CS240 Algorithm Design and Analysis

Lecture 4

Divide and Conquer (Cont.)

Quan Li
Fall 2025
2025.09.23

Last Time – What you need to know

- **Basic idea**
  - Make the locally optimal choice at each step
- **Algorithms**
  - Optimal Caching
    - Evict item that is requested farthest in future
- **Proof skills**
  - **Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's
  - **Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality
- **Mergesort**
- **Closest Pair of Points: Analysis**

Integer Multiplication (Revisit)

Integer Arithmetic

- **Add.** Given two n-digit integers a and b, compute a + b.
  - O(n) bit operations

- **Multiply.** Given two n-digit integers a and b, compute a * b
  - Brute force solution: $\Theta(n^2)$ bit operations

	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Add

```
                1 1 0 1 0 1 0 1 0
              * 0 1 1 1 1 1 0 1
              -------------------
                1 1 0 1 0 1 0 1 0
              0 0 0 0 0 0 0 0 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
---------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0
```

Multiply

Divide-and-Conquer Multiplication: Warmup

- To multiply two n-digit integers:
  - Multiply four ½n - digit integers
  - Add two ½n-digit integers, and shift to obtain result

$$x = \underbrace{1\,0\,0\,0}_{x_1}\ \underbrace{1\,1\,0\,1}_{x_0}$$

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \left(x_1 y_0 + x_0 y_1\right) + x_0 y_0
\end{aligned}
$$

$$\mathrm{T}(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow \mathrm{T}(n) = \Theta(n^2)$$

assumes n is a power of 2

Karatsuba Multiplication

- To multiply two n-digit integers:
  - Add two ½n digit integers
  - Multiply **three** ½n-digit integers
  - Add, subtract, and shift ½n-digit integers to obtain results

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^{n} \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^{n} \cdot \underbrace{x_1 y_1}_{A} + 2^{n/2} \cdot \big(\underbrace{(x_1 + x_0)(y_1 + y_0)}_{B} - \underbrace{x_1 y_1}_{A} - \underbrace{x_0 y_0}_{C} \big) + \underbrace{x_0 y_0}_{C}
\end{aligned}
$$

- **Theorem.** Can multiply two n-digit integers in $O(n^{1.585})$ bit operations

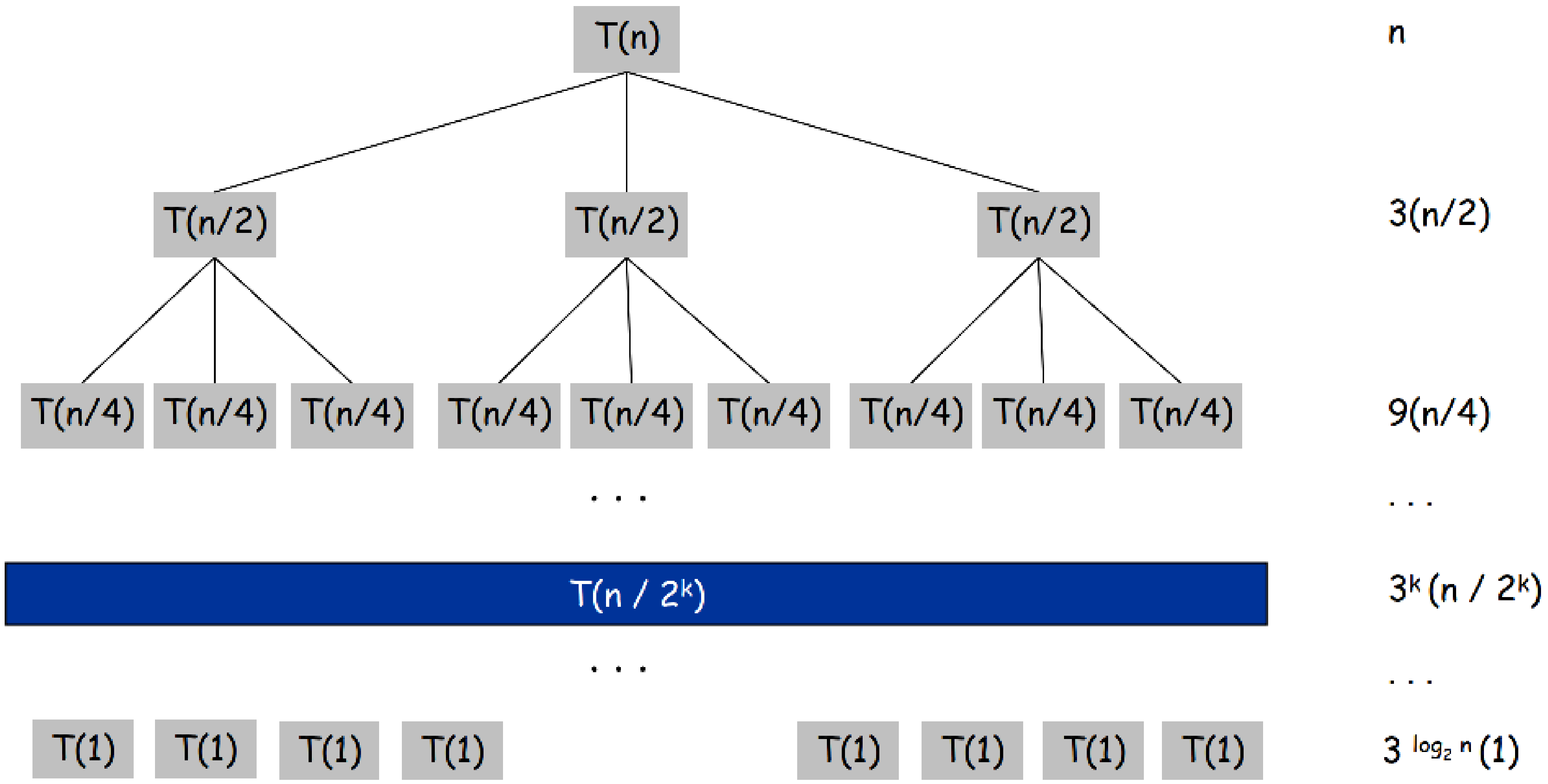
$$
T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
$$

$$
\Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585})
$$

Karatsuba: Recursion Tree

$$T(n)=\begin{cases} 1 & \text{if } n=1 \\ 3T(n/2)+n & \text{otherwise} \end{cases}$$

$$T(n)=\sum_{k=0}^{\log_2 n} n\left(\tfrac{3}{2}\right)^k = \frac{\left(\tfrac{3}{2}\right)^{1+\log_2 n}-1}{\tfrac{3}{2}-1}\, n = 3n^{\log_2 3}-2n$$



Matrix Multiplication

Matrix Multiplication

- **Matrix multiplication. Given two n-by-n matrices A and B, compute C= AB**

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

- **Brute force.** $\Theta(n^3)$ arithmetic operations.

- **Fundamental question.** Can we improve upon brute force?

Matrix Multiplication: Warmup

上海科技大学 ShanghaiTech University

- **Divide-and-conquer.**
  - Divide: partition A and B into ½n-by-½n blocks
  - Conquer: multiply 8 ½n-by-½n recursively
  - Combine: add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)$$

立志成才报国裕民

Matrix Multiplication: Key Idea

- **Key idea.** Multiply 2-by-2 block matrices with only 7 multiplications

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions)

Fast Matrix Multiplication

- **Fast matrix multiplication (Strassen, 1969)**
  - Divide: partition A and B into ½n-by-½n blocks
  - Compute: 14 ½n-by-½n matrices via 10 matrix additions
  - Conquer: multiply 7 ½n-by-½n matrices recursively
  - Combine: 7 products into 4 terms using 8 matrix additions

- **Analysis**
  - Assume n is a power of 2
  - T(n) = # arithmetic operations

$$\mathrm{T}(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow \mathrm{T}(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Fast Matrix Multiplication in Practice

- **Common misperception:** “Strassen is only a theoretical curiosity.”
  - Advanced computation group at apple computer reports 8* speedup on G4 Velocity Engine when n ~ 2,500
  - Range of instances where it’s useful is a subject of controversy

- **Remark.** Can “Strassenize” Ax = b, determinant, eigenvalues, and other matrix ops

Fast Matrix Multiplication in Theory

- **Q.** Multiply two 2-by-2 matrices with only 7 scalar multiplications?
- **A.** Yes! [Strassen, 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.81})$$

- **Q.** Multiply two 2-by-2 matrices with only 6 scalar multiplications?
- **A.** Impossible. [Hopcroft and Kerr, 1971]

$$\Theta(n^{\log_2 6}) = O(n^{2.59})$$

- **Q.** Two 3-by-3 matrices with only 21 scalar multiplications?
- **A.** Also impossible.

$$\Theta(n^{\log_3 21}) = O(n^{2.77})$$

- **Q.** Two 70-by-70 matrices with only 143,640 scalar multiplications?
- **A.** Yes!

$$\Theta(n^{\log_{70} 143640}) = O(n^{2.80})$$

Fast Matrix Multiplication in Theory

- **Decimal wars**
  - December 1979: $O(n^{2.521813})$
  - January 1980: $O(n^{2.521801})$
  - …
  - 1987: $O(n^{2.375477})$
  - 2010: $O(n^{2.374})$
  - 2011: $O(n^{2.3728642})$
  - 2014: $O(n^{2.3728639})$
- **Best known.** $O(n^{2.3728639})$ [Francois Gall, 2014]
- **Conjecture.** $O(2+\boldsymbol{\varepsilon})$ for any $\boldsymbol{\varepsilon} > 0$
- **Caveat.** Theoretical improvements to Strassen are progressively less practical

Convolution and FFT

Fast Fourier Transform: Applications

- **Applications**
  - Optics, acoustics, quantum physics, telecommunications, control systems, signal processing, speech recognition, data compression, image processing
  - DVD, JPEG, MP3, MRI, CAT scan
  - Numerical solutions to Poisson's equation

> The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. *-Charles van Loan*

Polynomials: Coefficient Representation

- **Polynomials: [coefficient representation]**

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

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

- **Add: O(n) arithmetic operations**

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1}$$

- **Evaluate: O(n) using Horner's method**

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

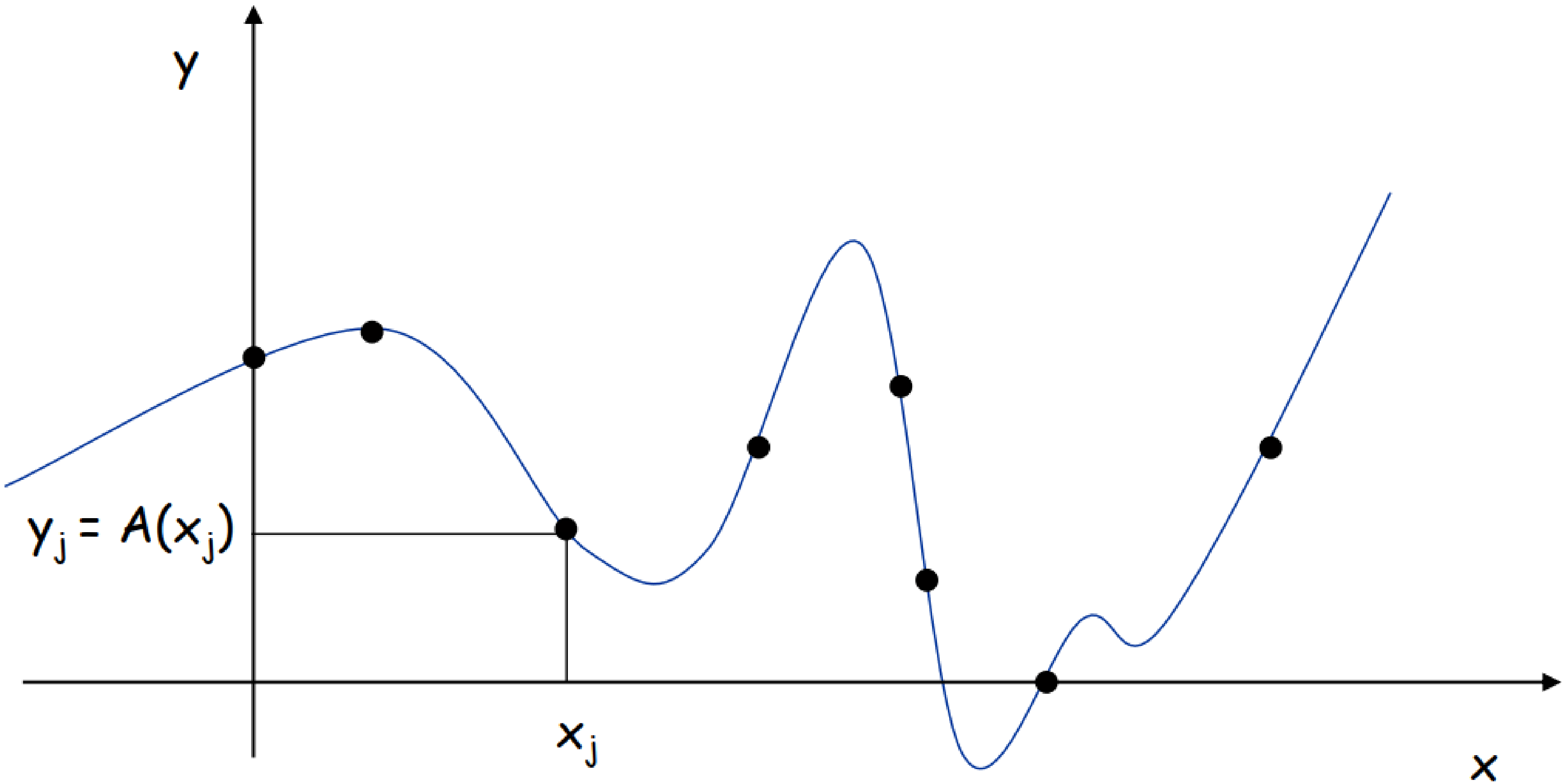
- **Multiply (convolve): O(n^2) using brute force**

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

立志 成才 报国 裕民

Polynomials: Point-Value Representation

- A degree n-1 polynomial A(x) is uniquely specified by its evaluation at n distinct values of x.



Polynomials: Point-Value Representation

- Polynomial: [point-value representation]

$$A(x): \ (\mathrm{x}_0, y_0), \ \ldots, (\mathrm{x}_{\mathrm{n}\text{-}1}, y_{n-1})$$
$$B(x): \ (\mathrm{x}_0, z_0), \ \ldots, (\mathrm{x}_{\mathrm{n}\text{-}1}, z_{n-1})$$

- Add: O(n) arithmetic operation

$$A(x)+B(x): \ (\mathrm{x}_0, y_0+z_0), \ldots, (\mathrm{x}_{\mathrm{n}\text{-}1}, y_{n-1}+z_{n-1})$$

- Multiply: O(n), but need 2n-1 points

$$A(x)\times B(x): \ (\mathrm{x}_0, y_0\times z_0), \ldots, (\mathrm{x}_{2\mathrm{n}\text{-}1}, y_{2n-1}\times z_{2n-1})$$

- Evaluate: $O(n^2)$ using Lagrange's formula

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

Converting Between Two Polynomial Representations

- Tradeoff. Fast evaluation or fast multiplication. We want both!

Representation	Multiply	Evaluate
Coefficient	$O(n^2)$	$O(n)$
Point-value	$O(n)$	$O(n^2)$

- Goal. Make all opt fast by efficiently converting between two representations

$$a_0, a_1, \ldots, a_{n-1} \rightleftarrows (x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Converting Between Two Polynomial Representations: Brute Force

- **Coefficient → point-value.** Given a polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Running time. $O(n^2)$ for matrix-vector multiplication or n times Horner’s method

Converting Between Two Polynomial Representations: Brute Force

- **Point-value → Coefficient.** Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ that has given values at given points.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Running time. $O(n^3)$ for Gaussian elimination.

Coefficient to Point-Value Representation: Intuition

- **Coefficient to point-value.** Given a polynomial $a_0 + a_1 + \ldots + a_{n-1}x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

We can choose which points!

- **Divide.** Break polynomial up into even and odd powers.

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$$
$$A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$$
$$A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$$
$$A(x) = A_{even}(x^2) + x\,A_{odd}(x^2).$$
$$A(-x) = A_{even}(x^2) - x\,A_{odd}(x^2).$$

- **Intuition.** Choose two points to be ± 1

$$A(1) = A_{even}(1) + 1\,A_{odd}(1).$$
$$A(-1) = A_{even}(1) - 1\,A_{odd}(1).$$

Can evaluate polynomial of degree <= n at 2 points by evaluating two polynomials of degree <= ½n at 1 points

Coefficient to Point-Value Representation: Intuition

- **Coefficient to point-value.** Given a polynomial $a_0 + a_1 + \ldots + a_{n-1}x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

We can choose which points!

- **Divide.** Break polynomial up into even and odd powers.

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$$
$$A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$$
$$A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$$
$$A(x) = A_{even}(x^2) + x\,A_{odd}(x^2).$$
$$A(-x) = A_{even}(x^2) - x\,A_{odd}(x^2).$$

- **Intuition.** Choose four complex points to be $\pm 1, \pm i$.

$$A(1) = A_{even}(1) + 1\,A_{odd}(1).$$
$$A(-1) = A_{even}(1) - 1\,A_{odd}(1).$$
$$A(i) = A_{even}(-1) + i\,A_{odd}(-1).$$
$$A(-i) = A_{even}(-1) - i\,A_{odd}(-1).$$

Can evaluate polynomial of degree <= n at 4 points by evaluating two polynomials of degree <= ½n at 2 points

Coefficient to Point-Value Representation: Intuition

- **Coefficient to point-value.** Given a polynomial $a_0 + a_1 + \ldots + a_{n-1}x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

We can choose which points!

- **Divide.** Break polynomial up into even and odd powers.

$$
\begin{aligned}
A(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7. \\
A_{even}(x) &= a_0 + a_2x + a_4x^2 + a_6x^3. \\
A_{odd}(x) &= a_1 + a_3x + a_5x^2 + a_7x^3.
\end{aligned}
$$

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

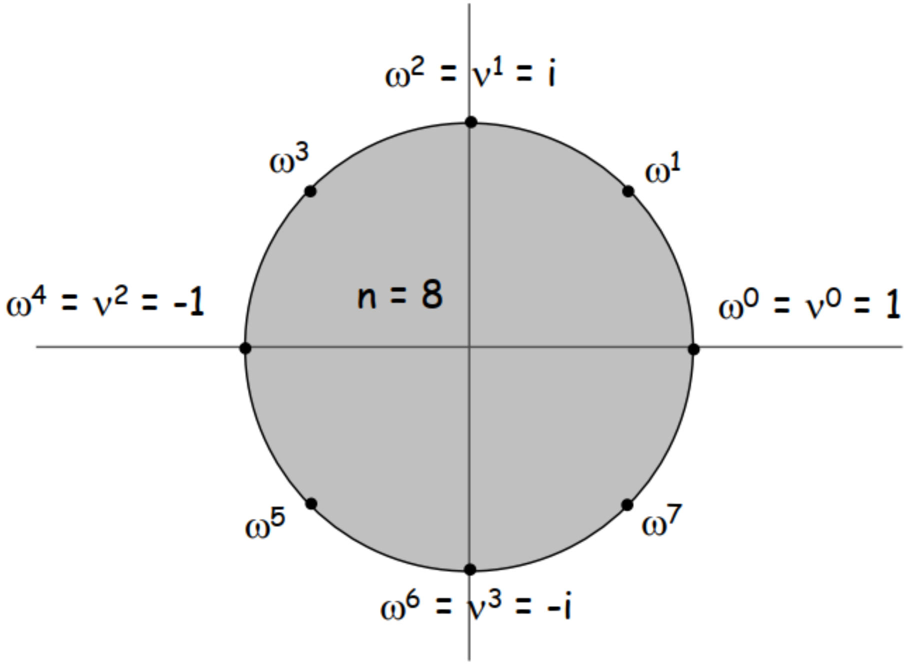
$$A(-x) = A_{even}(x^2) - x\ A_{odd}(x^2).$$

- **Goal.** Choose n points s.t.
  - Can evaluate polynomial of degree <= n at n points by evaluating two polynomials of degree <= ½n at ½n point
  - But also: can evaluate polynomial of degree <= ½n at ½n points by evaluating two polynomials of degree <= ¼n at ¼n point, and so on

Roots of Unity

- **Def.** An **n^{th} root of unity** is a complex number x such that $x^n = 1$
- **Fact.** The n^{th} roots of unity are : $\boldsymbol{\omega}^0, \boldsymbol{\omega}^1, \ldots, \boldsymbol{\omega}^{n-1}$ where $\boldsymbol{\omega} = e^{2\pi i/n}$
- **Pf.** $(\boldsymbol{\omega}^k)^n = (e^{2\pi ik/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$

- **Fact.** The ½n^{th} roots of unity are: $v^0, v^1, \ldots, v^{n/2-1}$ where $v = e^{4\pi i/n}$
- **Fact.** $\boldsymbol{\omega}^2 = v$ and $(\boldsymbol{\omega}^2)^k = v^k$



Discrete Fourier Transform

- **Coefficient to point-value.** Given a polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$
- **Key idea:** choose $x_k = \boldsymbol{\omega}^k$ where $\boldsymbol{\omega}$ is principal n^{th} root of unity

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Discrete Fourier Transform

Fourier matrix F_n

Fast Fourier Transform

- **Goal.** Evaluate a degree n-1 polynomial $A(x) = a_0 + \ldots + a_{n-1}x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$

- **Divide.** Break polynomial up into even and odd powers

$$
\begin{aligned}
A_{even}(x) &= a_0 + a_2 x + a_4 x^2 + \ldots + a_{n/2-2}\, x^{(n-1)/2}. \\
A_{odd}(x) &= a_1 + a_3 x + a_5 x^2 + \ldots + a_{n/2-1}\, x^{(n-1)/2}. \\
A(x) &= A_{even}(x^2) + x\, A_{odd}(x^2).
\end{aligned}
$$

- **Conquer.** Evaluate degree $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2}n^{th}$ roots of unity: $v^0, v^1, \ldots, v^{n/2-1}$

- **Combine**

$$
\begin{aligned}
A(\omega^k) &= A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2 \\
A(\omega^{k+n/2}) &= A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k), \quad 0 \le k < n/2
\end{aligned}
$$

$$\omega^{k+n/2} = -\omega^k$$

$$\nu^k = (\omega^k)^2 = (\omega^{k+n/2})^2$$

FFT Algorithm


```
FFT(n, a₀,a₁,…,aₙ₋₁) {
    if (n == 1) return a₀

    (e₀,e₁,…,e_{n/2-1}) ← FFT(n/2, a₀,a₂,a₄,…,aₙ₋₂)
    (d₀,d₁,…,d_{n/2-1}) ← FFT(n/2, a₁,a₃,a₅,…,aₙ₋₁)

    for k = 0 to n/2 - 1 {
        ω^k ← e^{2πik/n}
        y_k     ← e_k + ω^k d_k
        y_{k+n/2} ← e_k - ω^k d_k
    }

    return (y₀,y₁,…,yₙ₋₁)
}
```

FFT Summary

- Theorem. FFT algorithm evaluates a degree n-1 polynomial at each of the n^{th} roots of unity in O(nlogn) steps

assumes n is a power of 2

- Pf. T(2n) = 2T(n) + O(n) → T(n) = O(nlogn)

$$a_0, a_1, \ldots, a_{n-1} \xrightarrow{O(n \log n)} (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

$$a_0, a_1, \ldots, a_{n-1} \longleftarrow (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Point-Value to Coefficient Representation: Inverse DFT

- **Goal.** Given the values $y_0, \ldots, y_{n-1}$ of a degree n-1 polynomial at the n points $\boldsymbol{\omega}^0, \boldsymbol{\omega}^1, \ldots, \boldsymbol{\omega}^{n-1}$, find unique polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ that has given values at given points.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

↑ Inverse DFT

↑ Fourier matrix inverse $(F_n)^{-1}$

Inverse FFT

上海科技大学 ShanghaiTech University

- **Claim.** Inverse of Fourier matrix is given by following formula.

$$G_n = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Inverse FFT: Proof of Correctness

- **Claim.** F_n and G_n are inverses.
- **Pf.**

$$\left(F_n\, G_n\right)_{k\,k'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj}\, \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} \underset{\substack{\uparrow \\ \text{summation lemma}}}{=} \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

- **Summation lemma.** Let $\boldsymbol{\omega}$ be a principal n^{th} root of unity Then

$$\sum_{j=0}^{n-1} \omega^{kj} = \begin{cases} n & \text{if } k \equiv 0 \bmod n \\ 0 & \text{otherwise} \end{cases}$$

- **Pf.**

If k is a multiple of n then $\omega^k = 1 \Rightarrow$ sums to n.

Else: $\omega^k \neq 1$

- $x^n - 1 = (x - 1)(1 + x + x^2 + \ldots + x^{n-1})$
- Let $x = \omega^k$, we have $0 = \omega^{kn} - 1 = (\omega^k - 1)(1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)})$
- Therefore, $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0 \Rightarrow$ sums to 0. ▪

Point-Value to Coefficient Representation: Inverse DFT

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

- **Observation.** Almost the same form as DFT.
- **Consequence.** To compute inverse FFT, apply same algorithm but use $\boldsymbol{\omega}^{-1} = e^{-2\pi i/n}$ as principal n^{th} root of unity (and divide by n)

Inverse FFT: Algorithm

```
IFFT(n, a₀,a₁,…,aₙ₋₁) {
    if (n == 1) return a₀

    (e₀,e₁,…,e_{n/2-1}) ← IFFT(n/2, a₀,a₂,a₄,…,aₙ₋₂)
    (d₀,d₁,…,d_{n/2-1}) ← IFFT(n/2, a₁,a₃,a₅,…,aₙ₋₁)

    for k = 0 to n/2 - 1 {
        ωᵏ ← e^{-2πik/n}
        yₖ       ← (eₖ + ωᵏ dₖ)
        y_{k+n/2} ← (eₖ - ωᵏ dₖ)
    }

    return (y₀,y₁,…,yₙ₋₁)
}
```

Note. Need to divide the final result by n

Inverse FFT Summary

- **Theorem.** Inverse FFT algorithm interpolates a degree n-1 polynomial given values at each of the n^th^ roots of unity in O(nlogn) steps

assumes n is a power of 2

$$
\begin{tikzcd}
a_0, a_1, \ldots, a_{n-1} \arrow[r, shift left, "O(n \log n)"] & (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1}) \arrow[l, shift left, "O(n \log n)"]
\end{tikzcd}
$$

coefficient representation

point-value representation

Polynomial Multiplication

- **Theorem.** Can multiply two degree n-1 polynomials in O(nlogn) steps

coefficient representation

$a_0, a_1, \ldots, a_{n-1}$
$b_0, b_1, \ldots, b_{n-1}$

FFT O(n log n)

$A(x_0), \ldots, A(x_{2n-1})$
$B(x_0), \ldots, B(x_{2n-1})$

point-value multiplication O(n)

$C(x_0), C(x_1), \ldots, C(x_{2n-1})$

inverse FFT O(n log n)

coefficient representation

$c_0, c_1, \ldots, c_{2n-2}$

Divide-and-Conquer Summary

- **Basic idea**
  - Break up problem into several parts
  - Solve each part recursively
  - Combine solutions to sub-problems into overall solution
- **Algorithms**
  - Mergesort
    - Divide a sequence into two of same size
  - Closest Pair of Points
    - Vertically divide the space
  - Integer Multiplication
    - Divide each n-digit integer into two ½n-digit integers
  - Matrix Multiplication
    - Divide each n-by-n matrix into four ½n-by-½n blocks
  - Fast Fourier Transform
    - Divide a polynomial into two with even and odd powers