obtain the squared set $X^2 = \{x_k^2 \mid x_k \in X\}$.
- Recursively evaluate the polynomials on this new set:
  - Compute $A_{\text{even}}(x_k^2)$ for each $x_k^2 \in X^2$.
  - Compute $A_{\text{odd}}(x_k^2)$ for each $x_k^2 \in X^2$.

Combine Step: Merging Results

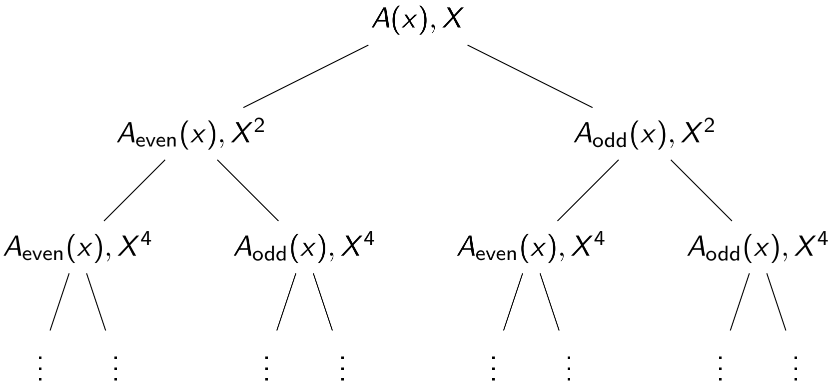
- Using results from the recursive step, compute the final evaluations:

$$A(x_k) = A_{\text{even}}(x_k^2) + x_k \cdot A_{\text{odd}}(x_k^2), \quad 0 \leq k < n.$$

Time Complexity: $O(n^2)$

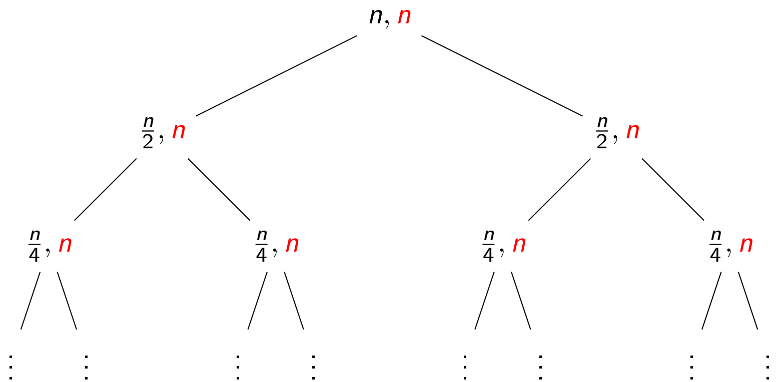
FFT Recursion Tree

Recursion Tree for FFT:



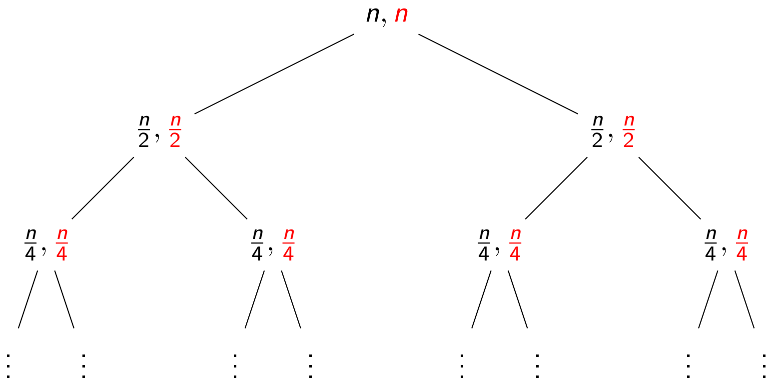
FFT Recursion Tree

Recursion Tree for FFT:



FFT Recursion Tree

Is this possible?:



$$S_0 = \{1\}$$

$$S_0 = \{1\}$$

$$S_1 = \{1, -1\}$$

$$S_0 = \{1\}$$

$$S_1 = \{1, -1\}$$

$$S_2 = \{1, i, -1, -i\}$$

$$S_0 = \{1\}$$

$$S_1 = \{1, -1\}$$

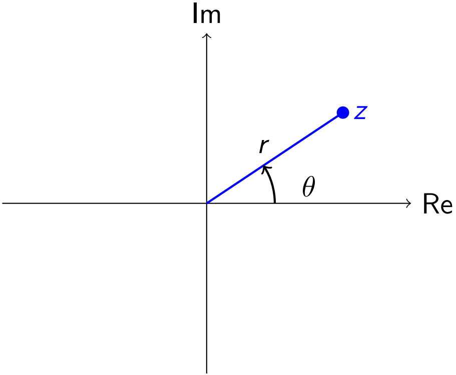
$$S_2 = \{1, i, -1, -i\}$$

$$S_3 = \left\{1, e^{i\pi/4}, i, e^{3i\pi/4}, -1, e^{5i\pi/4}, -i, e^{7i\pi/4}\right\}$$

Complex Numbers

A complex number $z = a + b\mathrm{i}$ is a point in the 2D plane:

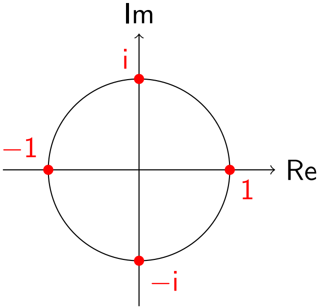
$$z = re^{\mathrm{i}\theta} = r(\cos\theta + \mathrm{i}\sin\theta)$$



The Unit Circle

Definition: The unit circle consists of all complex numbers z with magnitude $|z| = 1$:

$$z = e^{\mathrm{i}\theta} = \cos\theta + \mathrm{i}\sin\theta$$



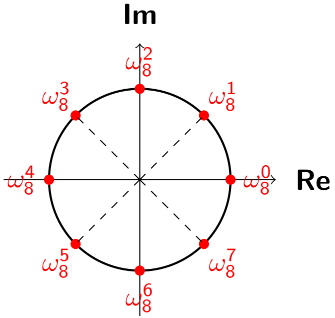
Roots of Unity

Definition: The nth roots of unity satisfy $z^n = 1$:

$$z = e^{\mathrm{i}\frac{2\pi k}{n}}, \quad k = 0, 1, \ldots, n-1$$

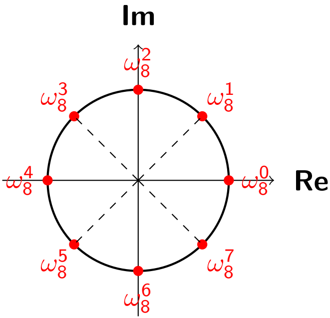
Example:

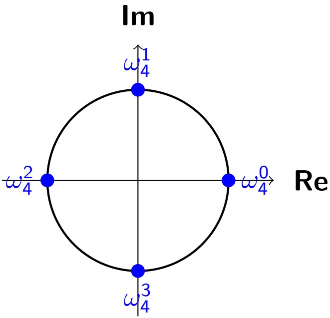
$$1, \omega_8, \omega_8^2, \ldots, \omega_8^7, \quad \text{where} \quad \omega_8 = e^{\mathrm{i}\frac{2\pi}{8}}$$



Squaring Roots of Unity

Key Observation: Squaring the 8th roots of unity maps them to the 4th roots of unity.





FFT Example: $n = 8$

Given Polynomial

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_7 x^7$$

We evaluate at roots of unity:

$$x = \omega_8^k, \quad k = 0, 1, \ldots, 7.$$

FFT Example: Splitting the Polynomial

Step 1: Divide into Even/Odd Parts

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$$

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$$

FFT Example: Splitting the Polynomial

Step 1: Divide into Even/Odd Parts

$$A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$$

$$A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$$

Since $\omega_8^2 = \omega_4$, we evaluate:

$$A_{\text{even}}(x) \text{ and } A_{\text{odd}}(x) \text{ at } \omega_4^k, \quad k = 0, 1, 2, 3.$$

FFT Example: Recursive Computation

Step 2: Recursively Compute FFT

- Compute FFT on the reduced polynomials.

Step 3: Combine the Results

For final evaluation at $\omega_8^k, \quad k = 0, 1, 2, 3$:

$$A(\omega_8^k) = A_{\text{even}}(\omega_4^k) + \omega_8^k A_{\text{odd}}(\omega_4^k)$$

FFT Example: Recursive Computation

Step 2: Recursively Compute FFT

- Compute FFT on the reduced polynomials.

Step 3: Combine the Results

For final evaluation at $\omega_8^k, \quad k = 0, 1, 2, 3$:

$$A(\omega_8^k) = A_{\text{even}}(\omega_4^k) + \omega_8^k A_{\text{odd}}(\omega_4^k)$$

$$A(\omega_8^{k+4}) = A_{\text{even}}(\omega_4^k) - \omega_8^k A_{\text{odd}}(\omega_4^k)$$

FFT Algorithm

Input: Polynomial a of length $n = 2^k$ (power of two)
Output: In-place computation of the DFT of a
$n \leftarrow \text{length}(a)$
if $n == 1$ **then**
    return ▷ Base case: no computation needed
end if
$a_{\text{even}} \leftarrow (a[0], a[2], \ldots, a[n-2])$ ▷ Even indexed elements
$a_{\text{odd}} \leftarrow (a[1], a[3], \ldots, a[n-1])$ ▷ Odd indexed elements
$\text{FFT}(a_{\text{even}})$ ▷ Recursively compute FFT of even elements
$\text{FFT}(a_{\text{odd}})$ ▷ Recursively compute FFT of odd elements
$\omega \leftarrow 1$, $\omega_n \leftarrow e^{2\pi i/n}$
for $k = 0$ to $n/2 - 1$ **do**
    $a[k] \leftarrow a_{\text{even}}[k] + \omega \cdot a_{\text{odd}}[k]$
    $a[k + n/2] \leftarrow a_{\text{even}}[k] - \omega \cdot a_{\text{odd}}[k]$
    $\omega \leftarrow \omega \cdot \omega_n$
end for

Almost There!

$$\begin{tikzcd}
\boxed{\begin{array}{l} P_1(x) = 1 + 2x + x^2 \\ P_2(x) = 1 - 2x + x^2 \end{array}} \arrow[r, dashed, "\text{FFT: } O(n \log n)"] \arrow[d, "O(n^2)"'] & \boxed{\begin{array}{c} (x_0, P_1(x_0))\ldots \\ (x_0, P_2(x_0))\ldots \end{array}} \arrow[d, "O(n)"] \\
\boxed{\begin{array}{l} R(x) = P_1(x) \cdot P_2(x) \\ R(x) = 1 - 2x^2 + x^4 \end{array}} & \boxed{\begin{array}{c} (x_0, P_1(x_0)P_2(x_0))\ldots \\ (x_0, R(x_0))\ldots \end{array}} \arrow[l, dashed, red, "\text{IFFT: } O(n \log n)"]
\end{tikzcd}$$

FFT: Polynomial Evaluation

So far, we've used the FFT to evaluate a polynomial $A(x)$ at nth roots of unity.

For each $k = 0, 1, \ldots, n-1$, we compute:

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

FFT computes all $A(\omega^k)$ values efficiently in $O(n \log n)$ time.

FFT: Polynomial Evaluation

Which can be written in the matrix form:

$$\mathrm{W}_n \mathrm{a} = \mathrm{A}$$

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}}_{\mathrm{W}_n} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{\mathrm{a}} = \underbrace{\begin{pmatrix} A(\omega^0) \\ A(\omega^1) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix}}_{\mathrm{A}}$$

IFFT: Recovering the Coefficients

In the IFFT, we start with the evaluations and need to recover the coefficients:

$$\mathsf{W}_n\mathsf{a} = \mathsf{A}$$

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}}_{\mathsf{W}_n} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{\mathsf{a}} = \underbrace{\begin{pmatrix} A(\omega^0) \\ A(\omega^1) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix}}_{\mathsf{A}}$$

IFFT: Computing a

To recover a, we use the inverse:

$$\mathrm{a} = \mathrm{W}_n^{-1}\mathrm{A}$$

$$\underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{\mathrm{a}} = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}^{-1}}_{\mathrm{W}_n^{-1}} \underbrace{\begin{pmatrix} A(\omega^0) \\ A(\omega^1) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix}}_{\mathrm{A}}$$

Inverse Property of W_n

The inverse of W_n follows a simple rule:

$$W_n^{-1} = \frac{1}{n}\overline{W_n}$$

$$W_n^{-1} \quad = \quad \frac{1}{n}\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \cdots & \omega^{-(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{pmatrix}$$

Key Observation:

- The IFFT uses the same FFT algorithm but with ω replaced by ω^{-1}.
- In the end a scaling factor of $\frac{1}{n}$ is applied.

FFT and IFFT Algorithms

It's time to write the code!

FFT Applications

Thank You

Thank You!