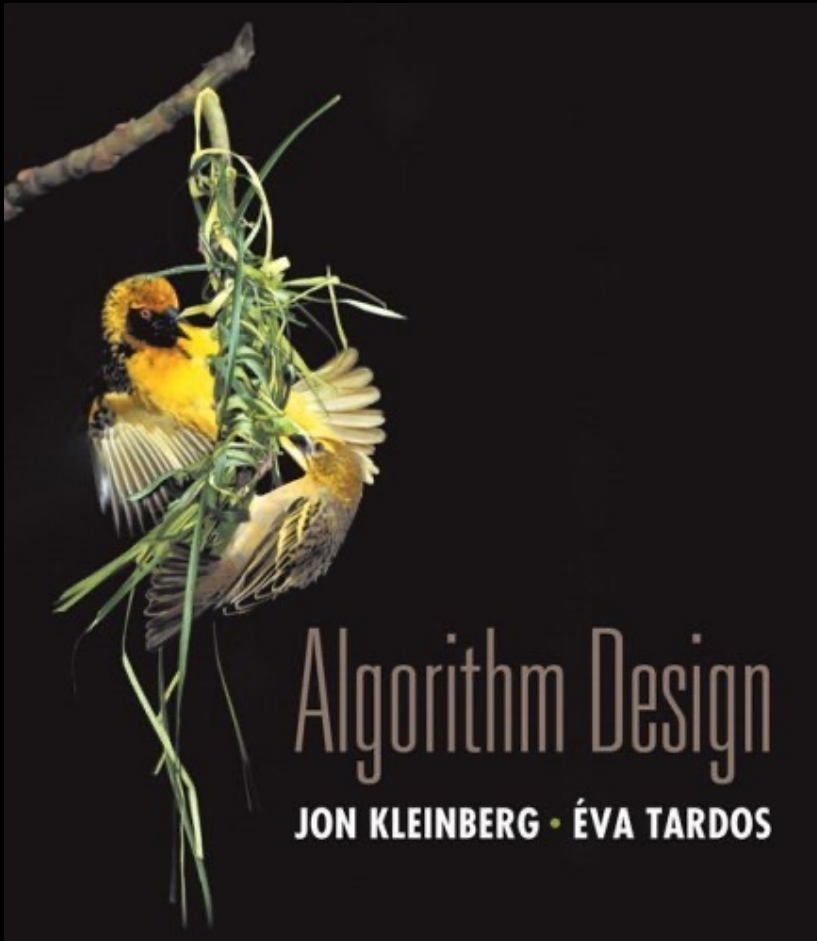


# How to Multiply

integers, matrices, and polynomials



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

Annotated and slightly changed by Yang Xu (徐焱)  
Contact: xuyang@sustech.edu.cn  
Not for commercial use.

# Complex Multiplication

Complex multiplication.  $(a + bi)(c + di) = x + yi$ .

Grade-school.  $x = ac - bd$ ,  $y = bc + ad$ .



4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

# Complex Multiplication

Complex multiplication.  $(a + bi)(c + di) = x + yi$ .

Grade-school.  $x = ac - bd$ ,  $y = bc + ad$ .



4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

A. Yes. [Gauss]  $x = ac - bd$ ,  $y = (a + b)(c + d) - ac - bd$ .



3 multiplications, 5 additions

Remark. Improvement if no hardware multiply.

## 5.5 Integer Multiplication

---

# Integer Addition

**Addition.** Given two  $n$ -bit integers  $a$  and  $b$ , compute  $a + b$ .

**Grade-school.**  $\Theta(n)$  bit operations.

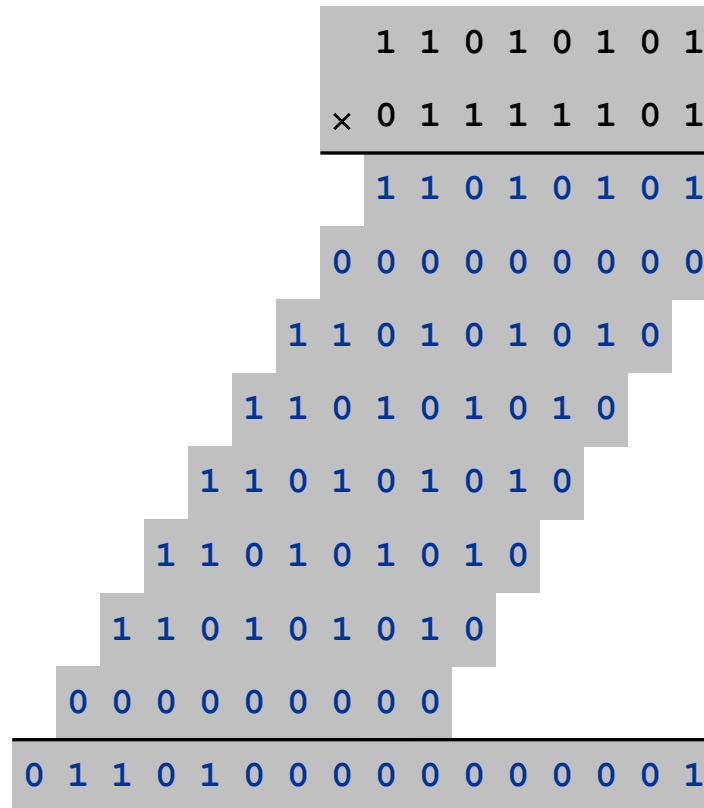
	1	1	1	1	1	1	0	1	
		1	1	0	1	0	1	0	1
+		0	1	1	1	1	1	0	1
<hr/>									
	1	0	1	0	1	0	0	1	0

**Remark.** Grade-school addition algorithm is optimal.

# Integer Multiplication

**Multiplication.** Given two  $n$ -bit integers  $a$  and  $b$ , compute  $a \times b$ .

Grade-school.  $\Theta(n^2)$  bit operations.



Q. Is grade-school multiplication algorithm optimal?

# Divide-and-Conquer Multiplication: Warmup

To multiply two  $n$ -bit integers  $a$  and  $b$ :

- Multiply four  $\frac{1}{2}n$ -bit integers, recursively.
- Add and shift to obtain result.

$$a = 2^{n/2} \cdot a_1 + a_0$$

$$b = 2^{n/2} \cdot b_1 + b_0$$

$$ab = (2^{n/2} \cdot a_1 + a_0)(2^{n/2} \cdot b_1 + b_0) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0$$

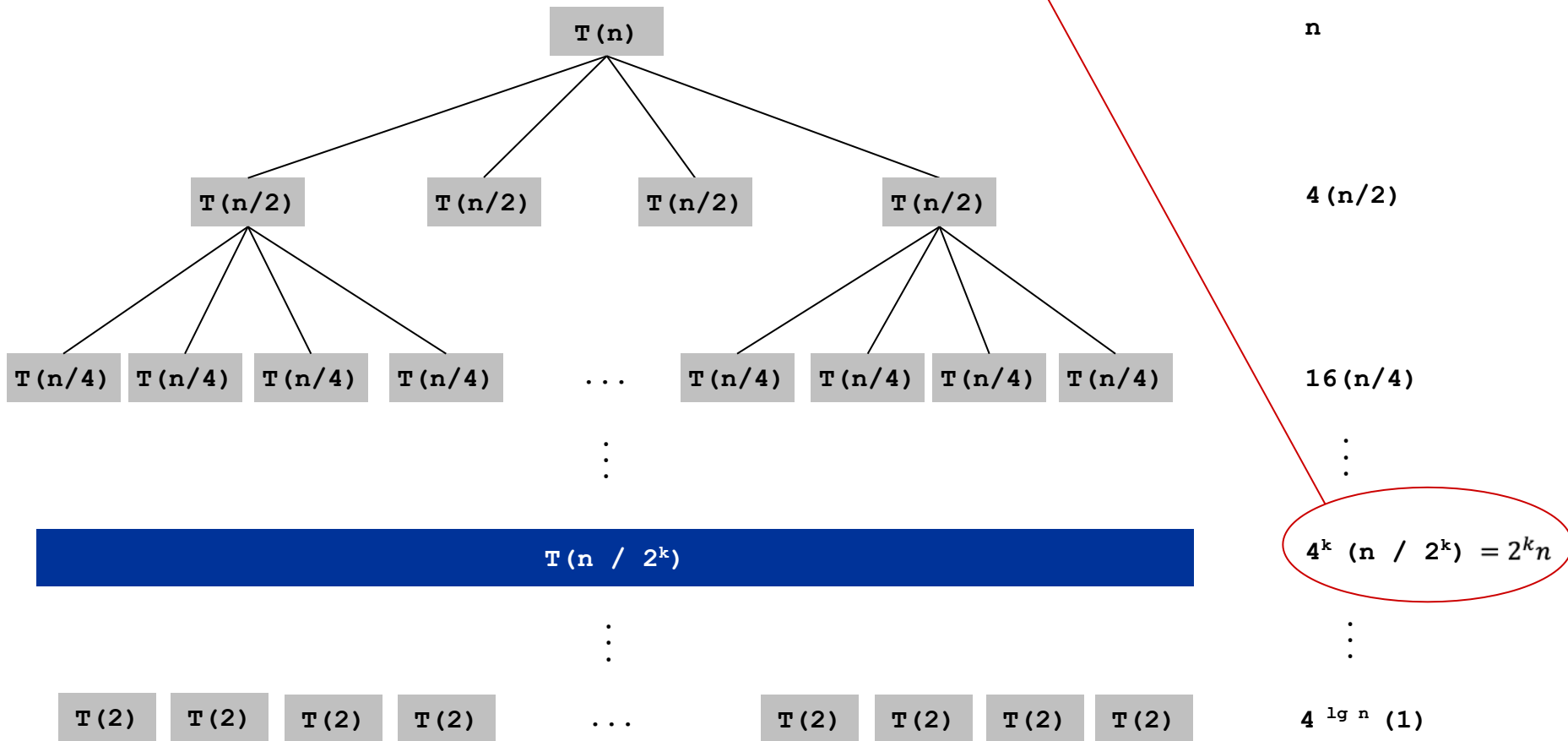
Ex.  $a = \underbrace{10001}_{a_1} \underbrace{1101}_{a_0} \quad b = \underbrace{1110}_{b_1} \underbrace{0001}_{b_0}$

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

# Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

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





# Karatsuba Multiplication

To multiply two  $n$ -bit integers  $a$  and  $b$ :

- Add two  $\frac{1}{2}n$  bit integers.
- Multiply **three**  $\frac{1}{2}n$ -bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$\begin{aligned} a &= 2^{n/2} \cdot a_1 + a_0 \\ b &= 2^{n/2} \cdot b_1 + b_0 \\ ab &= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot (a_1 b_0 + a_0 b_1) + a_0 b_0 \\ &= \underbrace{2^n \cdot a_1 b_1}_{\text{①}} + \underbrace{2^{n/2} \cdot ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0)}_{\text{②}} + \underbrace{a_0 b_0}_{\text{③}} \end{aligned}$$

# Karatsuba Multiplication

To multiply two  $n$ -bit integers  $a$  and  $b$ :

- Add two  $\frac{1}{2}n$  bit integers.
- Multiply **three**  $\frac{1}{2}n$ -bit integers, recursively.
- Add, subtract, and shift to obtain result.

Key step:

Replace  $a_1b_0 + a_0b_1$   
with  $(a_1 + a_0)(b_1 + b_0)$

$$\begin{aligned}
 a &= 2^{n/2} \cdot a_1 + a_0 \\
 b &= 2^{n/2} \cdot b_1 + b_0 \\
 ab &= 2^n \cdot a_1b_1 + 2^{n/2} \cdot (a_1b_0 + a_0b_1) + a_0b_0 \\
 &= 2^n \cdot a_1b_1 + 2^{n/2} \cdot ((a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0) + a_0b_0
 \end{aligned}$$

①
②
①
③
③

**Theorem.** [Karatsuba-Ofman 1962] Can multiply two  $n$ -bit integers in  $O(n^{1.585})$  bit operations.

$$T(n) \leq \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})$$

# Karatsuba Multiplication

Similar to complex multiplication problem

**Grade-school.**  $x = ac - bd$ ,  $y = bc + ad$ .

↖ 4 multiplications, 2 additions

**Q.** Is it possible to do with fewer multiplications?

**A. Yes. [Gauss]**  $x = ac - bd$ ,  $y = (a + b)(c + d) - ac - bd$ .

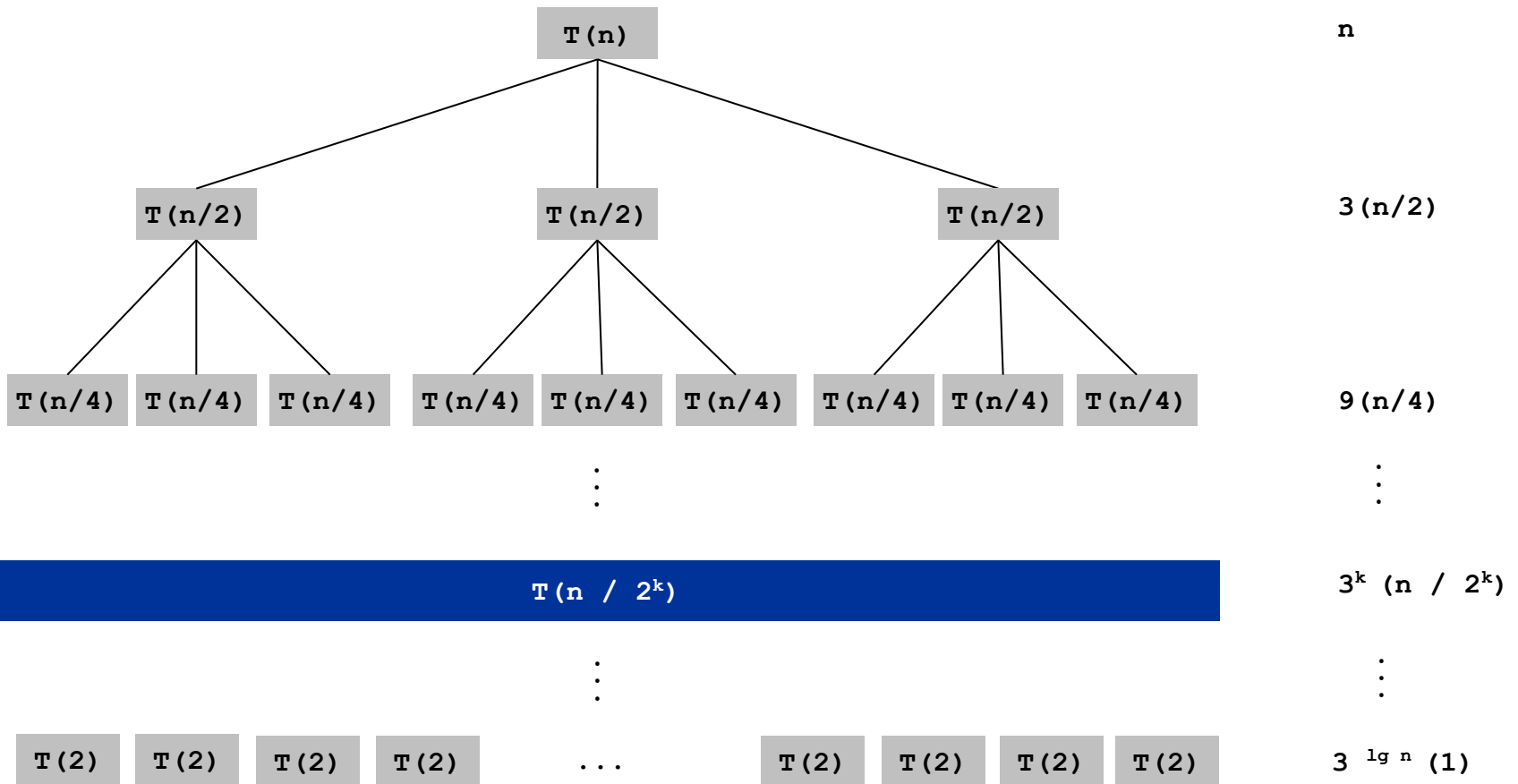
↖ 3 multiplications, 5 additions

## Karatsuba: Recursion Tree

$$O(n^{\lg 3}) = O(n^{1.585})$$

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n \left(\frac{3}{2}\right)^k = n \left( \frac{\left(\frac{3}{2}\right)^{1+\lg n} - 1}{\frac{3}{2} - 1} \right) = 3n^{\lceil \lg 3 \rceil} - 2n$$



# Fast Integer Division Too (!)

**Integer division.** Given two  $n$ -bit (or less) integers  $s$  and  $t$ , compute quotient  $q = s / t$  and remainder  $r = s \bmod t$ .

**Fact.** Complexity of integer division is same as integer multiplication.

To compute quotient  $q$ :

- Approximate  $x = 1 / t$  using Newton's method:
- After  $\log n$  iterations, either  $q = \lfloor s x \rfloor$  or  $q = \lceil s x \rceil$ .

$$x_{i+1} = 2x_i - tx_i^2$$

↖  
using fast  
multiplication


# Matrix Multiplication

---

# Dot Product

**Dot product.** Given two length  $n$  vectors  $a$  and  $b$ , compute  $c = a \cdot b$ .

**Grade-school.**  $\Theta(n)$  arithmetic operations.

$$a \cdot b = \sum_{i=1}^n a_i b_i$$


$$a = \begin{bmatrix} .70 & .20 & .10 \end{bmatrix}$$

$$b = \begin{bmatrix} .30 & .40 & .30 \end{bmatrix}$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

**Remark.** Grade-school dot product algorithm is optimal.

# Matrix Multiplication

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

**Grade-school.**  $\Theta(n^3)$  arithmetic operations.

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

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

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

**Q.** Is grade-school matrix multiplication algorithm optimal?



# Block Matrix Multiplication

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

Labels with arrows pointing to the corresponding blocks:

- $C_{11}$  points to the first two columns of the result matrix.
- $A_{11}$  points to the top-left 2x2 block of matrix A.
- $A_{12}$  points to the top-right 2x2 block of matrix A.
- $B_{11}$  points to the top-left 2x2 block of matrix B.
- $B_{21}$  points to the bottom-left 2x2 block of matrix B.

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

# Matrix Multiplication: Warmup

To multiply two  $n$ -by- $n$  matrices  $A$  and  $B$ :

- Divide: partition  $A$  and  $B$  into  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks.
- Conquer: multiply 8 pairs of  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

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

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

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

# Fast Matrix Multiplication

**Key idea.** multiply 2-by-2 blocks with only **7 multiplications**.

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

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$P_1 = A_{11} \times (B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12}) \times B_{22}$$

$$P_3 = (A_{21} + A_{22}) \times B_{11}$$

$$P_4 = A_{22} \times (B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

- 7 multiplications.
- $18 = 8 + 10$  additions and subtractions.

# Fast Matrix Multiplication

To multiply two  $n$ -by- $n$  matrices  $A$  and  $B$ : [Strassen 1969]

- Divide: partition  $A$  and  $B$  into  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks.
- Compute: 14  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

## Analysis.

- Assume  $n$  is a power of 2.
- $T(n) = \#$  arithmetic operations.

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

# Fast Matrix Multiplication: Practice

## Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around  $n = 128$ .

## Common misperception. “Strassen is only a theoretical curiosity.”

- Apple reports 8x speedup on G4 Velocity Engine when  $n \approx 2,500$ .
- Range of instances where it's useful is a subject of controversy.

**Remark.** Can “Strassenize”  $Ax = b$ , determinant, eigenvalues, SVD, ....

# Fast Matrix Multiplication: Theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?

A. Yes! [Strassen 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.807})$$

Q. Multiply two 2-by-2 matrices with 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 21 scalar multiplications?

A. Also impossible.

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

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]

▪ Two 20-by-20 matrices with 4,460 scalar multiplications.

$$O(n^{2.805})$$

▪ Two 48-by-48 matrices with 47,217 scalar multiplications.

$$O(n^{2.7801})$$

▪ A year later.

$$O(n^{2.7799})$$

▪ December, 1979.

$$O(n^{2.521813})$$

▪ January, 1980.

$$O(n^{2.521801})$$

# Fast Matrix Multiplication: Theory

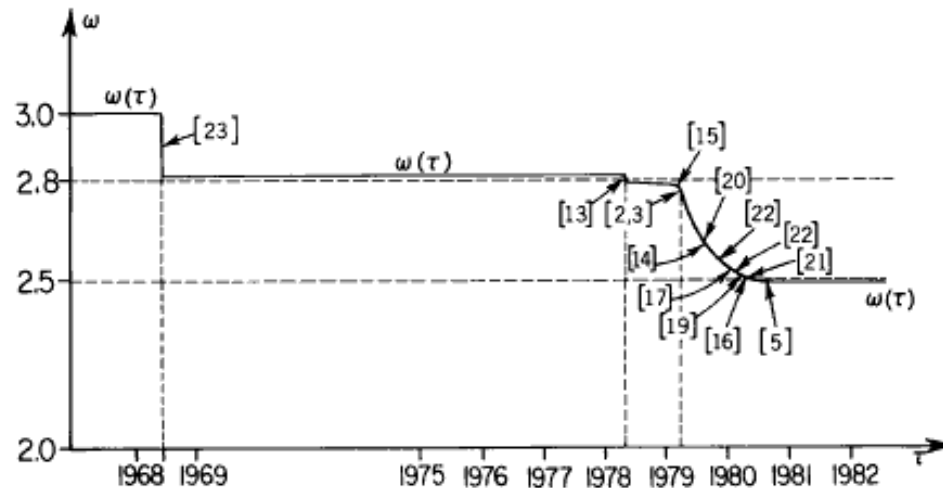


FIG. 1.  $\omega(t)$  is the best exponent announced by time  $\tau$ .

**Best known.**  $O(n^{2.376})$  [Coppersmith-Winograd, 1987]

**Conjecture.**  $O(n^{2+\varepsilon})$  for any  $\varepsilon > 0$ .

**Caveat.** Theoretical improvements to Strassen are progressively less practical.

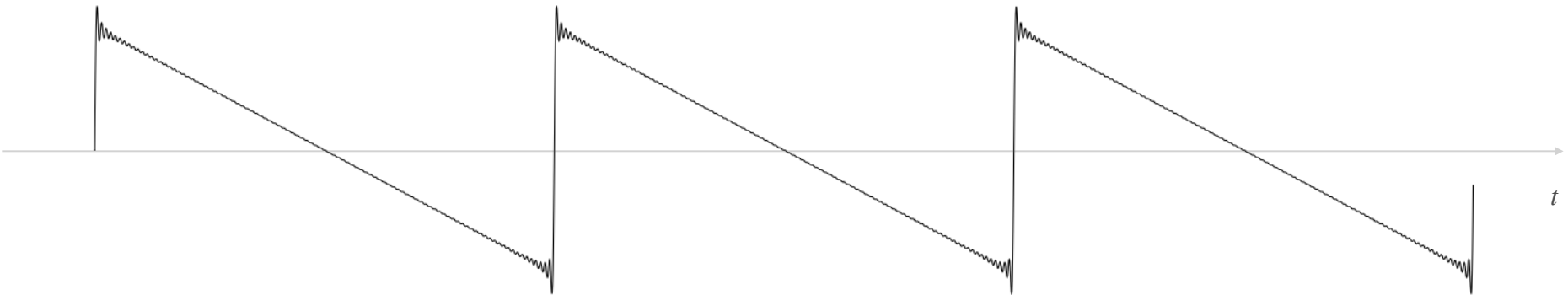
## 5.6 Convolution and FFT

---



# Fourier Analysis

**Fourier theorem.** [Fourier, Dirichlet, Riemann] Any periodic function  
can be expressed as the sum of a series of sinusoids. ↖ sufficiently smooth



$$y(t) = \frac{2}{\pi} \sum_{k=1}^N \frac{\sin kt}{k} \quad N = 100$$

# Euler's Identity

**Sinusoids.** Sum of sine and cosines.

$$e^{ix} = \cos x + i \sin x$$

*Euler's identity*

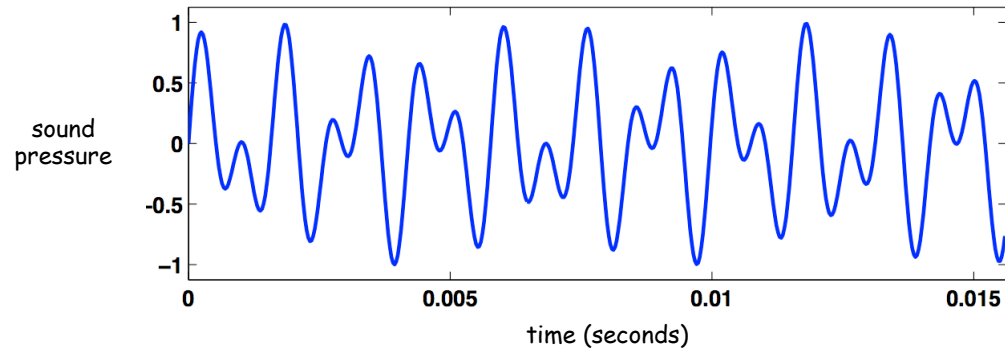
**Sinusoids.** Sum of complex exponentials.

# Time Domain vs. Frequency Domain

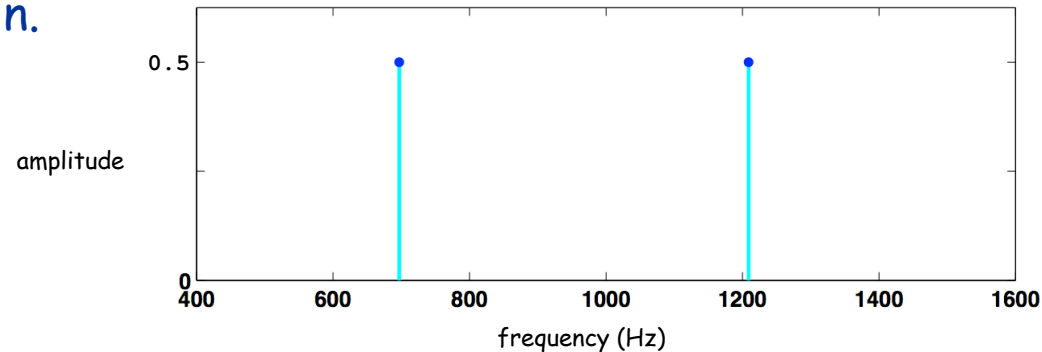
Signal. [touch tone button 1]  $y(t) = \frac{1}{2} \sin(2\pi \cdot 697 t) + \frac{1}{2} \sin(2\pi \cdot 1209 t)$



Time domain.

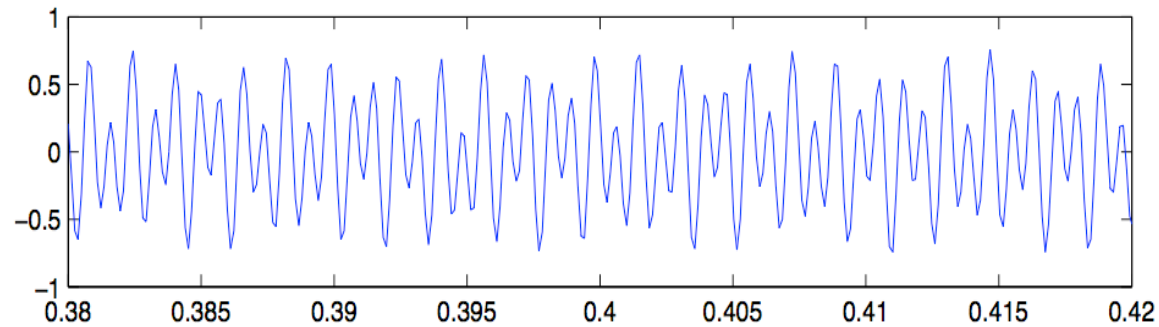


Frequency domain.

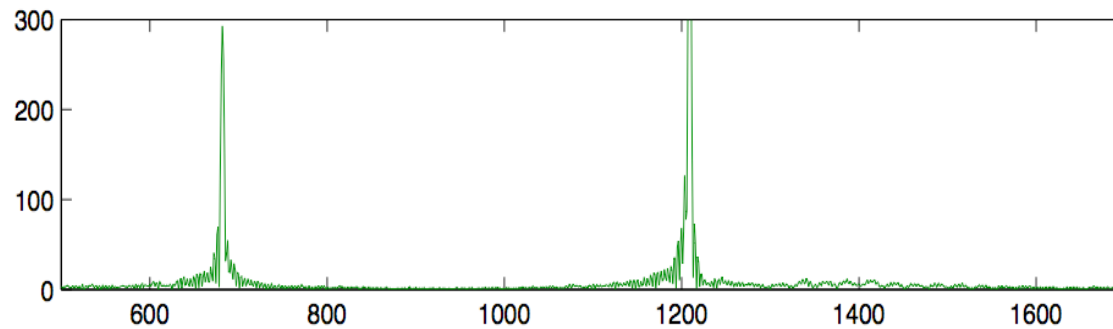


# Time Domain vs. Frequency Domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



# Fast Fourier Transform

**FFT.** Fast way to convert between time-domain and frequency-domain.

**Alternate viewpoint.** Fast way to multiply and evaluate **polynomials**.

↖  
we take this approach

If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it. -Numerical Recipes

# Fast Fourier Transform: Applications

## Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Shor's quantum factoring algorithm.
- ...

The FFT is one of the truly great computational developments of [the 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

# Fast Fourier Transform: Brief History

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.

Runge-König (1924). Laid theoretical groundwork.

Danielson-Lanczos (1942). Efficient algorithm, x-ray crystallography.

Cooley-Tukey (1965). Monitoring nuclear tests in Soviet Union and tracking submarines. Rediscovered and popularized FFT.

Importance not fully realized until advent of digital computers.

# Polynomials: Coefficient Representation

**Polynomial.** [coefficient representation]

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

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

**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.

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



# A Modest PhD Dissertation Title

"New Proof of the Theorem That Every Algebraic Rational Integral Function In One Variable can be Resolved into Real Factors of the First or the Second Degree."

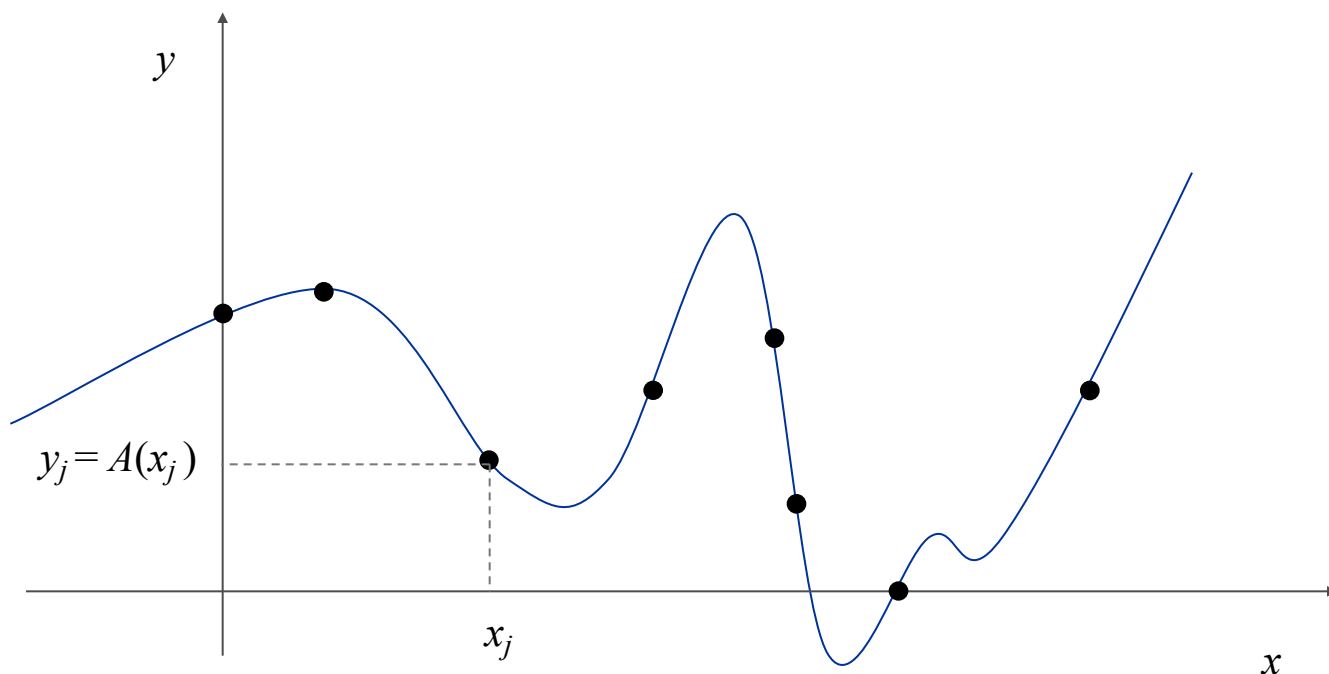
- PhD dissertation, 1799 the University of Helmstedt



# Polynomials: Point-Value Representation

**Fundamental theorem of algebra.** [Gauss, PhD thesis] A degree  $n$  polynomial with complex coefficients has exactly  $n$  complex roots.

**Corollary.** 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]

$$A(x): (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

$$B(x): (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

**Add.**  $O(n)$  arithmetic operations.

$$A(x) + B(x): (x_0, y_0 + z_0), \dots, (x_{n-1}, y_{n-1} + z_{n-1})$$

**Multiply (convolve).**  $O(n)$ , but need  $2n-1$  points.

$$A(x) \times B(x): (x_0, y_0 \times z_0), \dots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

**Evaluate.**  $O(n^2)$  using Lagrange's formula.

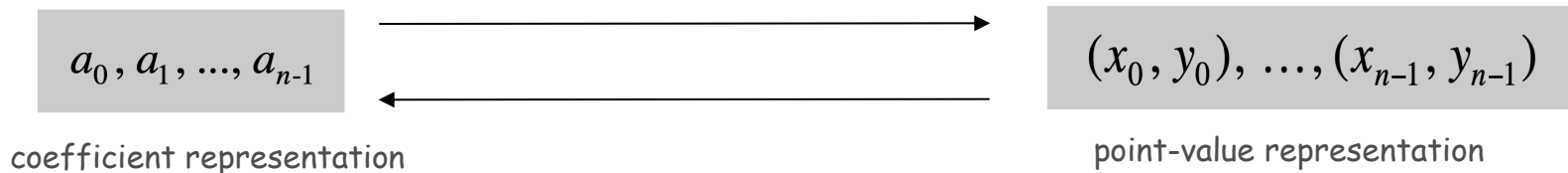
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

# 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.** Efficient conversion between two representations  $\Rightarrow$  all ops fast.



# Converting Between Two Representations: Brute Force

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

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

**Running time.**  $O(n^2)$  for matrix-vector multiply (or  $n$  Horner's).

# Converting Between Two Representations: Brute Force

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

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

↖  
Vandermonde matrix is invertible iff  $x_i$  distinct

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

↖  
or  $O(n^{2.376})$  via fast matrix multiplication

# Divide-and-Conquer

**Decimation in frequency.** Break up polynomial into low and high powers.

- $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
- $A_{low}(x) = a_0 + a_1x + a_2x^2 + a_3x^3.$
- $A_{high}(x) = a_4 + a_5x + a_6x^2 + a_7x^3.$
- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

**Decimation in time.** Break polynomial up into even and odd powers.

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

# Coefficient to Point-Value Representation: Intuition

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

 we get to choose which ones!

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

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

**Intuition.** Choose two points to be  $\pm 1$ .

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

Can evaluate polynomial of degree  $\leq n$  at 2 points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at 1 point.



# Coefficient to Point-Value Representation: Intuition

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

 we get to choose which ones!

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

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

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

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

Can evaluate polynomial of degree  $\leq n$  at 4 points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at 2 points.

# Discrete Fourier Transform

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

**Key idea.** Choose  $x_k = \omega^k$  where  $\omega$  is principal  $n^{\text{th}}$  root of unity.

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

$\uparrow$  DFT                       $\uparrow$  Fourier matrix  $F_n$

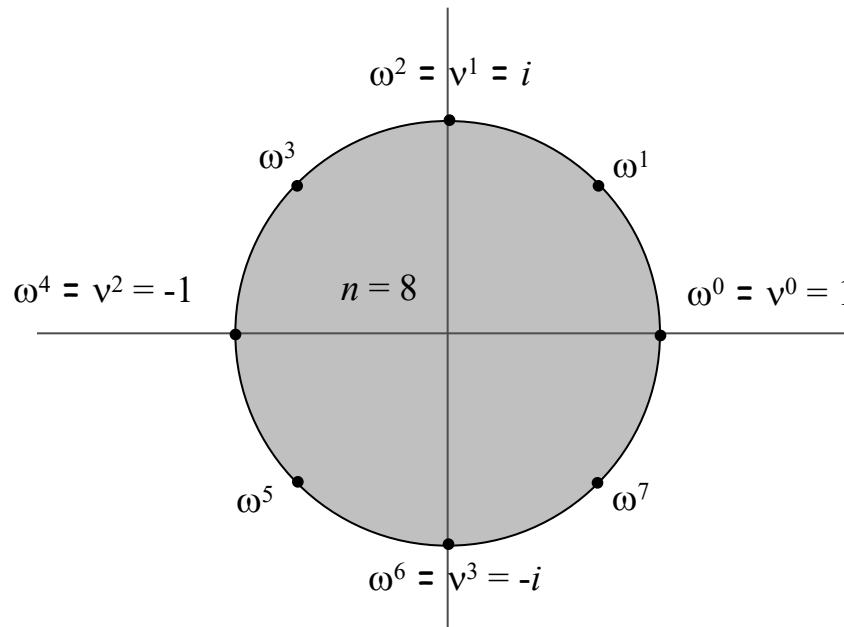
# Roots of Unity

**Def.** An  $n^{\text{th}}$  root of unity is a complex number  $x$  such that  $x^n = 1$ .

**Fact.** The  $n^{\text{th}}$  roots of unity are:  $\omega^0, \omega^1, \dots, \omega^{n-1}$  where  $\omega = e^{2\pi i/n}$ .

**Pf.**  $(\omega^k)^n = (e^{2\pi i k/n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$ .

**Fact.** The  $\frac{1}{2}n^{\text{th}}$  roots of unity are:  $v^0, v^1, \dots, v^{n/2-1}$  where  $v = \omega^2 = e^{4\pi i/n}$ .



# Fast Fourier Transform

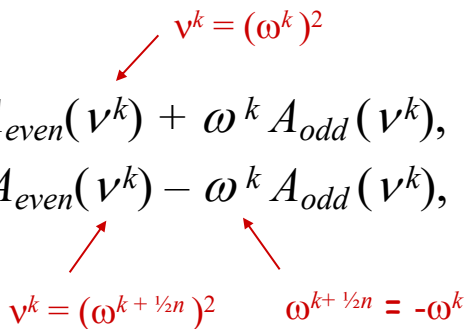
**Goal.** Evaluate a degree  $n-1$  polynomial  $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$  at its  $n^{\text{th}}$  roots of unity:  $\omega^0, \omega^1, \dots, \omega^{n-1}$ .

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

- $A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}.$
- $A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}.$
- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$

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

**Combine.**

- $A(\omega^k) = A_{\text{even}}(\nu^k) + \omega^k A_{\text{odd}}(\nu^k), \quad 0 \leq k < n/2$
  - $A(\omega^{k+\frac{1}{2}n}) = A_{\text{even}}(\nu^k) - \omega^k A_{\text{odd}}(\nu^k), \quad 0 \leq k < n/2$
- 
- $\nu^k = (\omega^k)^2$
- $\nu^k = (\omega^{k+\frac{1}{2}n})^2$
- $\omega^{k+\frac{1}{2}n} = -\omega^k$

# FFT Algorithm

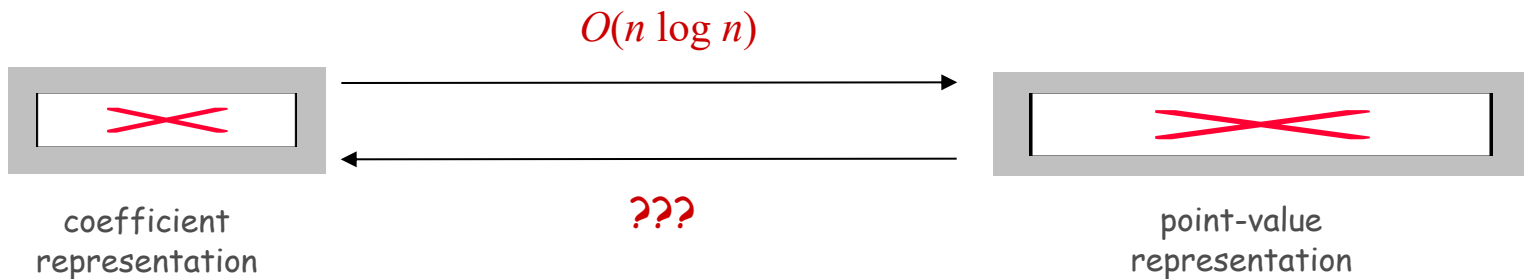
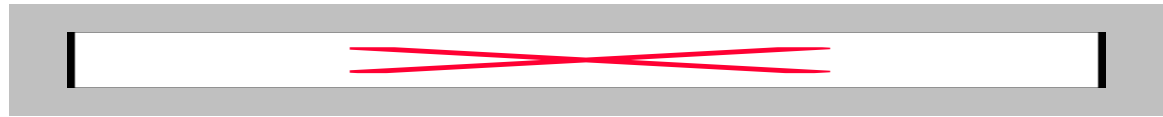
```
fft(n, a0, a1, ..., an-1) {  
    if (n == 1) return a0  
  
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a2, a4, ..., an-2)  
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, a5, ..., an-1)  
  
    for k = 0 to n/2 - 1 {  
        ωk ← e2πik/n  
        yk ← ek + ωk dk  
        yk+n/2 ← ek - ωk dk  
    }  
  
    return (y0, y1, ..., yn-1)  
}
```

# FFT Summary

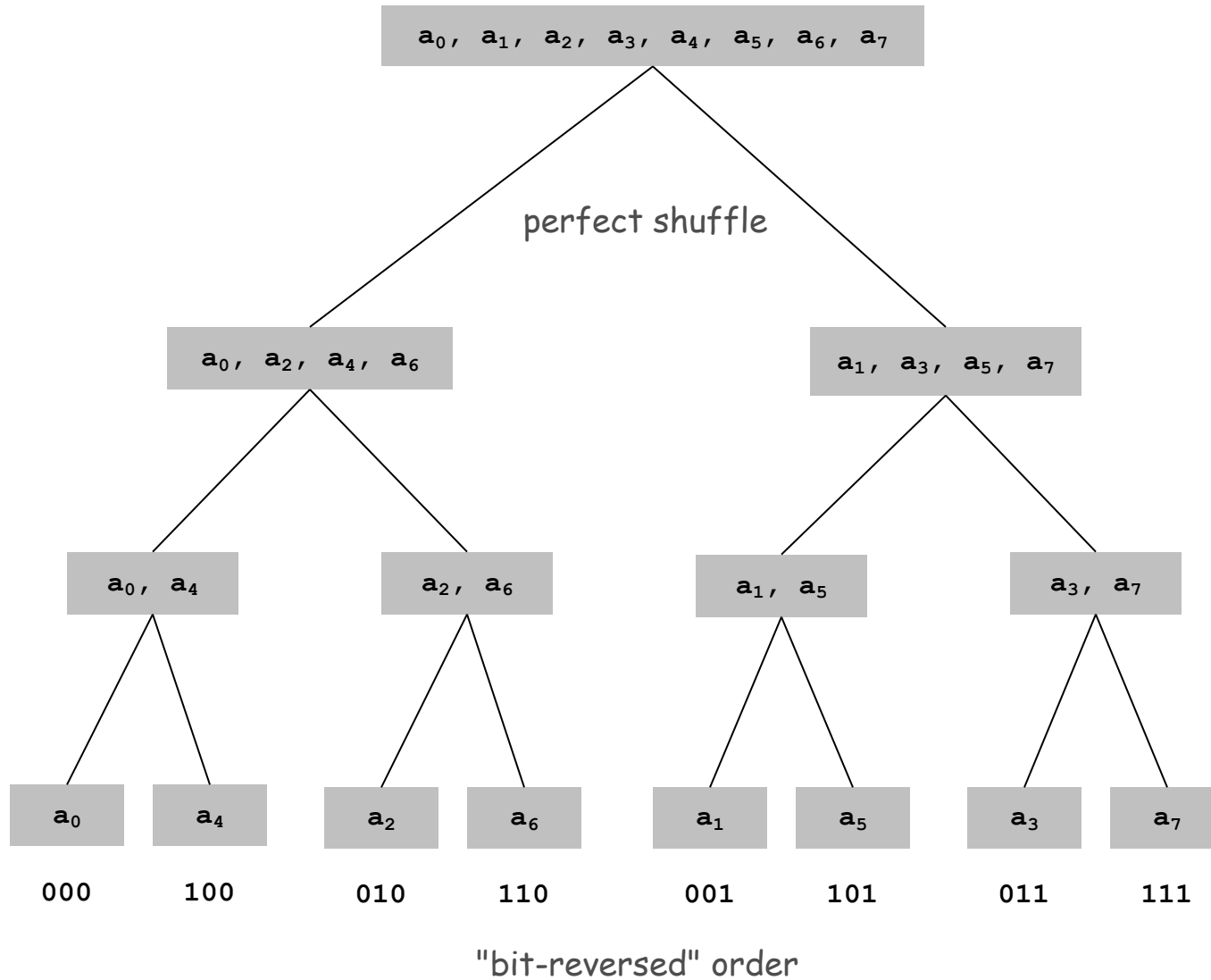
**Theorem.** FFT algorithm evaluates a degree  $n-1$  polynomial at each of the  $n^{\text{th}}$  roots of unity in  $O(n \log n)$  steps.

↙  
assumes  $n$  is a power of 2

Running time.

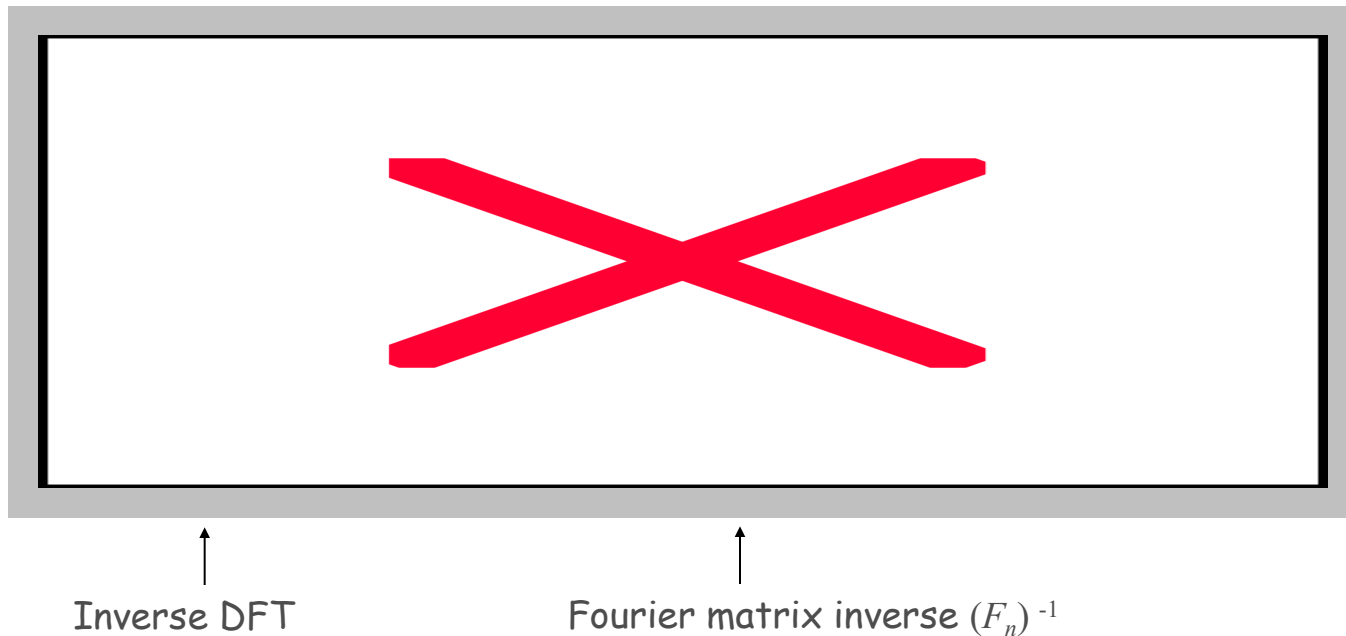


# Recursion Tree



# Inverse Discrete Fourier Transform

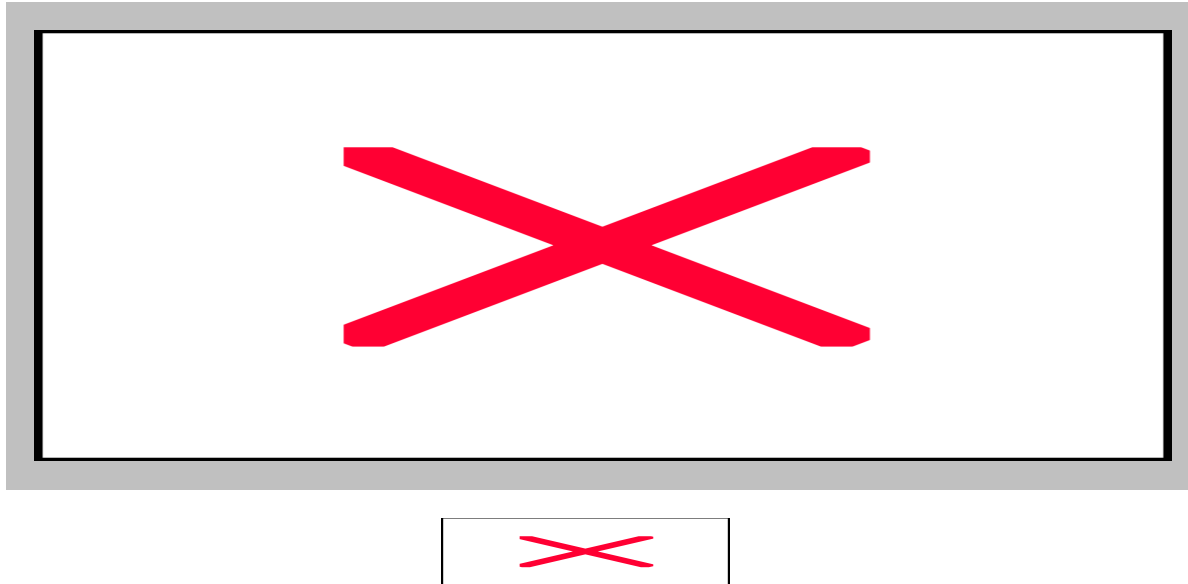
**Point-value  $\Rightarrow$  coefficient.** Given  $n$  distinct points  $x_0, \dots, x_{n-1}$  and values  $y_0, \dots, y_{n-1}$ , find unique polynomial  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , that has given values at given points.





# Inverse DFT

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

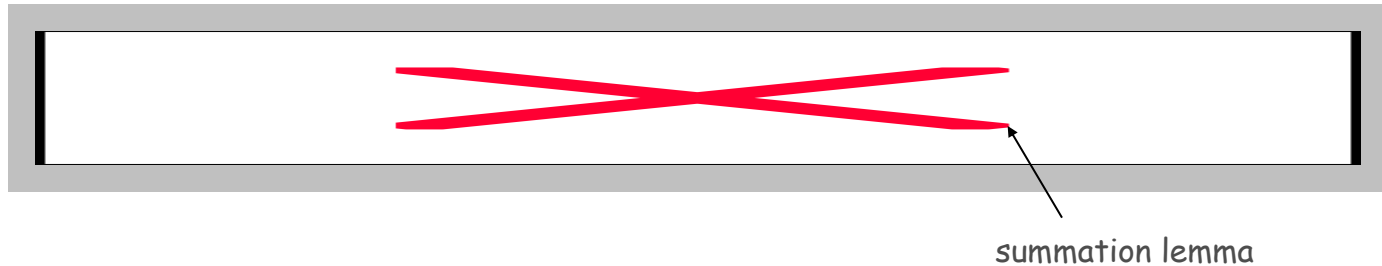


**Consequence.** To compute inverse FFT, apply same algorithm but use  $\omega^{-1} = e^{-2\pi i/n}$  as principal  $n^{\text{th}}$  root of unity (and divide by  $n$ ).

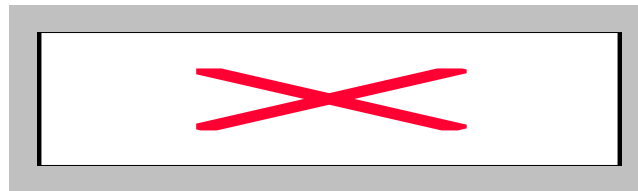
# Inverse FFT: Proof of Correctness

**Claim.**  $F_n$  and  $G_n$  are inverses.

**Pf.**



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



**Pf.**

- If  $k$  is a multiple of  $n$  then  $\omega^k = 1 \Rightarrow$  series sums to  $n$ .
- Each  $n^{\text{th}}$  root of unity  $\omega^k$  is a root of  $x^n - 1 = (x - 1)(1 + x + x^2 + \dots + x^{n-1})$ .
- if  $\omega^k \neq 1$  we have:  $1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)} = 0 \Rightarrow$  series sums to 0. ■

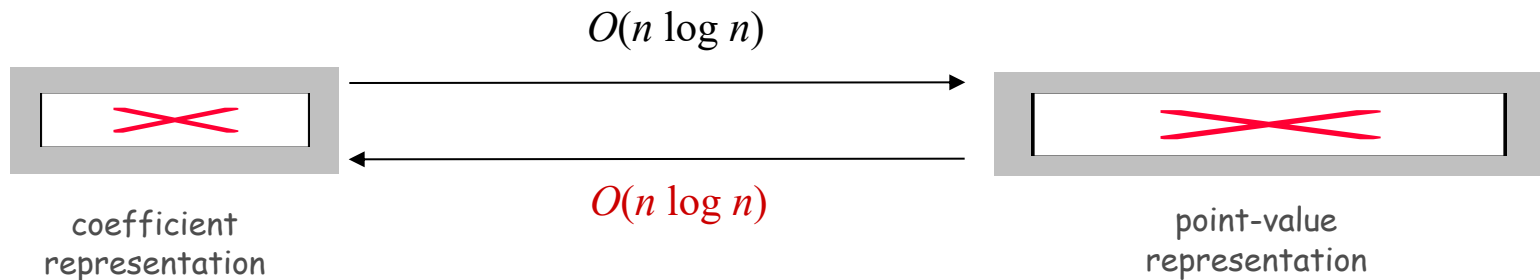
# Inverse FFT: Algorithm

```
ifft(n, a0, a1, ..., an-1) {  
    if (n == 1) return a0  
  
    (e0, e1, ..., en/2-1) ← FFT(n/2, a0, a2, a4, ..., an-2)  
    (d0, d1, ..., dn/2-1) ← FFT(n/2, a1, a3, a5, ..., an-1)  
  
    for k = 0 to n/2 - 1 {  
        ωk ← e-2πik/n  
        yk+n/2 ← (ek + ωk dk) / n  
        yk ← (ek - ωk dk) / n  
    }  
  
    return (y0, y1, ..., yn-1)  
}
```

# Inverse FFT Summary

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

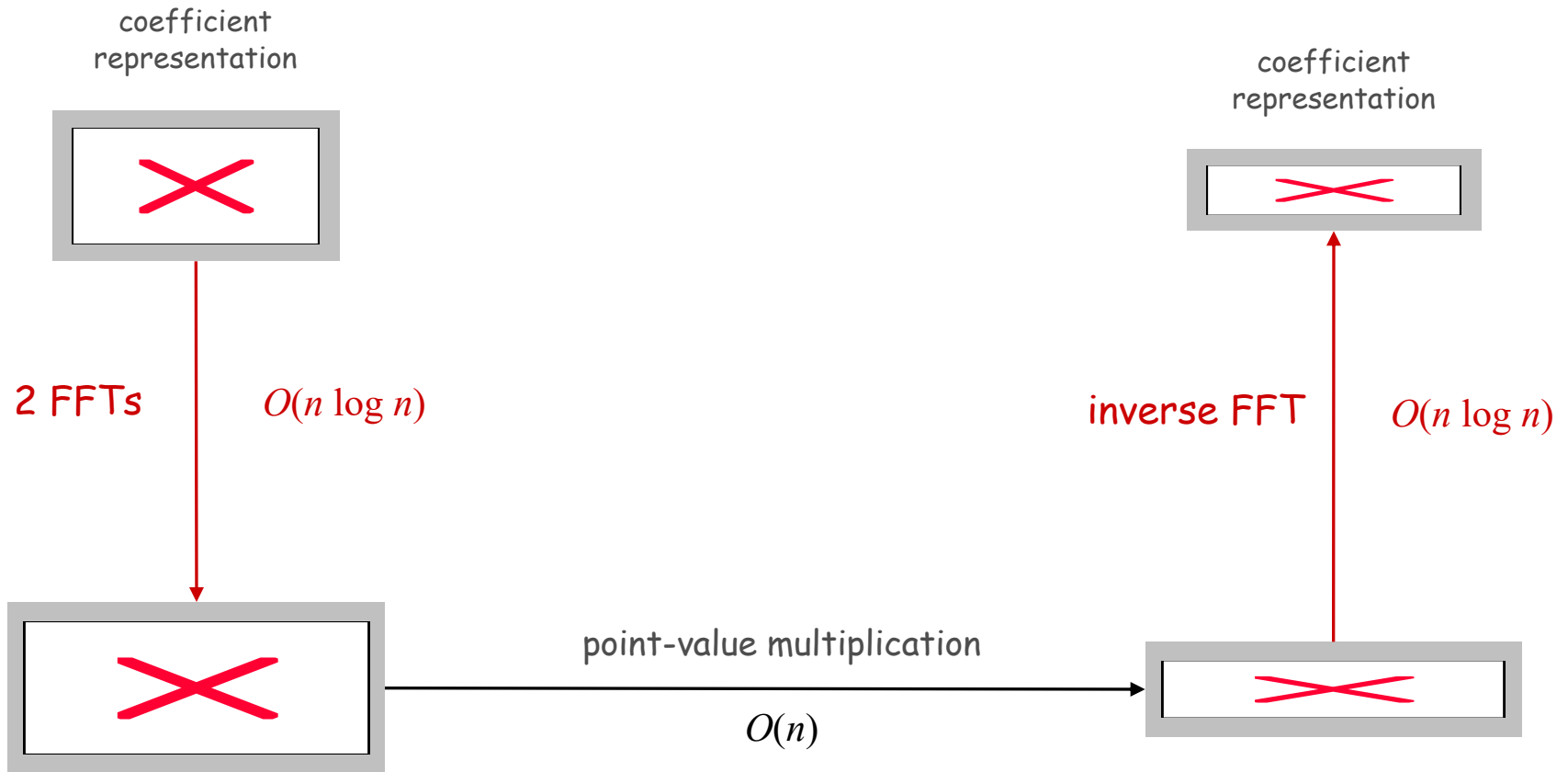
↑  
assumes  $n$  is a power of 2



# Polynomial Multiplication

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

pad with 0s to make  $n$  a power of 2



# FFT in Practice ?

The screenshot shows a Google search results page for the query "fft java". The browser window title is "fft java - Google Search". The address bar shows the URL "http://www.google.com/search?hl=en&q=fft+java&btnG=Google+Search". The search bar contains "fft java". The results are listed under the "Web" tab. The first result is "FFT.java" from Princeton University, with a description: "FFT code in Java. ... Compilation: javac FFT.java \* Execution: java FFT N \* Dependencies: Complex.java \* \* Compute the FFT and inverse FFT of a length N ...". The second result is "YOY408 Programming Resources - Code Spotlight - FFT Java source ..." from yoy408.com, with a description: "Compilation: javac FFT.java \* Execution: java FFT N \* Dependencies: ... A nice implementation of the FFT algorithm in Java, Eventhough it can use too much ...". The third result is "FFT JAVA Demo" from ling.upenn.edu, with a description: "This is a JAVA applet demonstrating basic concept of Fast Fourier ... If you want to run the program locally, download FFT.zip and unzip it to a directory. ...". The fourth result is "Mathtools.net : Java/FFT" from mathtools.net, with a description: "Listing of Java FFT related links, tools, and resources. ...". The fifth result is "FFT Spectrum Analyser Demo" from dsptutor.freeuk.com, with a description: "The following features are new in the Java 1.1 version of the FFT Spectrum Analyser applet. The signal is plotted in either the time domain (signal) or the ...". The sixth result is "Fun with Java. Understanding the Fast Fourier Transform (FFT) ..." from developer.com, with a description: "Fun with Java, Understanding the Fast Fourier Transform (FFT) Algorithm By Richard G. Baldwin. Java Programming, Notes # 1486. Preface; General Discussion ...". The seventh result is "Spectrum Analysis using Java. Sampling Frequency. Folding ..." from developer.com, with a description: "File Dsp030.java Copyright 2004, RGBBaldwin Rev 5/14/04 Uses an FFT algorithm to compute and display the magnitude of the spectral content for up to five ...". The eighth result is "Bruce R. Miller's Java(tm) Demo Page" from math.nist.gov, with a description: "These classes may be of use to other java programmers. Available Packages, Demos & Bug Fixes.: FFT. TabPanel. ObjectList. StackLayout. Scroller. ...". The ninth result is "FFT : Java Glossary" from mindprod.com, with a description: "Roedy Green's Java & Internet Glossary : FFT. ... You are here : home ⇐ Java Glossary ⇐ F words ⇐ FFT. FFT: Fast Fourier Transform. ...".

fft java - Google Search

http://www.google.com/search?hl=en&q=fft+java&btnG=Google+Search

Google

Web Images Groups News Froogle Local Scholar more »

fft java

Search

Advanced Search

Preferences

Web

Results 1 - 10 of about 630,000 for [fft java](#). (0.17 seconds)

**FFT.java**  
FFT code in Java. ... Compilation: javac FFT.java \* Execution: java FFT N \* Dependencies: Complex.java \* \* Compute the FFT and inverse FFT of a length N ...  
[www.cs.princeton.edu/introcs/97data/FFT.java.html](http://www.cs.princeton.edu/introcs/97data/FFT.java.html) - 36k - [Cached](#) - [Similar pages](#)

**YOY408 Programming Resources - Code Spotlight - FFT Java source ...**  
Compilation: javac FFT.java \* Execution: java FFT N \* Dependencies: ... A nice implementation of the FFT algorithm in Java, Eventhough it can use too much ...  
[www.yoy408.com/html/codespot.php?gg=35](http://www.yoy408.com/html/codespot.php?gg=35) - 26k - [Cached](#) - [Similar pages](#)

**FFT JAVA Demo**  
This is a JAVA applet demonstrating basic concept of Fast Fourier ... If you want to run the program locally, download FFT.zip and unzip it to a directory. ...  
[www.ling.upenn.edu/~tklee/Projects/dsp/](http://www.ling.upenn.edu/~tklee/Projects/dsp/) - 8k - [Cached](#) - [Similar pages](#)

**Mathtools.net : Java/FFT**  
Listing of Java FFT related links, tools, and resources.  
[www.mathtools.net/Java/FFT/index.html](http://www.mathtools.net/Java/FFT/index.html) - 18k - [Cached](#) - [Similar pages](#)

**FFT Spectrum Analyser Demo**  
The following features are new in the Java 1.1 version of the FFT Spectrum Analyser applet. The signal is plotted in either the time domain (signal) or the ...  
[www.dsptutor.freeuk.com/analyser/SpectrumAnalyser.html](http://www.dsptutor.freeuk.com/analyser/SpectrumAnalyser.html) - 4k - [Cached](#) - [Similar pages](#)

**Fun with Java. Understanding the Fast Fourier Transform (FFT) ...**  
Fun with Java, Understanding the Fast Fourier Transform (FFT) Algorithm By Richard G. Baldwin. Java Programming, Notes # 1486. Preface; General Discussion ...  
[www.developer.com/java/other/article.php/3457251](http://www.developer.com/java/other/article.php/3457251) - 116k - [Cached](#) - [Similar pages](#)

**Spectrum Analysis using Java. Sampling Frequency. Folding ...**  
File Dsp030.java Copyright 2004, RGBBaldwin Rev 5/14/04 Uses an FFT algorithm to compute and display the magnitude of the spectral content for up to five ...  
[www.developer.com/java/other/article.php/3380031](http://www.developer.com/java/other/article.php/3380031) - 278k - [Cached](#) - [Similar pages](#)

**Bruce R. Miller's Java(tm) Demo Page**  
These classes may be of use to other java programmers. Available Packages, Demos & Bug Fixes.: FFT. TabPanel. ObjectList. StackLayout. Scroller. ...  
[math.nist.gov/~BMiller/java/](http://math.nist.gov/~BMiller/java/) - 7k - [Cached](#) - [Similar pages](#)

**FFT : Java Glossary**  
Roedy Green's Java & Internet Glossary : FFT. ... You are here : home ⇐ Java Glossary ⇐ F words ⇐ FFT. FFT: Fast Fourier Transform. ...  
[mindprod.com/jgloss/fft.html](http://mindprod.com/jgloss/fft.html) - 8k - [Cached](#) - [Similar pages](#)

# FFT in Practice

## Fastest Fourier transform in the West. [Frigo and Johnson]

- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor-tuned code.

## Implementation details.

- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Core algorithm is nonrecursive version of Cooley-Tukey.
- $O(n \log n)$ , even for prime sizes.

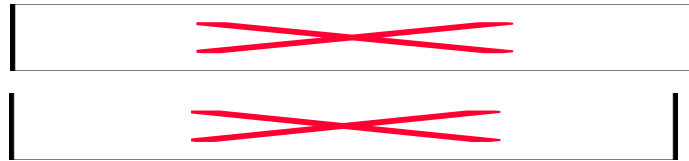
Reference: <http://www.fftw.org>

# Integer Multiplication, Redux

**Integer multiplication.** Given two  $n$  bit integers  $a = a_{n-1} \dots a_1 a_0$  and  $b = b_{n-1} \dots b_1 b_0$ , compute their product  $a \cdot b$ .

**Convolution algorithm.**

- Form two polynomials.
- Note:  $a = A(2)$ ,  $b = B(2)$ .
- Compute  $C(x) = A(x) \cdot B(x)$ .
- Evaluate  $C(2) = a \cdot b$ .
- Running time:  $O(n \log n)$  complex **arithmetic** operations.



**Theory.** [Schönhage-Strassen 1971]  $O(n \log n \log \log n)$  **bit** operations.

**Theory.** [Fürer 2007]  $O(n \log n 2^{O(\log^* n)})$  **bit operations**.



# Integer Multiplication, Redux

**Integer multiplication.** Given two  $n$  bit integers  $a = a_{n-1} \dots a_1 a_0$  and  $b = b_{n-1} \dots b_1 b_0$ , compute their product  $a \cdot b$ .

"the fastest bignum library on the planet"



**Practice.** [GNU Multiple Precision Arithmetic Library]

It uses brute force, Karatsuba, and FFT, depending on the size of  $n$ .

# Integer Arithmetic

Fundamental open question. What is complexity of arithmetic?

Operation	Upper Bound	Lower Bound
addition	$O(n)$	$\Omega(n)$
multiplication	$O(n \log n 2^{O(\log^* n)})$	$\Omega(n)$
division	$O(n \log n 2^{O(\log^* n)})$	$\Omega(n)$

# Factoring

**Factoring.** Given an  $n$ -bit integer, find its prime factorization.

$$2773 = 47 \times 59$$

$$2^{67}-1 = 147573952589676412927 = 193707721 \times 761838257287$$

a disproof of Mersenne's conjecture that  $2^{67} - 1$  is prime

740375634795617128280467960974295731425931888892312890849  
362326389727650340282662768919964196251178439958943305021  
275853701189680982867331732731089309005525051168770632990  
72396380786710086096962537934650563796359

RSA-704  
(\$30,000 prize if you can factor)

# Factoring and RSA

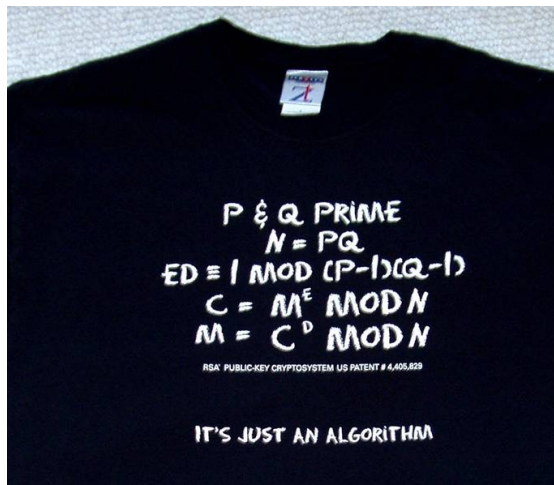
**Primality.** Given an  $n$ -bit integer, is it prime?

**Factoring.** Given an  $n$ -bit integer, find its prime factorization.

**Significance.** Efficient primality testing  $\Rightarrow$  can implement RSA.

**Significance.** Efficient factoring  $\Rightarrow$  can break RSA.

**Theorem.** [AKS 2002] Poly-time algorithm for primality testing.



# Shor's Algorithm

Shor's algorithm. Can factor an  $n$ -bit integer in  $O(n^3)$  time on a quantum computer.

algorithm uses quantum QFT !

Ramification. At least one of the following is wrong:

- RSA is secure.
- Textbook quantum mechanics.
- Extending Church-Turing thesis.



# Shor's Factoring Algorithm

Period finding.

$2^i$	1	2	4	8	16	32	64	128	...
$2^i \bmod 15$	1	2	4	8	1	2	4	8	...
$2^i \bmod 21$	1	2	4	8	16	11	1	2	...

period = 4

period = 6

**Theorem.** [Euler] Let  $p$  and  $q$  be prime, and let  $N = p q$ . Then, the following sequence repeats with a period divisible by  $(p-1)(q-1)$ :

$$x \bmod N, x^2 \bmod N, x^3 \bmod N, x^4 \bmod N, \dots$$

**Consequence.** If we can learn something about the **period** of the sequence, we can learn something about the divisors of  $(p-1)(q-1)$ .

by using random values of  $x$ , we get the divisors of  $(p-1)(q-1)$ , and from this, can get the divisors of  $N = p q$

# Extra Slides

---

# Fourier Matrix Decomposition

