



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

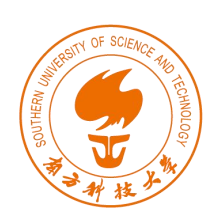
# Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

<http://faculty.sustech.edu.cn/yusq/>



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Divide and Conquer



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# 4. 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	1	0	1
<hr/>									
	1	0	1	0	1	0	0	1	0

- Remark. Grade-school addition algorithm is optimal.



# Integer Multiplication

- Addition. Given two  $n$ -bit integers  $a$  and  $b$ , compute  $a + b$ .
- Grade-school.  $\Theta(n)$  bit operations.

The diagram illustrates the grade-school multiplication algorithm for two 8-bit integers. The first number is 11010101 and the second is 01111101. The partial products are shown as blue boxes, each shifted to the left by its position. The final sum is shown at the bottom, with a horizontal line above it.

```
      1 1 0 1 0 1 0 1
    × 0 1 1 1 1 1 0 1
    -----
          1 1 0 1 0 1 0 1
         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
    0 0 0 0 0 0 0 0 0
   -----
  0 1 1 0 1 0 0 0 0 0 0 0 0 0 1
```

- 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.

$$\begin{aligned} xy &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1y_1 \cdot 2^n + (x_1y_0 + x_0y_1) \cdot 2^{n/2} + x_0y_0 \end{aligned}$$

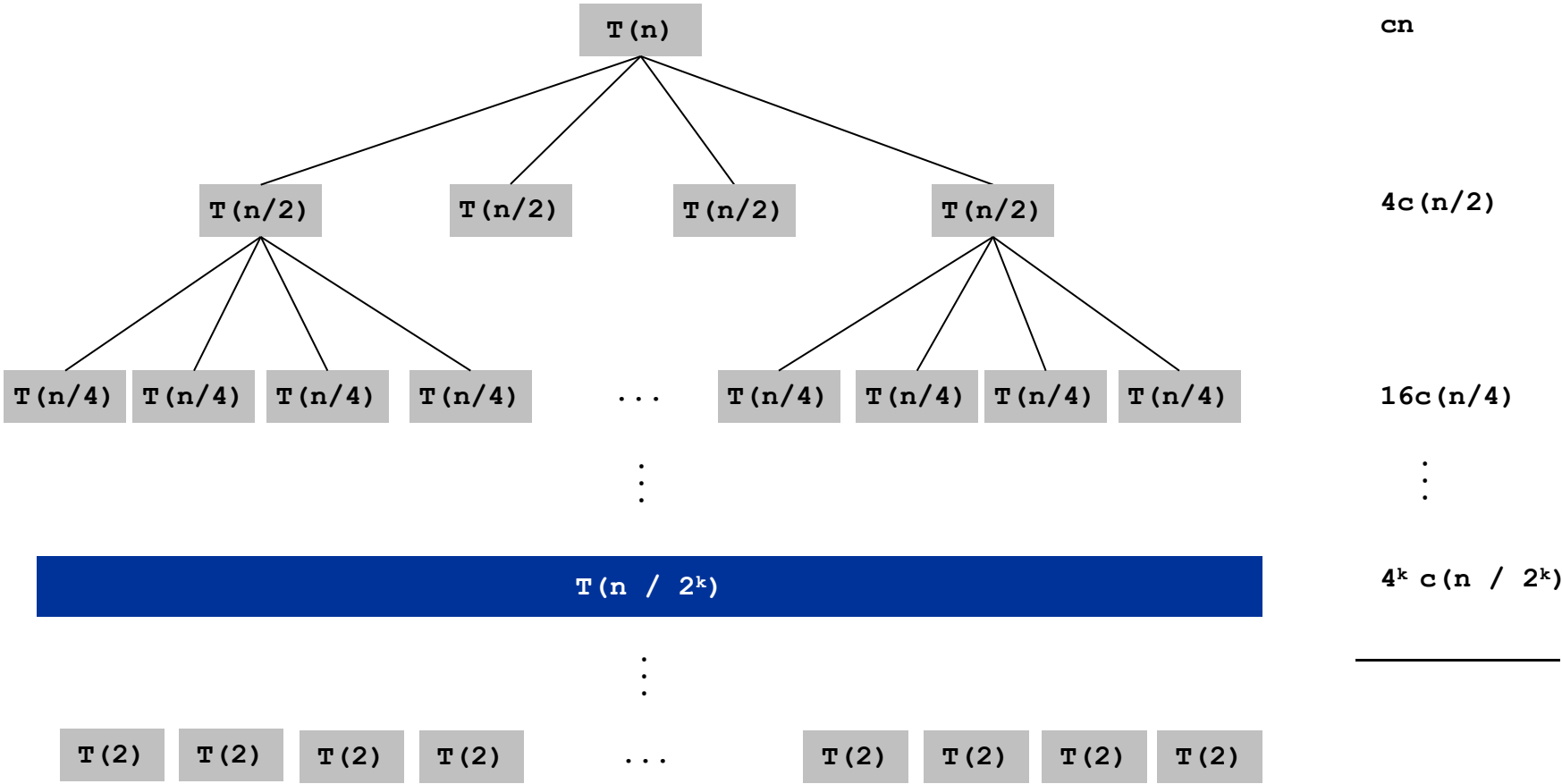
Ex.  $a = \underbrace{1000}_{a_1}\underbrace{1101}_{a_0} \quad b = \underbrace{11100000}_{b_1}\underbrace{01}_{b_0}$



# Recursion Tree

$$T(n) \leq 4T\left(\frac{n}{2}\right) + cn$$

$$T(n) \leq O(n^2)$$





# 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.

`Recursive-Multiply(x,y):`

`Write  $x = x_1 \cdot 2^{n/2} + x_0$`

`$y = y_1 \cdot 2^{n/2} + y_0$`

`Compute  $x_1 + x_0$  and  $y_1 + y_0$`

`$p = \text{Recursive-Multiply}(x_1 + x_0, y_1 + y_0)$`

`$x_1y_1 = \text{Recursive-Multiply}(x_1, y_1)$`

`$x_0y_0 = \text{Recursive-Multiply}(x_0, y_0)$`

`Return  $x_1y_1 \cdot 2^n + (p - x_1y_1 - x_0y_0) \cdot 2^{n/2} + x_0y_0$`

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

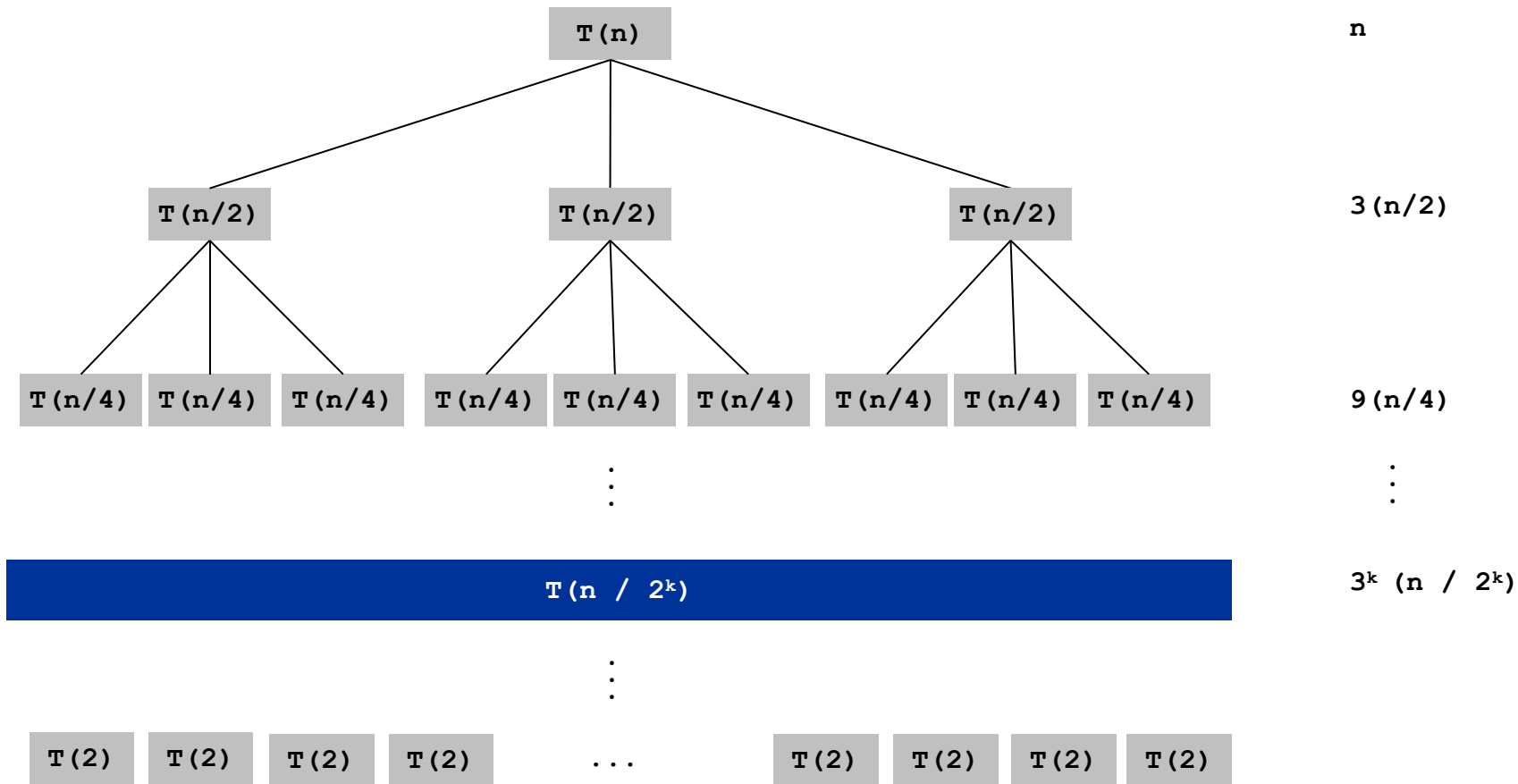


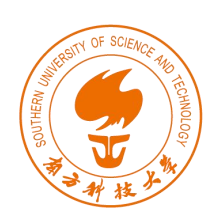


# Karatsuba: Recursion Tree

$$T(n) \leq 3T\left(\frac{n}{2}\right) + cn$$

$$T(n) \leq O(n^{1.585})$$





南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# 5. 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.

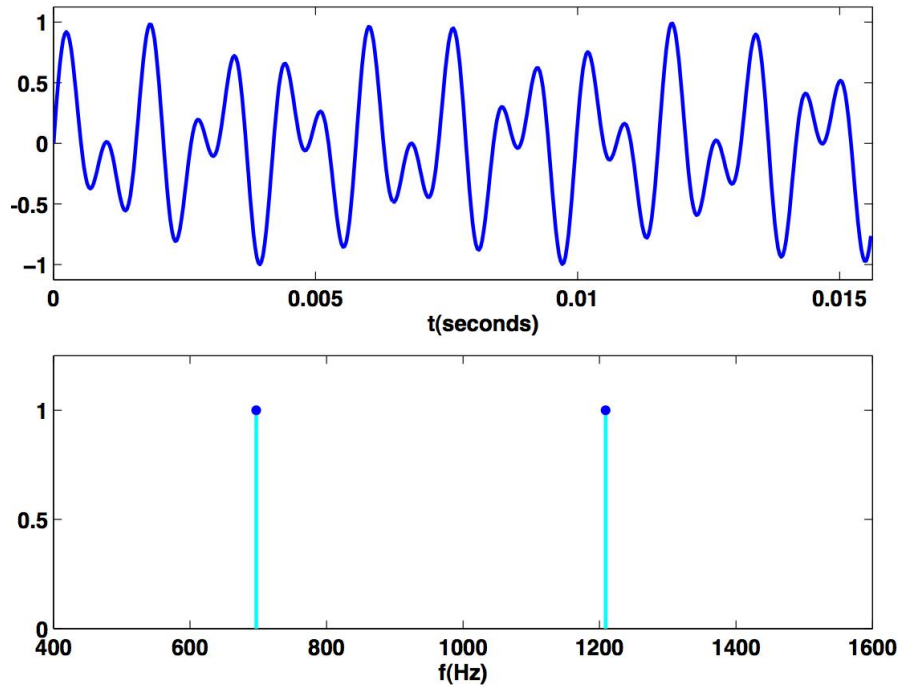
The FFT is one of the truly great computational developments of this [20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. -Charles van Loan



# Touch Tone

$$y_j = \sum_{k=0}^{n-1} a_k e^{-jk \frac{2\pi}{n} i}$$

- Button 1 signal.

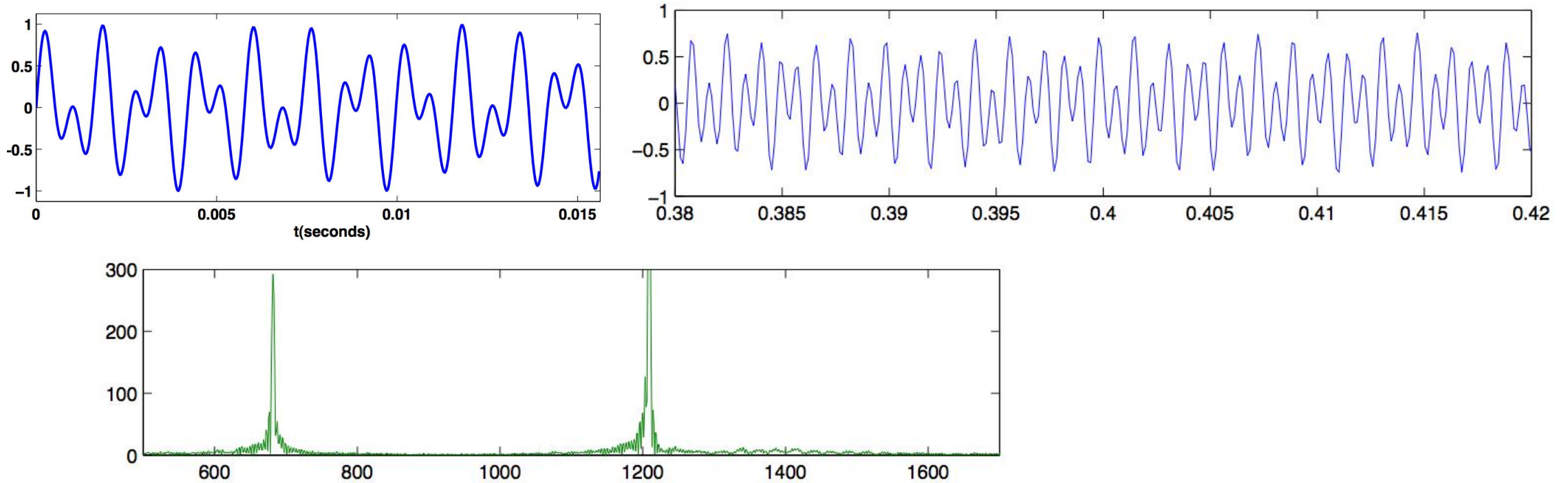


- Magnitude of Fourier transform of button 1 signal.



# Touch Tone

- Button 1 signal. [recorded, 8192 samples per second]



- Magnitude of FFT.



# 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.
- 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.



# Convolution

- What's convolution?
- $O(n^2)$ ?

$(a_0, a_1, a_2, \dots, a_{n-1})$

$(b_0, b_1, b_2, \dots, b_{n-1})$



# 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.

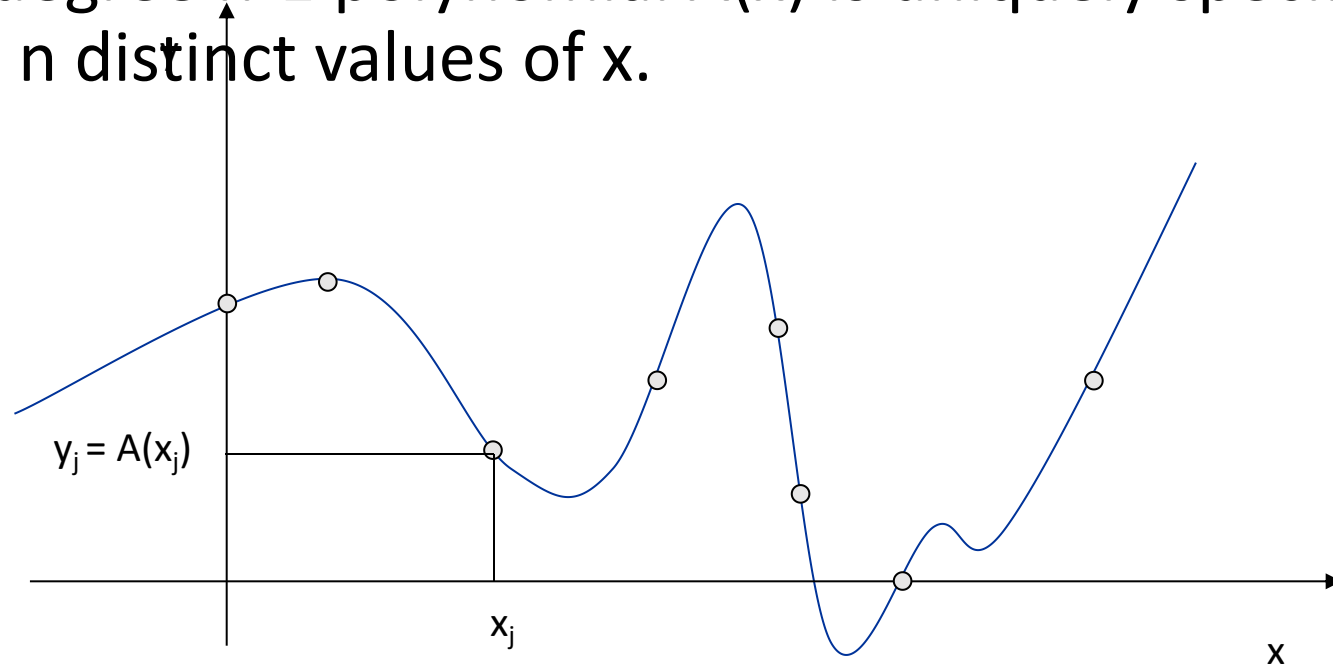
$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^i a_j b_{i-j}$$





# Polynomials: Point-Value Representation

- Fundamental theorem of algebra. [\[Gauss, PhD thesis\]](#) A degree  $n$  polynomial with complex coefficients has  $n$  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:  $O(n)$ , 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.

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

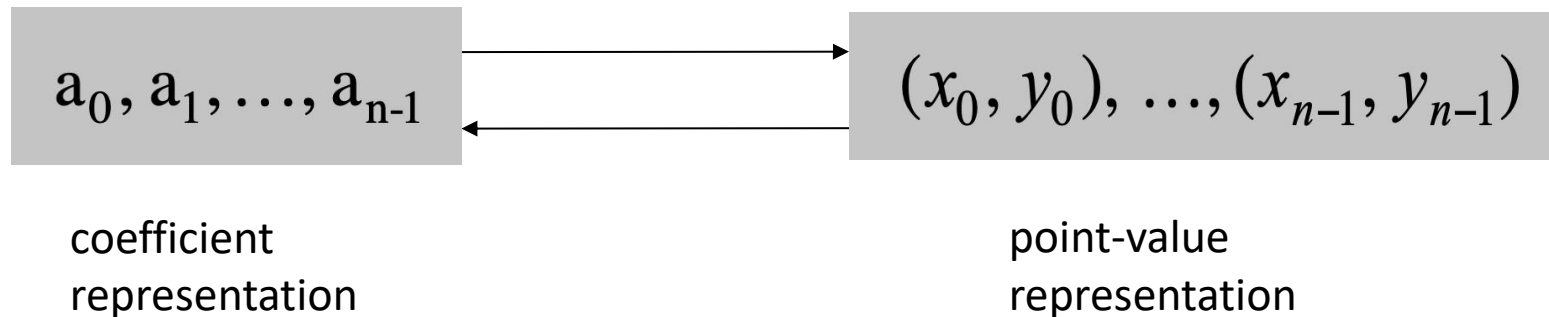


# Converting Between Two Polynomial Representations

- 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. Make all ops fast by efficiently converting between two representations.





# Converting Between Two Polynomial Representations: Brute Force

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

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

$O(n^2)$  for matrix-vector multiply

$O(n^3)$  for Gaussian elimination

Vandermonde matrix is invertible iff  $x_i$  distinct

- Point-value to coefficient. Given  $n$  distinct points  $x_0, \dots, x_{n-1}$  and values  $y_0, \dots, y_{n-1}$ , find unique polynomial  $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$  that has given values at given points.



# Coefficient to Point-Value Representation: Intuition

- Coefficient to point-value. Given a polynomial  $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ , evaluate it at  $n$  distinct points  $x_0, \dots, x_{n-1}$ .
- Divide. Break polynomial up into even and odd powers.
  - $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$ .
  - $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$ .
  - $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$ .
  - $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$ .
  - $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$ .
- 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 to point-value. Given a polynomial  $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ , evaluate it at  $n$  distinct points  $x_0, \dots, x_{n-1}$ .
- Divide. Break polynomial up into even and odd powers.
  - $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$ .
  - $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$ .
  - $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$ .
  - $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2)$ .
  - $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2)$ .
- Intuition. Choose four 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 2 points by evaluating two polynomials of  
degree  $\leq \frac{1}{2}n$  at 1 point.



# Discrete Fourier Transform

- Coefficient to point-value. Given a polynomial  $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ , evaluate it at  $n$  distinct points  $x_0, \dots, x_{n-1}$ .
- Key idea: choose  $x_k = \omega^k$  where  $\omega$  is principal  $n^{\text{th}}$  root of unity.

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

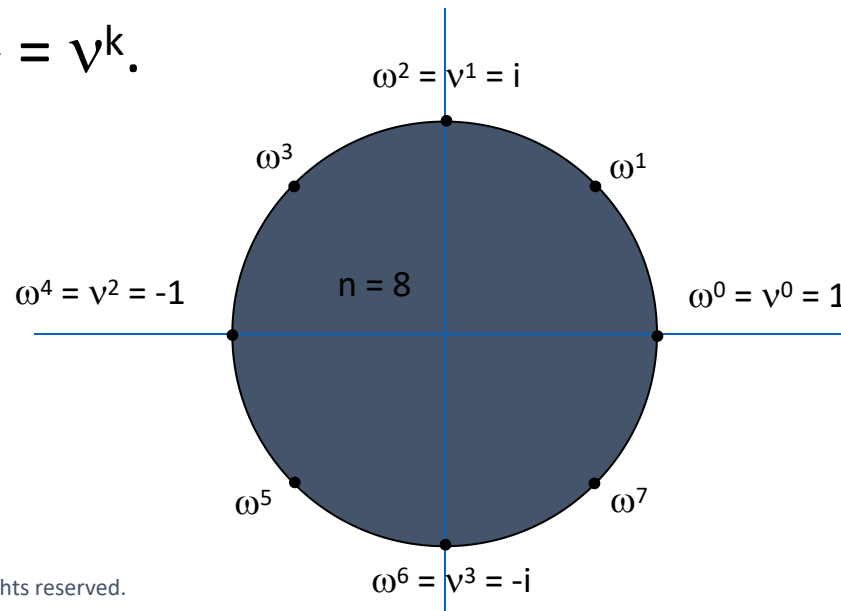
Discrete Fourier transform

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







# 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 polynomial up into even and odd powers.
  - $A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n/2-2} x^{(n-1)/2}.$
  - $A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n/2-1} x^{(n-1)/2}.$
  - $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$
- Conquer. Evaluate degree  $A_{\text{even}}(x)$  and  $A_{\text{odd}}(x)$  at the  $\frac{1}{2}n^{\text{th}}$  roots of unity:  $v^0, v^1, \dots, v^{n/2-1}$ .

$$v^k = (\omega^k)^2 = (\omega^{k+\frac{1}{2}n})^2$$

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

$$\begin{array}{c} \uparrow \\ \omega^{k+\frac{1}{2}n} = -\omega^k \end{array}$$



# 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)  
}
```

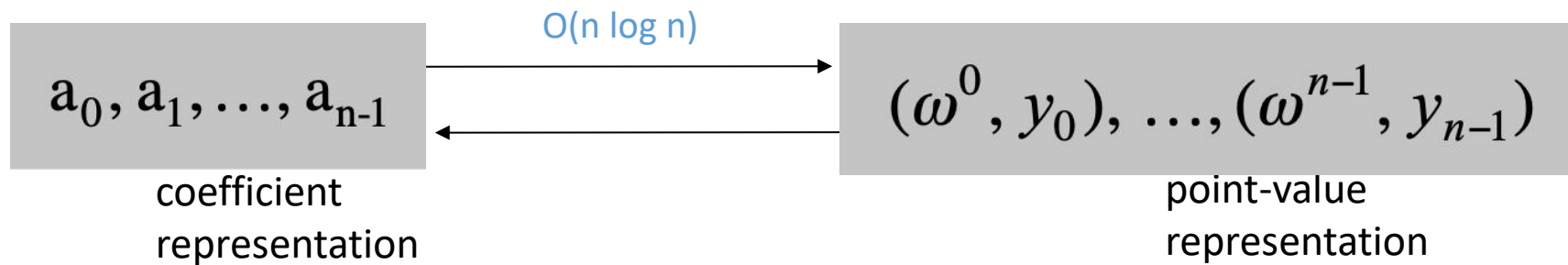


# FFT Summary

- 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.

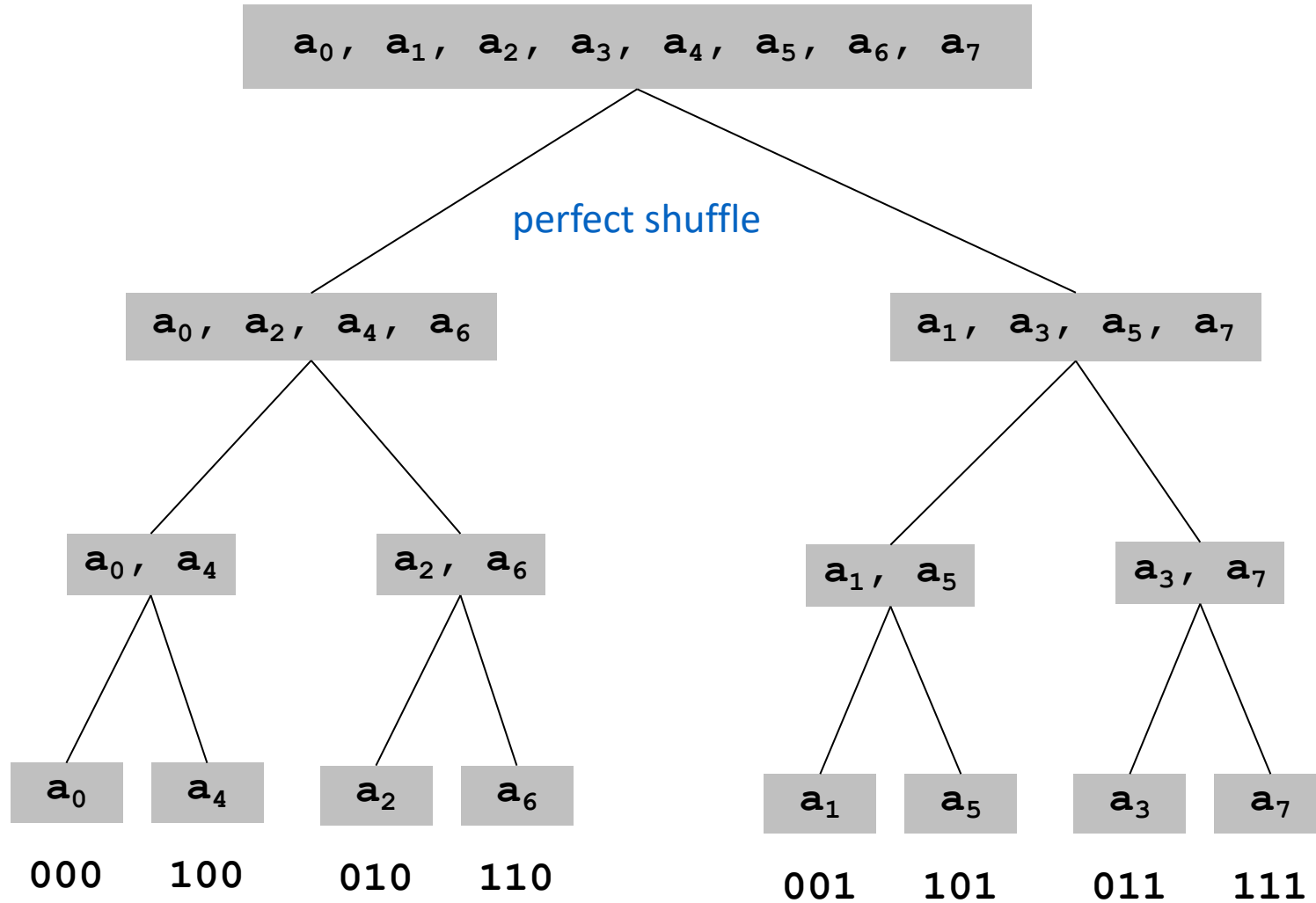
assumes  $n$  is a power of 2

- Running time.  $T(2n) = 2T(n) + O(n) \Rightarrow T(n) = O(n \log n)$ .





# Recursion Tree



"bit-reversed" order



# Point-Value to Coefficient Representation: Inverse DFT

- Goal. Given the values  $y_0, \dots, y_{n-1}$  of a degree  $n-1$  polynomial at the  $n$  points  $\omega^0, \omega^1, \dots, \omega^{n-1}$ , find unique polynomial  $a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$  that has given values at given points.

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

↑  
Inverse DFT

↑  
Fourier matrix inverse  $(F_n)^{-1}$



# Inverse FFT

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

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

- 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.

$$(F_n G_n)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma

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

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

- Pf.
  - If  $k$  is a multiple of  $n$  then  $\omega^k = 1 \Rightarrow$  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:  $1 + \omega^k + \omega^{k(2)} + \dots + \omega^{k(n-1)} = 0 \Rightarrow$  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      ← (ek + ωk dk) / n  
        yk+n/2 ← (ek - ωk dk) / n  
    }  
  
    return (y0, y1, ..., yn-1)  
}
```

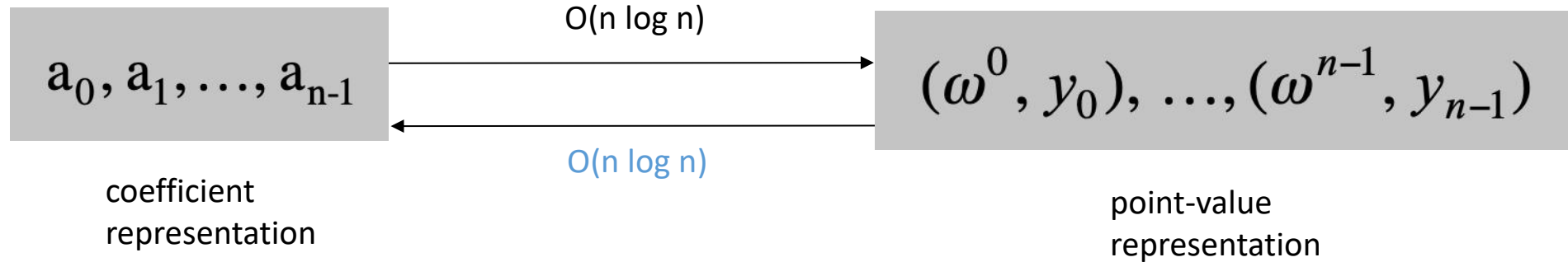




# Inverse FFT Summary

- 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.

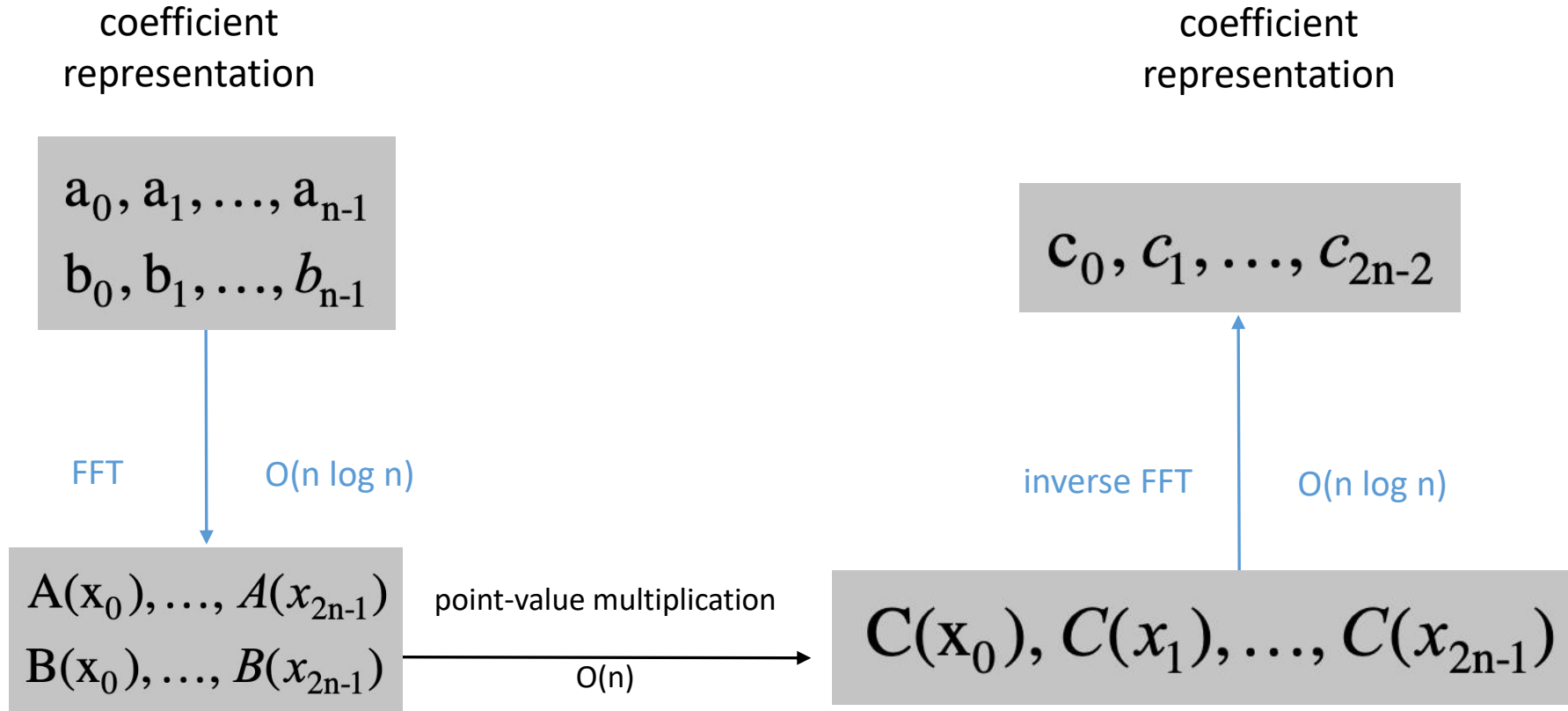
↑  
assumes  $n$  is a power of 2





# Polynomial Multiplication

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





# Integer Multiplication

- 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  $c = a \times b$ .
- Convolution algorithm.
  - Form two polynomials.
  - Note:  $a = A(2)$ ,  $b = B(2)$ .
  - Compute  $C(x) = A(x) \times B(x)$ .
  - Evaluate  $C(2) = a \times b$ .
  - 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.
- Practice. [GNU Multiple Precision Arithmetic Library] GMP proclaims to be "the fastest bignum library on the planet." It uses brute force, Karatsuba, and FFT, depending on the size of  $n$ .

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

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



# FFT in Practice ?

fft java - Google Search

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

Google Movies Weather Tech News Sports Princeton CS Java 1.5 Book 1 Book 2 Courses Other

Sign in

Web Images Groups News Froogle Local Scholar more »

Google 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](#)

**YOV408 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.yov408.com/html/codespot.php?gg=35](http://www.yov408.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, RGBaldwin 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](#)

**Codecs - FFT Java**



# 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 radix 2 FFT.
  - $O(n \log n)$ , even for prime sizes.