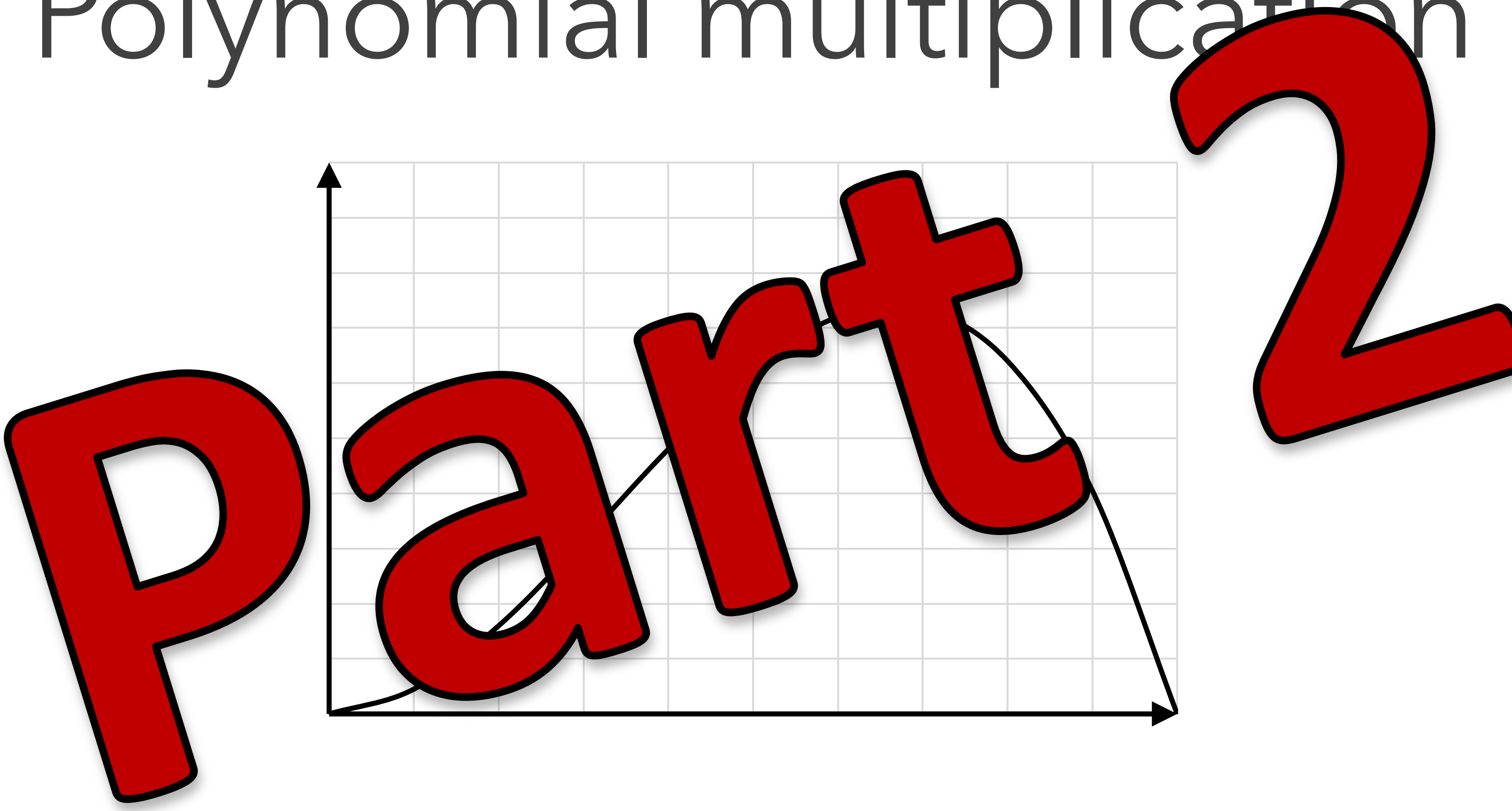
Lecture 5

Polynomial multiplication



Recap 1

We have two polynomials. We want to **multiply** them.

1. $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{n-1} x^{n-1}$
2. $q(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_{n-1} x^{n-1}$

Coefficient representation:

$(p_0,\ p_1,\ p_2, \ldots, p_{n-1})$ $\longrightarrow$ Multiply in $O(n^2)$ time.

Value representation:

$(p(\boldsymbol{\alpha_1}), p(\boldsymbol{\alpha_2}), \ldots, p(\boldsymbol{\alpha_m}))$ $\longrightarrow$ Multiply in $O(n)$ time.

$(\boldsymbol{m} \geq \boldsymbol{n})$

Main question:

Can we use **value** rep multiplication
to speed up **coefficient** rep multiplication?

Recap 2: Roots of Unity

n^{th} roots of unity (1)

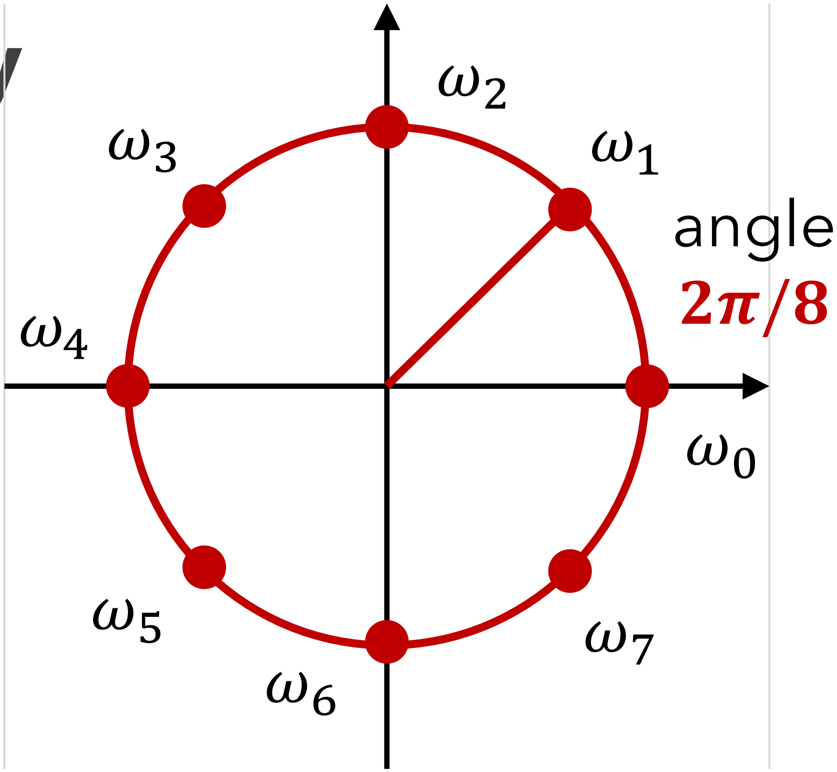
= {solutions to $x^n = 1$}

= n equally spaced points on unit circle

Example:

8^{th} roots of unity $= \sqrt[8]{1}$

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



8^{th} roots of unity

Generator fact: For all $0 \le i \le m-1$, $\omega_i = \omega_1^i$.

In addition, $\omega_1^0 = 1 = \omega_0 = \omega_1^m$.

Magical Fact: Squares of n^{th} roots = $(n/2)^{th}$ roots

(So squaring **halves** the number of roots)

Recap 3: algorithm outline

Coeffs of deg-$(n-1)$ p and q $\xrightarrow{\text{Evaluation}}$ Values $p(\alpha_1), \ldots, p(\alpha_m)$ $q(\alpha_1), \ldots, q(\alpha_m)$

Recall: $p \cdot q$ is degree-$(2n-2)$, so need $m \geq 2n-1$

Let $\boldsymbol{m}$ be first power of $\mathbf{2}$ such that $m \geq 2n-1$

Will evaluate p and q on $\boldsymbol{m^{th}}$ roots of unity

$$\{\omega_0, \omega_1, \ldots, \omega_{m-1}\}$$

in time $O(\boldsymbol{m \log(m)}) = O(\boldsymbol{n \log(n)})$.

This is the **Fast Fourier transform**.

Outline

1. Complex numbers
2. Polynomial multiplication I: fast evaluation
3. Polynomial multiplication II: fast interpolation
4. The matrix viewpoint
5. Applications

Fast Fourier Transform

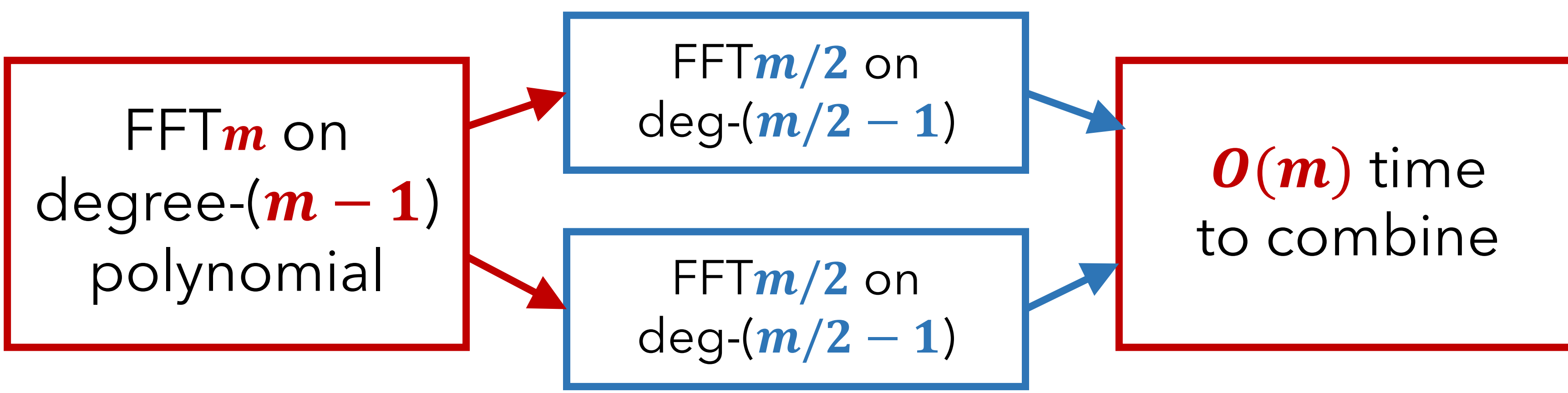
Input: 1. m, a power of two

2. $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-1} x^{m-1}$

Output: $p(\omega_0), p(\omega_1), \ldots, p(\omega_{m-1})$

where $\omega_0, \omega_1, \ldots, \omega_{m-1}$ are m^{th} roots of unity

Divide and conquer alg:



Runtime: $T(m) \leq 2 \cdot T(m/2) + O(m)$

$\Rightarrow T(m) = O(m \log(m))$ ✓

Divide and Conquer

Let's write out $p(x)$. And split it into two parts.

$$p_0 + p_1 x + p_2 x^2 + p_3 x^3 + p_4 x^4 + p_5 x^5 + p_6 x^6 + p_7 x^7 + \cdots$$

Even part:

$$\begin{aligned} & p_0 + p_2 x^2 + p_4 x^4 + p_6 x^6 + \cdots + p_{m-2} x^{m-2} \\ = \; & p_0 + p_2 (x^2) + p_4 (x^2)^2 + p_6 (x^2)^3 + \cdots \\ = \; & \mathbf{Even}(x^2), \\ & \quad \text{where } \mathbf{Even}(z) = p_0 + p_2 z + p_4 z^2 + p_6 z^3 + \cdots \end{aligned}$$

Odd part:

$$\begin{aligned} & p_1 x + p_3 x^3 + p_5 x^5 + p_7 x^7 + \cdots + p_{m-1} x^{m-1} \\ = \; & x \cdot (p_1 + p_3 x^2 + p_5 x^4 + p_7 x^6 + \cdots) \\ = \; & x \cdot \mathbf{Odd}(x^2), \\ & \quad \text{where } \mathbf{Odd}(z) = p_1 + p_3 z + p_5 z^2 + p_7 z^3 + \cdots \end{aligned}$$

$\therefore p(x) = \mathbf{Even}(x^2) + x \cdot \mathbf{Odd}(x^2)$

$\deg(\mathbf{Even}) = (m-2)/2 = (m/2 - 1)$. Same for $\deg(\mathbf{Odd})$.

Divide and Conquer

Fact: Let $p(x)$ have degree $(m-1)$.

Then $p(x) = \mathbf{Even}(x^2) + x \cdot \mathbf{Odd}(x^2)$,

where $\deg(\mathbf{Even}) = \deg(\mathbf{Odd}) = (m/2 - 1)$.

Goal: Compute $p(\boldsymbol{\omega_0}), p(\boldsymbol{\omega_1}), \dots, p(\boldsymbol{\omega_{m-1}})$ $(\boldsymbol{m^{th}}$ roots of unity)

$p(\boldsymbol{\omega_i}) = \mathbf{Even}(\boldsymbol{\omega_i^2}) + \boldsymbol{\omega_i} \cdot \mathbf{Odd}(\boldsymbol{\omega_i^2})$, $\longleftarrow$ $\boldsymbol{O(1)}$ more work!

$\boldsymbol{(m/2)^{th}}$ roots of unity **(Magical Fact!)**

To compute $p(\boldsymbol{\omega_i})$'s:

1. Inductively evaluate $\mathbf{Even}$ and $\mathbf{Odd}$ at

   $\boldsymbol{\alpha_0, \alpha_1, \dots, \alpha_{m/2-1}}$ $(\boldsymbol{(m/2)^{th}}$ roots of unity)

2. Do $\boldsymbol{O(m)}$ more work. $\therefore T(\boldsymbol{m}) \le 2 \cdot T(\boldsymbol{m/2}) + O(\boldsymbol{m})$

Example

Let's evaluate $p(x) = 1 + 3x - 4x^2 + 7x^3$

on the 4th roots of unity $= \sqrt[4]{1} = \{+1, -1, +i, -i\}$

$$\mathbf{Even}(z) = 1 - 4z, \qquad \mathbf{Odd}(z) = 3 + 7z$$

Check: $p(x) = \mathbf{Even}(x^2) + x \cdot \mathbf{Odd}(x^2)$

$$p(+1) = \mathbf{Even}(1) + \mathbf{Odd}(1)$$
$$p(-1) = \mathbf{Even}(1) - \mathbf{Odd}(1)$$

Repeated work!

$$p(+i) = \mathbf{Even}(-1) + i \cdot \mathbf{Odd}(-1)$$
$$p(-i) = \mathbf{Even}(-1) - i \cdot \mathbf{Odd}(-1)$$

Repeated work!

Reduces to evaluating **Even** and **Odd** on

$\{+1, -1\} = \sqrt{1} =$ the 2nd roots of unity

Outline

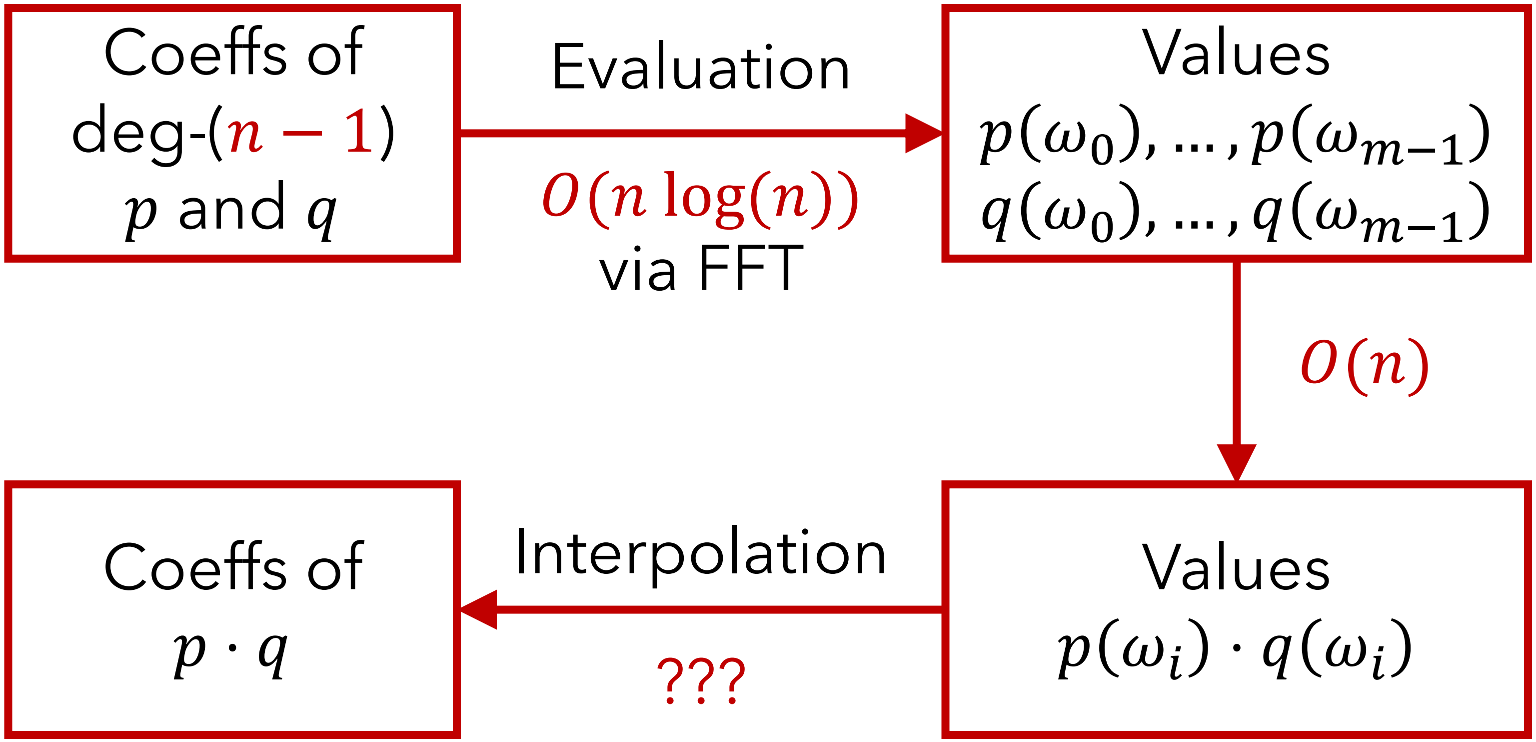
1. Complex numbers
2. Polynomial multiplication I: fast evaluation
3. Polynomial multiplication II: fast interpolation
4. The matrix viewpoint
5. Applications

Fast interpolation

3-minute break

and close the doors

Fast polynomial multiplication algorithm



Recall: $p\cdot\ q$ is degree-($2n-2$), so need $m \geq 2n-1$

Inverse FFT: Given $r(\omega_0), r(\omega_1), \dots, r(\omega_{m-1})$, returns

$$r(x) = r_0 + r_1 x + r_2 x^2 + \cdots + r_{m-1}x^{m-1}$$

in time $O(m\log(m))$.

Inverse FFT

Input: $p(\boldsymbol{\omega_0}), p(\boldsymbol{\omega_1}), \dots, p(\boldsymbol{\omega_{m-1}})$

Output: $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-1} x^{m-1}$

FT formula:

$$p(\boldsymbol{\omega_\ell}) = \sum_{j=0}^{m-1} p_j \cdot (\boldsymbol{\omega_\ell})^j$$

Inverse FT formula:

$$p_\ell = \frac{1}{m} \cdot \sum_{j=0}^{m-1} p(\omega_j) \cdot (\boldsymbol{\omega_{m-\ell}})^j$$

$$= \frac{1}{m} \cdot q(\boldsymbol{\omega_{m-\ell}}),$$

$$q(x) = p(\omega_0) + p(\omega_1)x + \cdots + p(\omega_{m-1})x^{m-1}$$

$\therefore$ Can compute $p_0, p_1, \dots, p_{m-1}$ by evaluating q at

$$\omega_0, \omega_1, \dots, \omega_{m-1}$$

Just an FFT! Time $O(n \log n)$.

Inverse FFT

Input: $p(\boldsymbol{\omega_0}), p(\boldsymbol{\omega_1}), \ldots, p(\boldsymbol{\omega_{m-1}})$

Output: $p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-1} x^{m-1}$

FT formula:

$$p(\boldsymbol{\omega_\ell}) = \sum_{j=0}^{m-1} p_j \cdot (\boldsymbol{\omega_\ell})^j$$

Inverse FT formula:

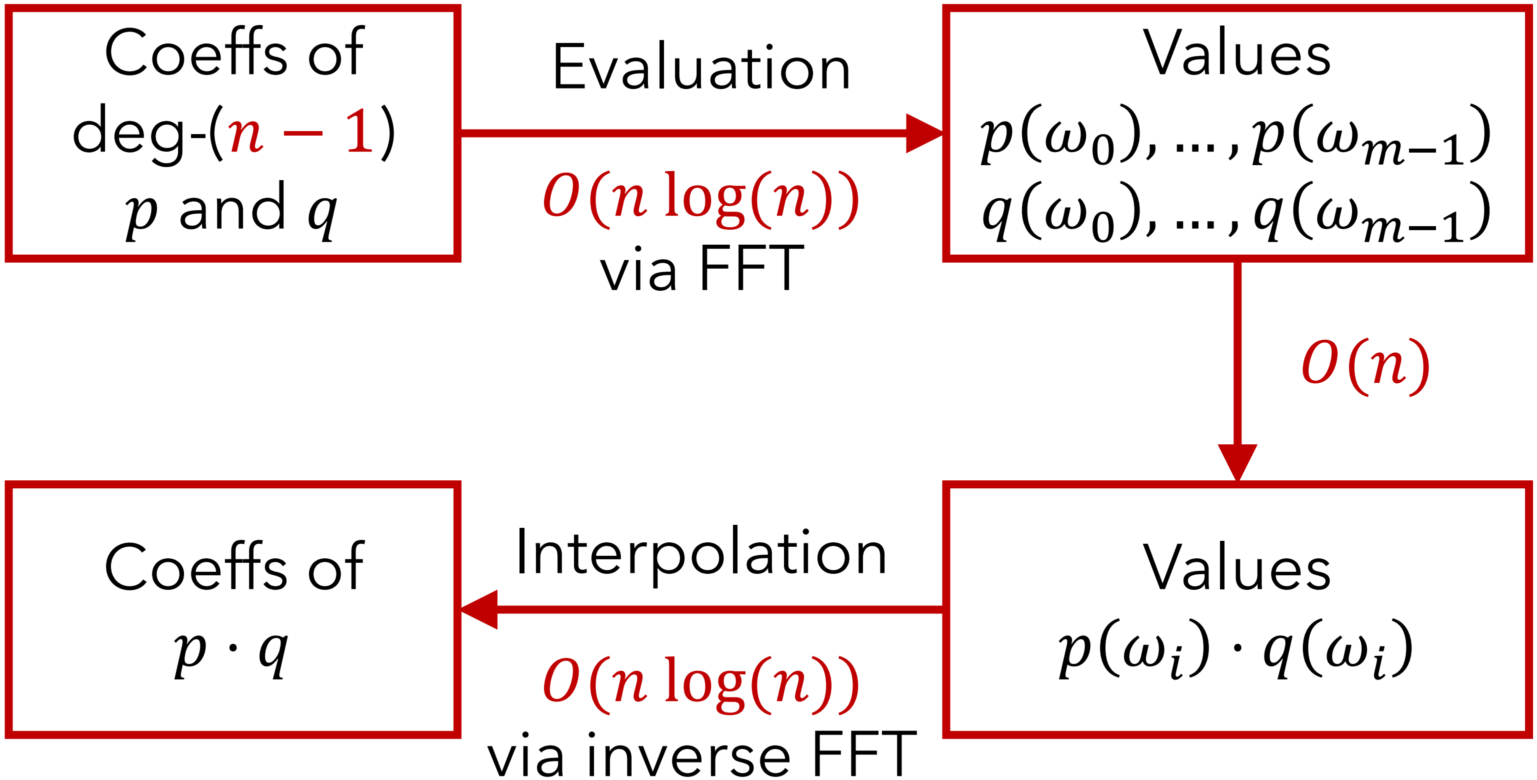
$$p_\ell = \frac{1}{m} \cdot \sum_{j=0}^{m-1} \underbrace{p(\omega_j)}_{(\star)} \cdot (\boldsymbol{\omega_{m-\ell}})^j$$

Proof of Inverse FT formula:

1. Plug **FT formula** into $(\star)$
2. Use **Generator fact** on all $\boldsymbol{\omega_\ell}$'s
   $\omega_i = \omega_1^i$, etc.
3. Use this formula:

$$\sum_{k=0}^{m-1} \omega_j^k = \begin{cases} m & \text{if } j = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Fast polynomial multiplication algorithm



Finishes the proof of
polynomial multiplication in $O(n\log(n))$ time!

Outline

- [x] 1. Complex numbers
- [x] 2. Polynomial multiplication I: fast evaluation
- [x] 3. Polynomial multiplication II: fast interpolation

4. The matrix viewpoint

5. Applications

The matrix viewpoint

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-1} x^{m-1}$$

$$\begin{bmatrix} p(\omega_0) \\ p(\omega_1) \\ p(\omega_2) \\ \dots \\ p(\omega_{m-1}) \end{bmatrix} = \left.\begin{bmatrix} p_0 + p_1\omega_0 + p_2\omega_0^2 + \dots + p_{m-1}\omega_0^{m-1} \\ p_0 + p_1\omega_1 + p_2\omega_1^2 + \dots + p_{m-1}\omega_1^{m-1} \\ p_0 + p_1\omega_2 + p_2\omega_2^2 + \dots + p_{m-1}\omega_2^{m-1} \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ p_0 + p_1\omega_{m-1} + p_2\omega_{m-1}^2 + \dots + p_{m-1}\omega_{m-1}^{m-1} \end{bmatrix}\right\} v$$

$$= \underbrace{\begin{bmatrix} 1 & \omega_0 & \omega_0^2 & \dots & \omega_0^{m-1} \\ 1 & \omega_1 & \omega_1^2 & \dots & \omega_1^{m-1} \\ 1 & \omega_2 & \omega_2^2 & \dots & \omega_2^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_{m-1} & \omega_{m-1}^2 & \dots & \omega_{m-1}^{m-1} \end{bmatrix}}_{M} \cdot \underbrace{\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \dots \\ p_{m-1} \end{bmatrix}}_{[p]}$$

$$p(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{m-1} x^{m-1}$$

$$\begin{bmatrix} p(\omega_0) \\ p(\omega_1) \\ p(\omega_2) \\ \dots \\ p(\omega_{m-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \omega_0 & \omega_0^2 & \dots & \omega_0^{m-1} \\ 1 & \omega_1 & \omega_1^2 & \dots & \omega_1^{m-1} \\ 1 & \omega_2 & \omega_2^2 & \dots & \omega_2^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_{m-1} & \omega_{m-1}^2 & \dots & \omega_{m-1}^{m-1} \end{bmatrix}}_{M} \cdot \underbrace{\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \dots \\ p_{m-1} \end{bmatrix}}_{[p]}$$

Naively, the matrix-vector multiplication $\boldsymbol{M} \cdot [\boldsymbol{p}]$ takes $\boldsymbol{O(m^2)}$ time

But FFT solves this problem in $\boldsymbol{O(m \log(m))}$ time!

Note: it solves this ***<u>***without ever writing down M***</u>***

(In fact, we didn't even know it was matrix-vector multiplication when we designed FFT!)

Fourier transform:

$$\begin{bmatrix} p(\omega_0) \\ p(\omega_1) \\ p(\omega_2) \\ \dots \\ p(\omega_{m-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \omega_0 & \omega_0^2 & \dots & \omega_0^{m-1} \\ 1 & \omega_1 & \omega_1^2 & \dots & \omega_1^{m-1} \\ 1 & \omega_2 & \omega_2^2 & \dots & \omega_2^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \omega_{m-1} & \omega_{m-1}^2 & \dots & \omega_{m-1}^{m-1} \end{bmatrix}}_{M} \cdot \underbrace{\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \dots \\ p_{m-1} \end{bmatrix}}_{[p]}$$

Inverse Fourier transform:

$$M^{-1} \cdot \begin{bmatrix} p(\omega_0) \\ p(\omega_1) \\ p(\omega_2) \\ \dots \\ p(\omega_{m-1}) \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \dots \\ p_{m-1} \end{bmatrix}$$

Formulas:

$$M_{ij} = \omega_i^j = \left(\omega_1^i\right)^j = \omega_1^{i \cdot j}$$

$$(M^{-1})_{ij} = \frac{1}{n} \cdot \omega_{m-i}^j$$

(from the **Inverse FT formula**)

Outline

- [x] 1. Complex numbers
- [x] 2. Polynomial multiplication I: fast evaluation
- [x] 3. Polynomial multiplication II: fast interpolation
- [x] 4. The matrix viewpoint
- [ ] 5. Applications

Applications of polynomial multiplication and the FFT

Let's look at polynomial multiplication again…

Given: 1. $p(x) = p_0 + p_1 x + p_2 x^2 + p_3 x^3$

2. $q(x) = q_0 + q_1 x + q_2 x^2 + q_3 x^3$

Consider $(p_0 + p_1 x + p_2 x^2 + p_3 x^3) \cdot (q_0 + q_1 x + q_2 x^2 + q_3 x^3)$.

Q: What does the x^2 term look like?

$$p_0 q_2 x^2 + p_1 q_1 x^2 + p_2 q_0 x^2$$

Q: What does the x^3 term look like?

$$p_0 q_3 x^3 + p_1 q_2 x^3 + p_2 q_1 x^3 + p_3 q_0 x^3$$

General rule: coefficient on $\boldsymbol{x^i}$ in $p(x) \cdot q(x)$ is

$$p_0 q_i + p_1 q_{i-1} + \cdots + p_{i-1} q_1 + p_i q_0$$

General rule: coefficient on $\boldsymbol{x^i}$ in $p(x) \cdot q(x)$ is

$$p_0 q_i + p_1 q_{i-1} + \cdots + p_{i-1} q_1 + p_i q_0$$

Consider two arrays:

1. $[p_0 \quad p_1 \quad p_2 \quad p_3]$
2. $[q_3 \quad q_2 \quad q_1 \quad p_0]$

Stack them on top of each other.

Take the dot product of the overlap.

$$\begin{array}{cccccc} & & [p_0 & p_1 & p_2 & p_3] \\ [q_3 & q_2 & q_1 & q_0] & & \end{array} \qquad \begin{aligned} &= p_0 \cdot q_1 + p_1 \cdot q_0 \\ &= \text{coefficient on } \boldsymbol{x^1} \end{aligned}$$

These "**shifted dot products**" are equal to coefficients of $p(x) \cdot q(x)$

Hence, can compute them in $\boldsymbol{O(n \log(n))}$ time

Application #1: Cross correlation

Input: Two arrays 1. $[p_0 \quad p_1 \quad p_2 \quad p_3 \quad \cdots \quad p_{n-1}]$

2. $[q_0 \quad q_1 \quad q_2 \quad q_3 \quad \cdots \quad q_{n-1}]$

Goal: Compute all the shifted dot products

$$\begin{array}{ccccccc} [p_0 & p_1 & p_2 & \boxed{\begin{array}{c} p_3] \\ [q_0 \end{array}} & q_1 & q_2 & q_3] \end{array}$$

This is called "**cross correlation**"

Roughly, it measures how well different segments of p and q "overlap" with each other.

Can solve in $\boldsymbol{O(n\log(n))}$ time via polynomial multiplication!

(Look out for this in recitations, on hwks, etc.)

(Might sometimes involve two vectors of unequal lengths!)

Application #2: Integer Multiplication

Say you want to multiply two $\boldsymbol{n}$-digit integers…

$$a = a_{n-1} \cdots a_2 a_1 a_0 \qquad \text{(each digit between 0 and 9)}$$

$$b = b_{n-1} \cdots b_2 b_1 b_0$$

So write down these two polynomials:

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

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

$O(n \log(n))$ time, right?

Note that $A(10) = a$ and $B(10) = b$

Now, compute $A(x) \cdot B(x)$ and plug in $x = 10$

$$A(10) \cdot B(10) = a \cdot b$$

Is this an $O(n \log(n))$ time alg for integer multiplication?

Not quite…

Application #2: Integer Multiplication

Remember the modeling assumption we made?
That all additions and multiplications are $O(1)$ time?
Strictly speaking, it's not 100% true!

Consider the x^{n-1} coefficient in $A(x) \cdot B(x)$:

$$a_0 b_{n-1} + a_1 b_{n-2} + \cdots + a_{n-2} b_1 + a_{n-1} b_0$$

Can be as large as $\Theta(n)$!

Need more than $O(1)$ time to add/multiply this.

To analyze this integer mult alg,
need to keep track of this stuff.

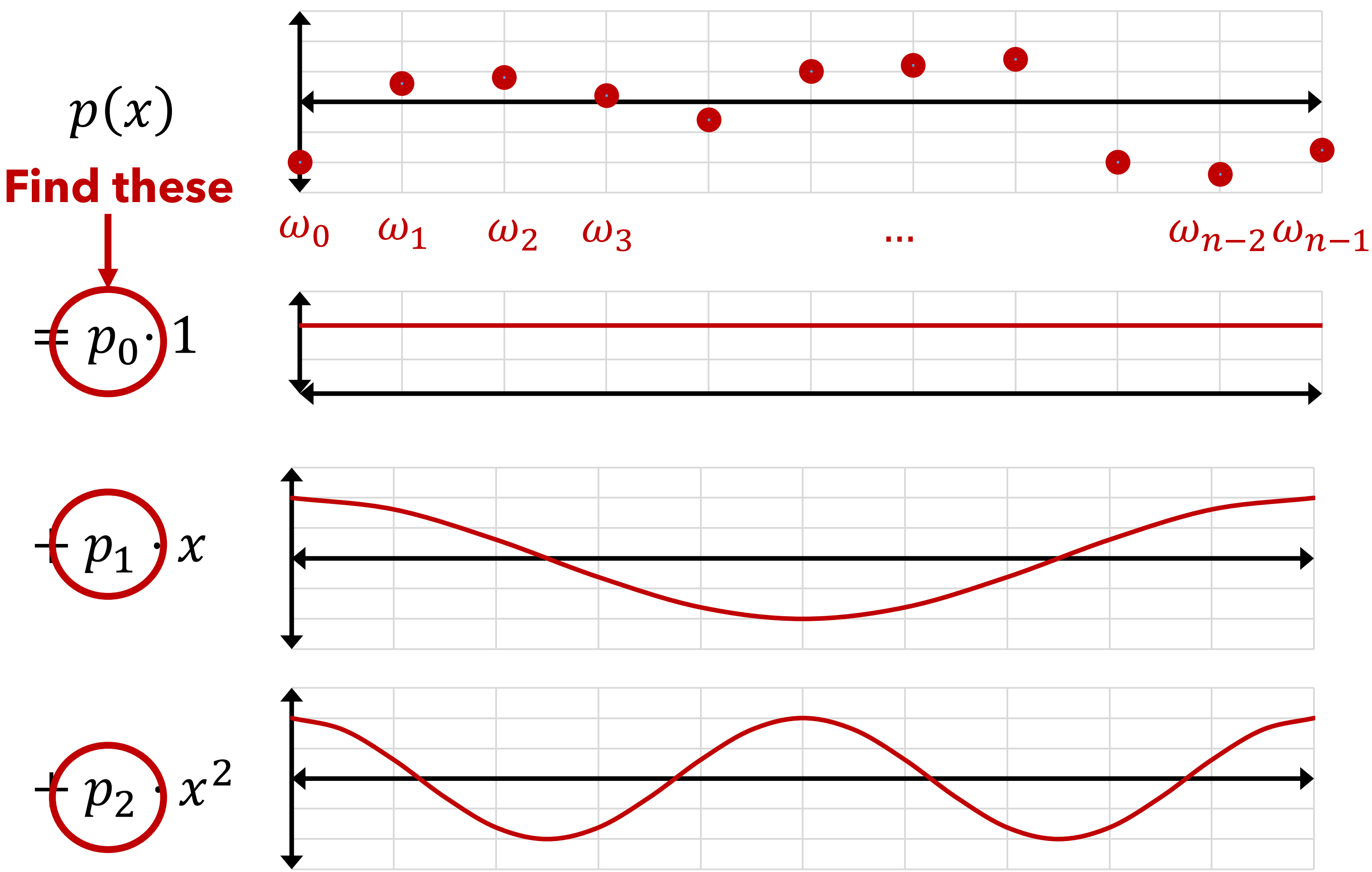
A bit tedious!

Result: $O(n \log(n) \log(\log(n)))$ time algorithm.

[Schönhage–Strassen algorithm]

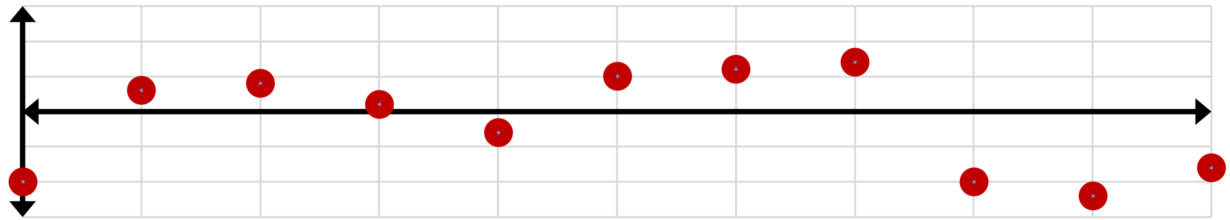
Application #3: Fourier Transform

Inverse FT: Given $p(\boldsymbol{\omega_0}), p(\boldsymbol{\omega_1}), \ldots, p(\boldsymbol{\omega_{m-1}})$,
compute $p(x) = p_0 + p_1 x + \cdots + p_{m-1} x^{m-1}$



Application #3: Fourier Transform

Given data that looks like this:



decompose it into sin/cosine waves.

Applications:

1. Really, too many to mention!
2. Music software
3. Heart rate monitor
4. Signal processing (e.g. cell phones)
5. The ability to do **~*~ultrafast~*~ FTs** is at the heart of many **quantum** algs

History

1822: Joseph Fourier “discovers” the Fourier transform

1805: Gauss discovers the Fast Fourier Transform

“The method greatly reduces the tediousness of mechanical calculations.”

1828+: FFT rediscovered 10+ times

Principal Discoveries of Efficient Methods of Computing the DFT

Researcher(s)	Date	Sequence Lengths	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	—	Harmonic analysis of barometric pressure
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	$2^n k$	All	Harmonic analysis of functions
K. Stumpff [16]	1939	$2^n k$, $3^n k$	All	Harmonic analysis of functions
Danielson and Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley and Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

Source: “Gauss and the History of the Fast Fourier Transform” by Heideman, Johnson, and Burrus

History

Modern rediscovery in 1965:

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

Original application:

Detecting nuclear-weapons tests from the Soviet Union by analyzing seismographic data.

Outline

- ☑ 1. Complex numbers
- ☑ 2. Polynomial multiplication I: fast evaluation
- ☑ 3. Polynomial multiplication II: fast interpolation
- ☑ 4. The matrix viewpoint
- ☑ 5. Applications