

Fast Fourier Transform

Polynomial Multiplications and Fast Fourier Transform

Polynomial Multiplication

- Problem: Given **two polynomials** $p(x)$ and $q(x)$ with degree $d - 1$, compute its product $r(x) = p(x)q(x)$.
- Each polynomial is encoded by its coefficients:
 - $p(x) = \sum_{i=0}^{d-1} a_i x^i \rightarrow (a_0, a_1, \dots, a_{d-1})$
 - $q(x) = \sum_{i=0}^{d-1} b_i x^i \rightarrow (b_0, b_1, \dots, b_{d-1})$
- Need to compute
$$r(x) = \sum_{i=0}^{2d-2} c_i x^i \quad \text{where} \quad c_i = \sum_{k=0}^i a_k b_{i-k}$$
- Naïve computation: $O(d^2)$

Polynomial Multiplication

- Given $p(x) = \sum_{i=0}^{d-1} a_i x^i$ and $q(x) = \sum_{i=0}^{d-1} b_i x^i$
- Compute $r(x) = \sum_{i=0}^{2d-2} c_i x^i$ where $c_i = \sum_{k=0}^i a_k b_{i-k}$
- Can we do better than $O(d^2)$?

Divide and Conquer

- Adapt Karatsuba Algorithm
- Assume d is an integer power of 2.
- Write $p(x) = p_1(x) + p_2(x) \cdot x^{\frac{d}{2}}$ where
$$p_1(x) = a_0 + a_1x + \cdots + a_{\frac{d}{2}-1}x^{\frac{d}{2}-1} \quad \text{and} \quad p_2(x) = a_{\frac{d}{2}} + a_{\frac{d}{2}+1}x + \cdots + a_{d-1}x^{\frac{d}{2}-1}$$
- Similarly, write $q(x) = q_1(x) + q_2(x) \cdot x^{\frac{d}{2}}$
- Then, $r = p_1q_1 + (p_1q_2 + p_2q_1)x^{\frac{d}{2}} + p_2q_2x^d$. We need to compute

Adapting Karatsuba Algorithm

- Need to compute p_1q_1 , p_2q_2 , and $p_1q_2 + p_2q_1$
- $(p_1q_2 + p_2q_1) = (p_1 + p_2)(q_1 + q_2) - p_1q_1 - p_2q_2$
- Compute
 - p_1q_1
 - p_2q_2
 - $(p_1 + p_2)(q_1 + q_2)$
- One size- d multiplication \rightarrow Three size- $\frac{d}{2}$ multiplications
- Time Complexity

$$T(d) = 3T\left(\frac{d}{2}\right) + O(d) \Rightarrow T(d) = O(d^{\log_2 3})$$

Polynomial Multiplications vs Integer Multiplications

- $23341 = 2 \times 10^4 + 3 \times 10^3 + 3 \times 10^2 + 4 \times 10 + 1$
- $p(x) = 2x^4 + 3x^3 + 3x^2 + 4x + 1$
- Polynomials and integers are similar!
- Perhaps the only difference in multiplications is "carry".
- Some tricky things about computational model.
- FFT-based algorithms for integer multiplications:
 - Schonhage-Strassen (1971): $O(n \log n \log \log n)$
 - Furer (2007): $O(n \log n \log^* n)$
 - Harvey and van der Hoeven (2019): $O(n \log n)$

Fast Fourier Transform (FFT)

- In this lecture, we will learn a new divide and conquer algorithm with time complexity $O(d \log d)$!
- Fast Fourier Transform (FFT)
- Polynomial Interpolation
- Complex Numbers

Another Interpretation of A Polynomial

Polynomial Interpolation

- Represent a polynomial $p(x)$ of degree $d - 1$ by d points
 $(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{d-1}, p(x_{d-1}))$

where x_0, x_1, \dots, x_{d-1} are distinct.

Before we move on...

- Let's prove that d distinct points can indeed uniquely determine a polynomial of degree $d - 1$.

Interpolation Theorem. Given d points $(x_0, y_0), (x_1, y_1), \dots, (x_{d-1}, y_{d-1})$ such that $x_i \neq x_j$ for any $i \neq j$, there exists a unique polynomial $p(x)$ with degree at most $d - 1$ such that $p(x_i) = y_i$ for each i .

Proof of Interpolation Theorem

- Let $p(x) = \sum_{t=0}^{d-1} a_t x^t$. We have $y_i = \sum_{t=0}^{d-1} a_t x_i^t$ for each i .

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{d-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{d-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{d-1} & x_{d-1}^2 & \cdots & x_{d-1}^{d-1} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{bmatrix}$$

- We want to show: $(a_0, a_1, \dots, a_{d-1})$ satisfying the above equation is **unique**.
- The **yellow matrix** is a *Vandermonde matrix* with determinant $\prod_{0 \leq i < j \leq d-1} (x_j - x_i)$, which is nonzero given $x_i \neq x_j$.
- Uniqueness is proved: $\mathbf{y} = \mathbf{X}\mathbf{a} \Rightarrow \mathbf{a} = \mathbf{X}^{-1}\mathbf{y}$

Framework for FFT

- Interpolation Step (FFT):
 - Choose $2d - 1$ distinct numbers $x_0, x_1, \dots, x_{2d-2}$, and
 - Compute the values of
 - $p(x_0), p(x_1), \dots, p(x_{2d-2})$
 - $q(x_0), q(x_1), \dots, q(x_{2d-2})$
- Multiplication Step:
 - For each $i = 0, 1, \dots, 2d - 2$, compute $r(x_i) = p(x_i)q(x_i)$
 - Obtain interpolation for $r(x)$: $(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$
- Recovery Step (inverse FFT):
 - Recover $(c_0, c_1, \dots, c_{2d-2})$, the polynomial $r(x) = \sum_{i=0}^{2d-2} c_i x^i$, from the interpolation obtained in the previous step.

Framework for FFT

$$p(x) = a_0 + a_1x + \cdots + a_{d-1}x^{d-1}$$
$$q(x) = b_0 + b_1x + \cdots + b_{d-1}x^{d-1}$$

Objective



$$r(x) = p(x) \cdot q(x)$$
$$= c_0 + c_1x + \cdots + c_{2d-2}x^{2d-2}$$



Interpolation Step
(FFT)

$$(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{2d-2}, p(x_{2d-2}))$$
$$(x_0, q(x_0)), (x_1, q(x_1)), \dots, (x_{2d-2}, q(x_{2d-2}))$$



Multiplication
 $r(x_i) = p(x_i)q(x_i)$



Recovery Step
(Inverse FFT)

$$(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$$

Step 1: Interpolation

Interpolation Step (FFT)

Interpolation Step

- Interpolation Step (FFT):
 - Choose $2d - 1$ distinct numbers $x_0, x_1, \dots, x_{2d-2}$, and
 - Compute the values of $p(x_0), p(x_1), \dots, p(x_{2d-2}), q(x_0), q(x_1), \dots, q(x_{2d-2})$
- Computing each $p(x_i)$ or $q(x_i)$ requires $O(d)$ time.
 - assume we can do x^d fast.
- We need to compute $4d - 2$ of them.
- Overall time complexity: $O(d^2)$.
 - Even assume we can calculate a_i^d fast.
- Can we do faster by divide and conquer?

Some Notations

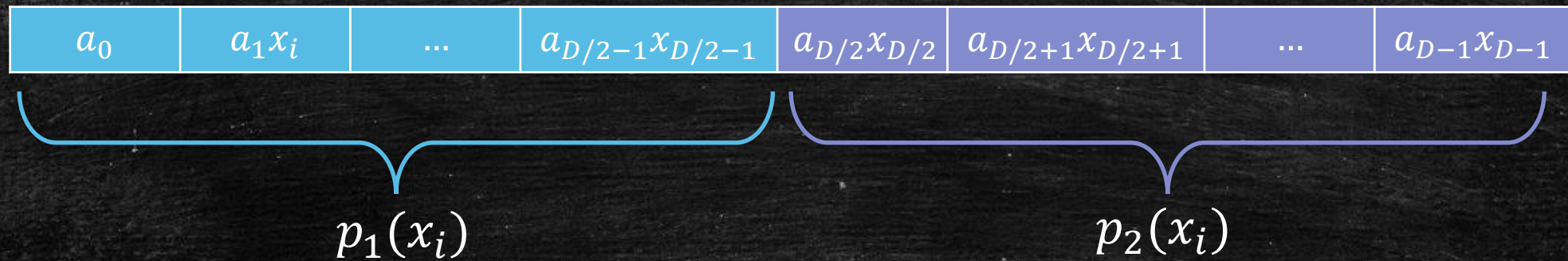
- Let $D = 2d - 1$.
- Assume D is an integer power of 2.
 - We want to divide and conquer!
- We can solve x^d in $O(1)$!
- Interpolation Step (FFT):
 - Choose $D - 1$ distinct numbers x_0, x_1, \dots, x_{D-1} , and
 - compute the values of $p(x_0), p(x_1), \dots, p(x_{D-1}), q(x_0), q(x_1), \dots, q(x_{D-1})$

Seems impossible, but it
dose not matter.

Can we calculate $p(x_i)$
faster?

Divide and Conquer: Computing $p(x_i)$

- "Left-right decomposition": $p(x_i) = p_1(x_i) + p_2(x_i) \cdot \alpha_i^{\frac{D}{2}}$



Divide and Conquer: Computing $p(x_i)$

- Compute $p_1(x_i)$ and $p_2(x_i)$ recursively.
- Time complexity: $T(D) = 2T\left(\frac{D}{2}\right) + O(1) \Rightarrow T(D) = O(D)$
- No faster than direct computation!

Divide and Conquer: Computing different x_i

- Divide among different x_i ...

$p(x_0)$						
$p(x_1)$						
...						
$p(x_{D/2-1})$						
$p(x_{D/2})$						
$p(x_{D/2+1})$						
...						
$p(x_{D-1})$						



Lessons we learned

- Computing each $p(x_i)$ requires $O(D)$ time.
 - It seems very hard to improve!
- We need to choose x_0, x_1, \dots, x_{D-1} in a clever way so that, for example, $p(x_0)$ and $p(x_1)$ can be computed **together**!
- Consider the example $p(1)$ and $p(-1)$.

An Idea to Compute $p(x_1)$ and $p(x_2)$ Together

- **Even-Odd Decomposition:**

$$p(x) = p_e(x^2) + x \cdot p_o(x^2),$$

where

$$p_e(x) = a_0 + a_2x + a_4x^2 + \cdots + a_{D-2}x^{\frac{D-2}{2}}$$

$$p_o(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{D-1}x^{\frac{D-1}{2}}$$

- Choose x_1 and x_2 such that $x_1 = -x_2$. We have
 $p_e(x_1^2) = p_e(x_2^2)$ and $p_o(x_1^2) = p_o(x_2^2)$

An Idea to Compute $p(\alpha_0)$ And $p(\alpha_1)$ Together

$$p(x_0) = p_e(x_0^2) + x_1 \cdot p_o(x_0^2)$$

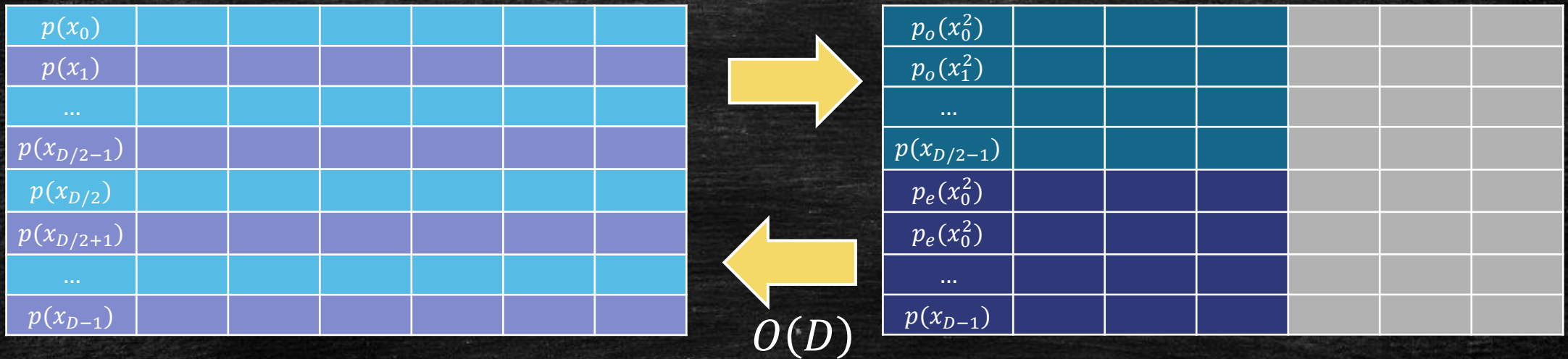
$$p(x_1) = p_e(x_1^2) + x_2 \cdot p_o(x_1^2)$$

Two size- D computations \rightarrow ~~four~~ two size- $\frac{D}{2}$ computations, **great!**

A Divide and Conquer Attempt

- Choose x_0, x_1, \dots, x_{D-1} such that $x_0 = -x_1, x_2 = -x_3, \dots, x_{D-2} = -x_{D-1}$.
- **Divide:**
 - $p_e(x_0^2), p_e(x_2^2), \dots, p_e(x_{D-2}^2)$
 - $p_o(x_0^2), p_o(x_2^2), \dots, p_o(x_{D-2}^2)$
- **Combine:** Compute $p(x_i) = p_e(x_i^2) + x_i \cdot p_o(x_i^2)$.
- **Time Complexity**
- $T(D) = 2T\left(\frac{D}{2}\right) + O(D) \Rightarrow T(D) = O(D \log D)$

What Happens?



- **Time Complexity**

- $T(D) = 2T\left(\frac{D}{2}\right) + O(D) \Rightarrow T(D) = O(D \log D)$

Are We Done?

Are We Done?

- Choose x_0, x_1, \dots, x_{D-1} such that $x_0 = -x_1, x_2 = -x_3, \dots, x_{D-2} = -x_{D-1}$.
- **Divide:**
 - $p_e(x_0^2), p_e(x_2^2), \dots, p_e(x_{D-2}^2)$
 - $p_o(x_0^2), p_o(x_2^2), \dots, p_o(x_{D-2}^2)$
- **Combine:** Compute $p(x_i) = p_e(x_i^2) + x_i \cdot p_o(x_i^2)$.
- In the second step:
 - $x_0^2, x_1^2 \dots$ are all **positive!**
 - How to make $x_0^2 = -x_1^2$?

How to do it recursively?



Why it fails?

x_0	$x_1 = -x_0$	x_2	$x_3 = -x_2$	x_4	$x_5 = -x_4$	x_6	$x_7 = -x_6$
-------	--------------	-------	--------------	-------	--------------	-------	--------------



I can do it!

x_0^2	x_2^2	x_4^2	x_6^2
---------	---------	---------	---------



How to make
 $x_0^2 = -x_2^2$.

?	?
---	---

Complex Numbers

- $z = a + bi$
 - a : real part
 - b : imaginary part
 - $i = \sqrt{-1}$: imaginary unit
- Polar form: $z = r(\cos \theta + i \sin \theta)$
 - r : the length of the 2-dimensional vector (a, b)
 - θ : the angle between the vector (a, b) and the x -axis (the real axis)
- Euler's formula: $z = r(\cos \theta + i \sin \theta) = r \cdot e^{\theta i}$

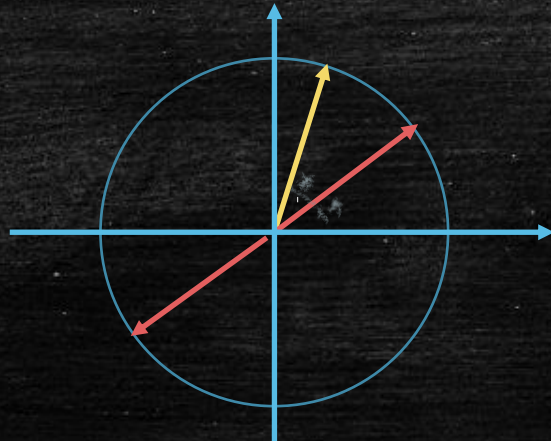


Squares and Square Roots of Unit Length Complex Numbers

- The square of $e^{\theta i}$ is $e^{2\theta i}$: we have just rotated $e^{\theta i}$ by an angle θ .
- Two complex numbers of unit length opposite to each other have the same square:

$$(e^{(\theta+\pi)i})^2 = e^{2\theta i} \cdot e^{2\pi i} = e^{2\theta i} = (e^{\theta i})^2$$

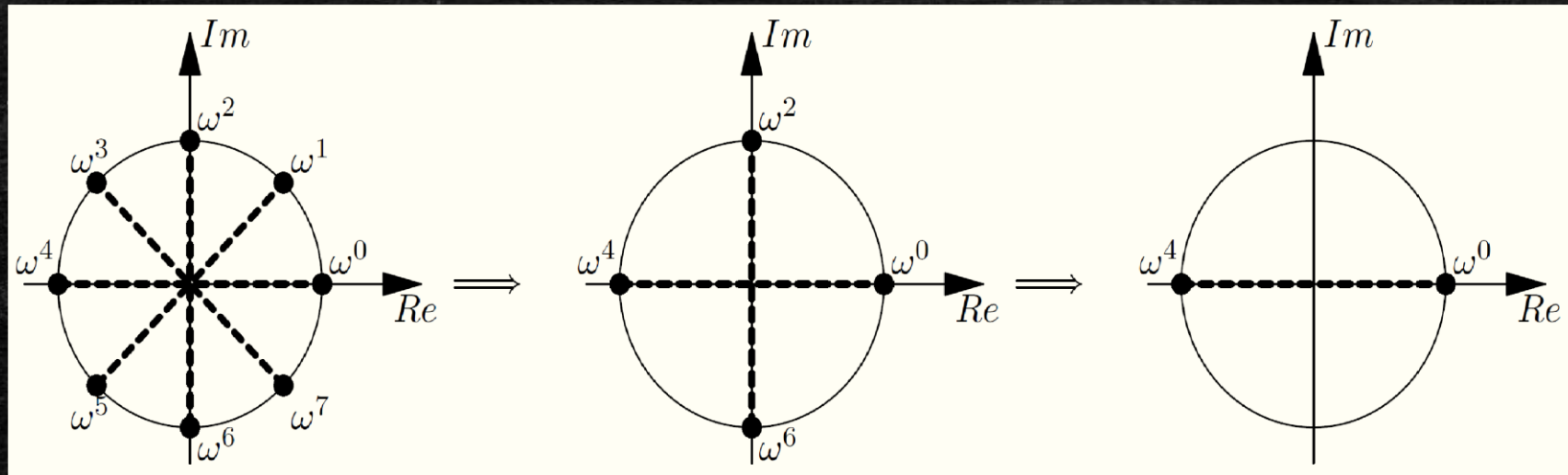
- The square roots of $e^{\theta i}$ are $e^{\frac{\theta}{2}i}$ and $e^{(\frac{\theta}{2}+\pi)i}$



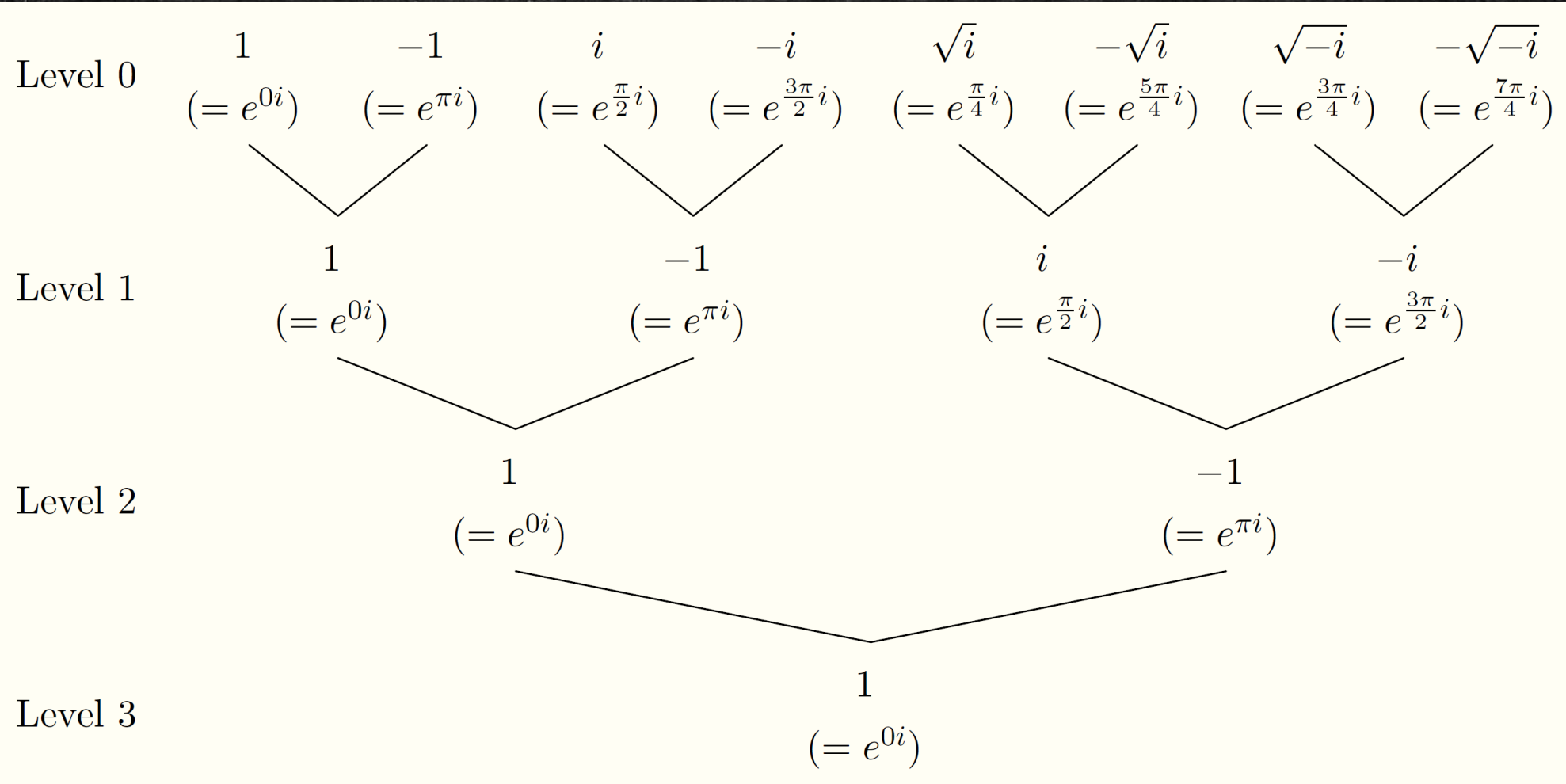
Example for $D = 8$

$$\omega_0 = 1, \quad \omega_1 = e^{\frac{\pi}{4}i}, \quad \omega_2 = e^{\frac{\pi}{2}i}, \quad \omega_3 = e^{\frac{3\pi}{4}i}$$

$$\omega_4 = e^{\pi i}, \quad \omega_5 = e^{\frac{5\pi}{4}i}, \quad \omega_6 = e^{\frac{3\pi}{2}i}, \quad \omega_7 = e^{\frac{7\pi}{4}i}$$

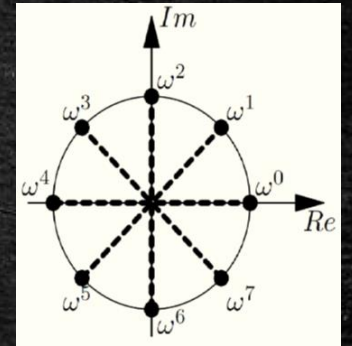


Example for $D = 8$



How to represent the D points

- Fix D , we know the D numbers are $\omega^0, \omega^1, \omega^2 \dots, \omega^{D-1}$.
 - $\omega = e^{\frac{2\pi i}{D}}$
- We only need one parameter ω to represent the D numbers!
- What about the next level numbers?
- They are $\omega^0, \omega^2, \omega^4 \dots, \omega^{D-2}$.
- We can use ω^2 to represent the next level numbers!



Interpolation: Putting Together

Algorithm 1: Fast Fourier Transform

FFT(p, ω): // p is a polynomial of degree $D - 1$ and $\omega = e^{\frac{2\pi i}{D}}$

1. if $\omega = 1$, return ($p(1)$);
2. $p_e(x) = a_0 + a_2x + a_4x^2 + \dots + a_{D-2}x^{\frac{D-2}{2}}$
3. $p_o(x) = a_1 + a_3x + a_5x^2 + \dots + a_{D-1}x^{\frac{D-2}{2}}$
4. $(p_e(\omega^0), p_e(\omega^2), \dots, p_e(\omega^{D-2})) \leftarrow \text{FFT}(p_e, \omega^2);$
5. $(p_o(\omega^0), p_o(\omega^2), \dots, p_o(\omega^{D-2})) \leftarrow \text{FFT}(p_o, \omega^2);$
6. for $t = 0, 1, \dots, D - 1$:
7. $p(\omega^t) = p_e(\omega^{2t}) + \omega^t \cdot p_o(\omega^{2t})$
8. return ($p(\omega^0), p(\omega^1), \dots, p(\omega^{D-1})$);

Time Complexity for Interpolation Step

- Let $T(D)$ be the time complexity for computing **FFT** (p, ω) , where p has degree $D - 1$.
- We have $T(D) = 2T\left(\frac{D}{2}\right) + O(D) = O(D \log D)$.
- Interpolation step requires $T(D) = O(d \log d)$ time.

Framework for FFT

$$p(x) = a_0 + a_1x + \cdots + a_{d-1}x^{d-1}$$
$$q(x) = b_0 + b_1x + \cdots + b_{d-1}x^{d-1}$$

Objective



$$r(x) = p(x) \cdot q(x)$$
$$= c_0 + c_1x + \cdots + c_{2d-2}x^{2d-2}$$



Interpolation Step
(FFT)

$O(d \log d)$

$$(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{2d-2}, p(x_{2d-2}))$$
$$(x_0, q(x_0)), (x_1, q(x_1)), \dots, (x_{2d-2}, q(x_{2d-2}))$$



Multiplication
 $r(x_i) = p(x_i)q(x_i)$



Recovery Step
(Inverse FFT)

$$(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$$

Step 2: Multiplication

Multiplication Step:

For each $i = 0, 1, \dots, 2d - 2$, compute $r(x_i) = p(x_i)q(x_i)$

Obtain interpolation for $r(x)$: $(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$

It's easy! Just compute it one-by-one...

- For each $i = 0, 1, \dots, 2d - 2$, compute $r(x_i) = p(x_i)q(x_i)$
- Time complexity: $O(d)$

Framework for FFT

$$p(x) = a_0 + a_1x + \cdots + a_{d-1}x^{d-1}$$
$$q(x) = b_0 + b_1x + \cdots + b_{d-1}x^{d-1}$$

Objective



$$r(x) = p(x) \cdot q(x)$$
$$= c_0 + c_1x + \cdots + c_{2d-2}x^{2d-2}$$

Interpolation Step
(FFT) ✓
 $O(d \log d)$

$$(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{2d-2}, p(x_{2d-2}))$$
$$(x_0, q(x_0)), (x_1, q(x_1)), \dots, (x_{2d-2}, q(x_{2d-2}))$$

$O(d)$ ✓



Multiplication
 $r(x_i) = p(x_i)q(x_i)$

Recovery Step
(Inverse FFT)

$$(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$$

Step 3: Recovery

Recovery Step (inverse FFT):

Recover $(c_0, c_1, \dots, c_{2d-2})$, the polynomial $r(x) = \sum_{i=0}^{2d-2} c_i x^i$, from the interpolation obtained in the previous step.

We Have Interpolation of $r(x)$ Now...

- We have $(1, r(1)), (\omega, r(\omega)), (\omega^2, r(\omega^2)), \dots, (\omega^{D-1}, r(\omega^{D-1}))$, where $\omega = e^{\frac{2\pi i}{D}}$.

$$\begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{D-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{D-1} & \omega^{2(D-1)} & \dots & \omega^{(D-1)(D-1)} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix}$$

↑
What we want...

We Have Interpolation of $r(x)$ Now...

- We have $(1, r(1)), (\omega, r(\omega)), (\omega^2, r(\omega^2)), \dots, (\omega^{D-1}, r(\omega^{D-1}))$, where $\omega = e^{\frac{2\pi i}{D}}$.

$$A = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{D-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{D-1} & \omega^{2(D-1)} & \dots & \omega^{(D-1)(D-1)} \end{bmatrix}$$

$$\textcircled{A^{-1}} \begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix}$$

Complex Matrices Recap

- The **complex conjugate** of $z = a + bi$ is $\bar{z} = a - bi$.
- Given two complex vectors $\mathbf{a}, \mathbf{b} \in \mathbb{C}^n$, their **inner product** is

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{j=1}^n \overline{a_j} \cdot b_j$$

- \mathbf{a}, \mathbf{b} are **orthogonal** if $\langle \mathbf{a}, \mathbf{b} \rangle = 0$; \mathbf{a}, \mathbf{b} are **orthonormal** if $\langle \mathbf{a}, \mathbf{b} \rangle = 0$ and $\langle \mathbf{a}, \mathbf{a} \rangle = \langle \mathbf{b}, \mathbf{b} \rangle = 1$.
正交矩阵
- A square matrix A is an **orthonormal (unitary)** matrix if every pair of its columns is orthonormal.
- **Conjugate transpose** of A , denoted by A^* , is defined as $(A^*)_{i,j} = \overline{A_{j,i}}$.
- If A is an orthonormal, then A is invertible and $A^{-1} = A^*$.

Complex Matrices Recap

- A square matrix A is an **orthonormal** (**unitary**) matrix if every pair of its columns is orthonormal.
- **Conjugate transpose** of A , denoted by A^* , is defined as $(A^*)_{i,j} = \overline{A_{j,i}}$.
- If A is an orthonormal, then A is invertible and $A^{-1} = A^*$.

$$\text{Example: } A^* = \begin{pmatrix} \bar{a} & \bar{b} \\ \bar{c} & \bar{d} \end{pmatrix} A = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

We at least have a method to calculate A^{-1} !

Let's come back...

- We have $(1, r(1)), (\omega, r(\omega)), (\omega^2, r(\omega^2)), \dots, (\omega^{D-1}, r(\omega^{D-1}))$, where $\omega = e^{\frac{2\pi i}{D}}$.

$$\begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{D-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{D-1} & \omega^{2(D-1)} & \dots & \omega^{(D-1)(D-1)} \end{bmatrix}}_{A(\omega)} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix}$$

Is A orthonormal (unitary)?

Two Different Columns

- We have $(1, r(1)), (\omega, r(\omega)), (\omega^2, r(\omega^2)), \dots, (\omega^{D-1}, r(\omega^{D-1}))$, where $\omega = e^{\frac{2\pi i}{D}}$.

$$\begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix} = \begin{bmatrix} 1 & \boxed{1} & \boxed{1} & \dots & 1 \\ 1 & \boxed{\omega} & \boxed{\omega^2} & \dots & \omega^{D-1} \\ 1 & \boxed{\omega^2} & \boxed{\omega^4} & \dots & \omega^{2(D-1)} \\ \vdots & \boxed{\vdots} & \boxed{\vdots} & \ddots & \vdots \\ 1 & \boxed{\omega^{D-1}} & \boxed{\omega^{2(D-1)}} & \dots & \omega^{(D-1)(D-1)} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix}$$

- $\langle \mathbf{c}_i, \mathbf{c}_j \rangle = \sum_{k=1}^D \overline{\omega^{(k-1)(i-1)}} \omega^{(k-1)(j-1)}$

$\bar{\omega}\omega = 1$

- $= \sum_{k=1}^D \omega^{(k-1)(j-i)} = \frac{1 - \omega^{(j-i)D}}{1 - \omega^{j-i}} = 0$

$\omega^D = e^{2\pi i} = 1$

The Same Column

- We have $(1, r(1)), (\omega, r(\omega)), (\omega^2, r(\omega^2)), \dots, (\omega^{D-1}, r(\omega^{D-1}))$, where $\omega = e^{\frac{2\pi i}{D}}$.

$$\begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \overset{c_i}{\boxed{1}} & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{D-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{D-1} & \omega^{2(D-1)} & \dots & \omega^{(D-1)(D-1)} \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix}$$

- $\langle \mathbf{c}_i, \mathbf{c}_i \rangle = \sum_{k=1}^D \overline{\omega^{(k-1)(i-1)}} \omega^{(k-1)(i-1)} = D$

$\bar{\omega}\omega = 1$

A is not unitary.

But we can scale it!

$\frac{1}{\sqrt{D}} A(\omega)$ is orthonormal!

$$\omega = e^{\frac{2\pi i}{D}}$$

Proposition. $\frac{1}{\sqrt{D}}A(\omega)$ is orthonormal for $\omega = e^{\frac{2\pi i}{D}}$.

Proof.

- Let $\mathbf{c}_i, \mathbf{c}_j$ be two arbitrary columns of $\frac{1}{\sqrt{D}}A(\omega)$.

$$\langle \mathbf{c}_i, \mathbf{c}_j \rangle = \sum_{k=1}^D \frac{1}{D} \cdot \overline{\omega^{(k-1)(i-1)}} \cdot \omega^{(k-1)(j-1)} = \frac{1}{D} \sum_{k=1}^D \omega^{(k-1)(j-i)}$$

- If $i = j$, we have $\langle \mathbf{c}_i, \mathbf{c}_j \rangle = \frac{1}{D} \sum_{k=1}^D \omega^0 = 1$;

- If $i \neq j$, then

$$\langle \mathbf{c}_i, \mathbf{c}_j \rangle = \frac{1}{D} \sum_{k=1}^D \omega^{(k-1)(j-i)} = \frac{1}{D} \frac{1 - \omega^{(j-i)D}}{1 - \omega^{j-i}} = 0$$

$$\omega^D = e^{2\pi i} = 1$$

- Thus, $\frac{1}{\sqrt{D}}A(\omega)$ is orthonormal.

Inverting $A(\omega)$...

- **Theorem.** If A is an orthonormal, then A is invertible and $A^{-1} = A^*$.

- **Proposition.** $\frac{1}{\sqrt{D}}A(\omega)$ is orthonormal for $\omega = e^{\frac{2\pi i}{D}}$.

- We have

$$A(\omega)^{-1} = \left(\sqrt{D} \cdot \frac{1}{\sqrt{D}} \cdot A(\omega) \right)^{-1} = \frac{1}{\sqrt{D}} \left(\frac{1}{\sqrt{D}} \cdot A(\omega) \right)^{-1} = \frac{1}{\sqrt{D}} \left(\frac{1}{\sqrt{D}} \cdot A(\omega) \right)^* = \frac{1}{D} A(\omega)^*$$

- Therefore,

$$(A(\omega)^{-1})_{i,j} = \frac{1}{D} \overline{(A(\omega))_{j,i}} = \frac{1}{D} \cdot \omega^{-(i-1)(j-1)} = \frac{1}{D} (\omega^{-1})^{(i-1)(j-1)},$$

which implies

$$A(\omega)^{-1} = \frac{1}{D} \cdot A(\omega^{-1}).$$

After blablabla math parts

Putting $A(\omega)^{-1} = \frac{1}{D} \cdot A(\omega^{-1})$ back

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix} = \frac{1}{D} \cdot \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(D-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \dots & \omega^{-2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(D-1)} & \omega^{-2(D-1)} & \dots & \omega^{-(D-1)(D-1)} \end{bmatrix} \cdot \begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix}$$

What we want...

- Naïve way also need $O(D^2)$ times!
- How to improve it?

This is A^{-1} !

Putting $A(\omega)^{-1} = \frac{1}{D} \cdot A(\omega^{-1})$ back

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{D-1} \end{bmatrix} = \frac{1}{D} \cdot \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(D-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \dots & \omega^{-2(D-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(D-1)} & \omega^{-2(D-1)} & \dots & \omega^{-(D-1)(D-1)} \end{bmatrix} \cdot \begin{bmatrix} r(1) \\ r(\omega) \\ r(\omega^2) \\ \vdots \\ r(\omega^{D-1}) \end{bmatrix}$$

- This is very similar to the first step!
- Let $s(x) = r(1) + r(\omega) \cdot x + r(\omega^2) \cdot x^2 \dots + r(\omega^{D-1}) \cdot x^{D-1}$
- Can we just apply **FFT**(s, ω^{-1})?

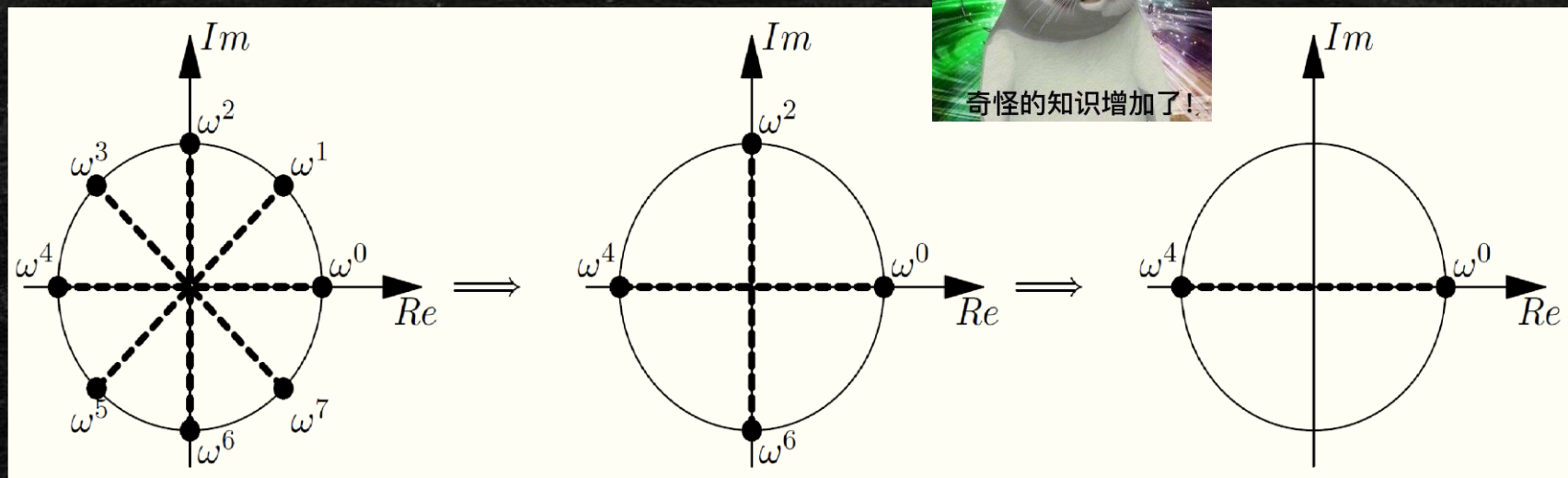
Problem

- In the first step.
- We chose D good numbers $\omega^0, \omega^1 \dots \omega^{D-1}$ to make our algorithm quick!
- But in this step.
- We must choose $\omega^0, \omega^{-1}, \omega^{-2} \dots \omega^{-(D-1)}$
- Are they still good?

Let us recall when
FFT (s, ω^{-1}) is good?

Is $(\omega^{-1}, \omega^{-2}, \dots, \omega^{-(D-1)})$ good?

- $(\omega^{-1}, \omega^{-2}, \dots, \omega^{-(D-1)})$ is just the same as $(\omega^1, \omega^2, \dots, \omega^{(D-1)})$ with a clockwise orientation!
- Yes, we can just apply $\text{FFT}(s, \omega^{-1})$!



Framework for FFT

$$p(x) = a_0 + a_1x + \cdots + a_{d-1}x^{d-1}$$
$$q(x) = b_0 + b_1x + \cdots + b_{d-1}x^{d-1}$$



Interpolation Step
(FFT)

$O(d \log d)$

$$(x_0, p(x_0)), (x_1, p(x_1)), \dots, (x_{2d-2}, p(x_{2d-2}))$$
$$(x_0, q(x_0)), (x_1, q(x_1)), \dots, (x_{2d-2}, q(x_{2d-2}))$$

Objective



$$r(x) = p(x) \cdot q(x)$$
$$= c_0 + c_1x + \cdots + c_{2d-2}x^{2d-2}$$



$O(d \log d)$

Recovery Step
(Inverse FFT)



$O(d)$



Multiplication

$$r(x_i) = p(x_i)q(x_i)$$

$$(x_0, r(x_0)), (x_1, r(x_1)), \dots, (x_{2d-2}, r(x_{2d-2}))$$

Putting 3 Steps Together

Putting Three Steps Together

Algorithm 2: Polynomial multiplication by FFT

Multiply(p, q): // p, q are two polynomials with degrees at most d

1. let D be the smallest integer power of 2 such that $d \leq \frac{D}{2}$;
2. let $\omega = e^{\frac{2\pi i}{D}}$;
3. $(p_0, p_1, \dots, p_{D-1}) \leftarrow \text{FFT}(p, \omega)$; // where $p_i = p(\omega^i)$
4. $(q_0, q_1, \dots, q_{D-1}) \leftarrow \text{FFT}(q, \omega)$; // where $q_i = q(\omega^i)$
5. for each $t = 0, 1, \dots, D - 1$: compute $r_t \leftarrow p_t \cdot q_t$
6. let $s(x) = \sum_{t=0}^{D-1} r_t x^t$
7. $(c_0, c_1, \dots, c_{D-1}) \leftarrow \text{FFT}(s, \omega^{-1})$;
8. let $r(x) = \sum_{t=0}^{D-1} \frac{c_t}{D} x^t$;
9. return r ;

Overall Time Complexity

$$O(d \log d) + O(d) + O(d \log d) = O(d \log d)$$