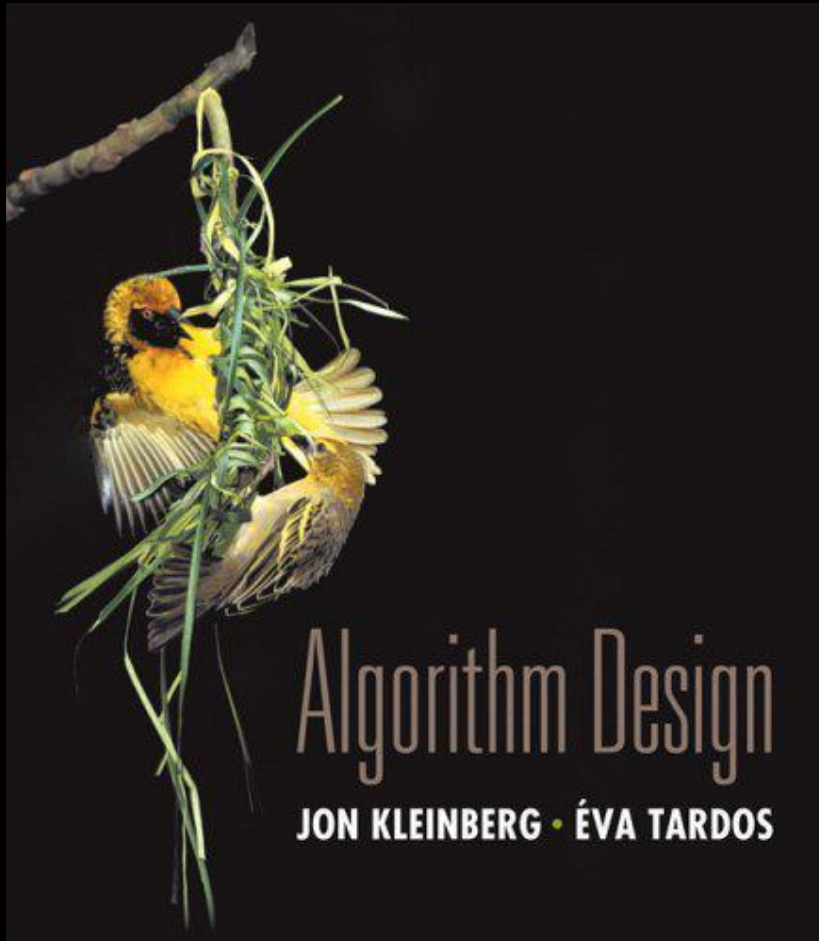


# How to Multiply

integers, matrices, and polynomials



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

# 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.

## Divide into more than 2 subproblems

What happens if the divide-and-conquer algorithms that create recursive calls on  $q$  sub-problems of size  $n/2$  each with  $q > 2$ ?

If  $T(n)$  obeys the following recurrence relation

$$T(n) \leq qT(n/2) + cn$$

when  $n > 2$  and  $T(2) \leq c$ .

$T(\cdot)$  satisfying the above with  $q > 2$  is bounded by  $O(n^{\log_2 q})$ .

When  $q=3$ ,  $O(n^{\log_2 q}) = O(n^{1.585})$

When  $q=4$ ,  $O(n^{\log_2 q}) = O(n^2)$

For details, please read the Section 5.2 of the Textbook

## 5.5 Integer Multiplication

---

# Integer Addition

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

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

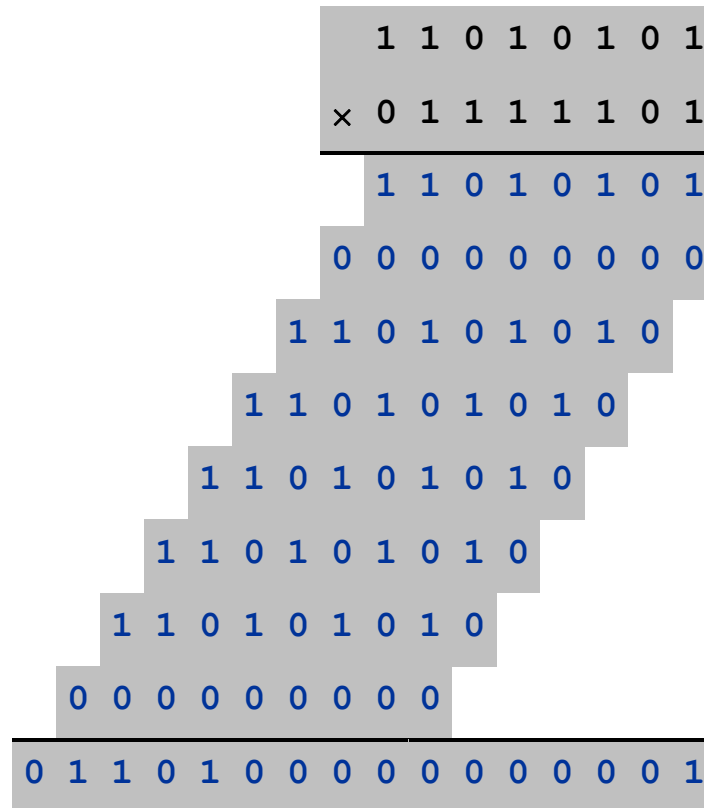
	1	1	1	1	1	1	0	1	
		1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	1	0	1
<hr/>									
	1	0	1	0	1	0	0	1	0

**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.

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

Ex.  $a = \underbrace{1000}_{a_1} \underbrace{1101}_{a_0} \quad b = \underbrace{1110}_{b_1} \underbrace{0001}_{b_0}$

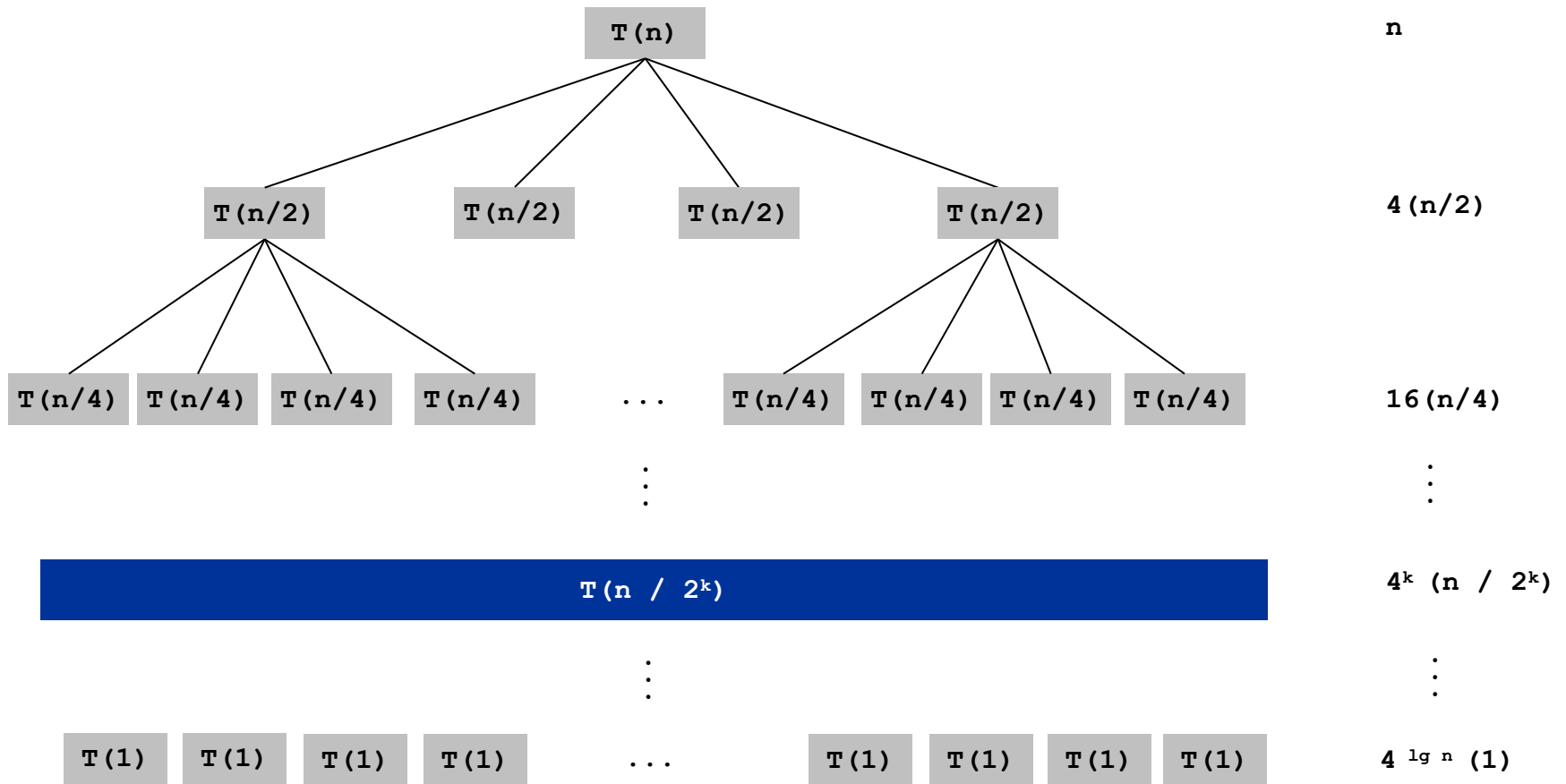
$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$



# 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.

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

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

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

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

1

2

1

3

3

# 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.

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

(1)                      (2)                      (1)                      (3)                      (3)

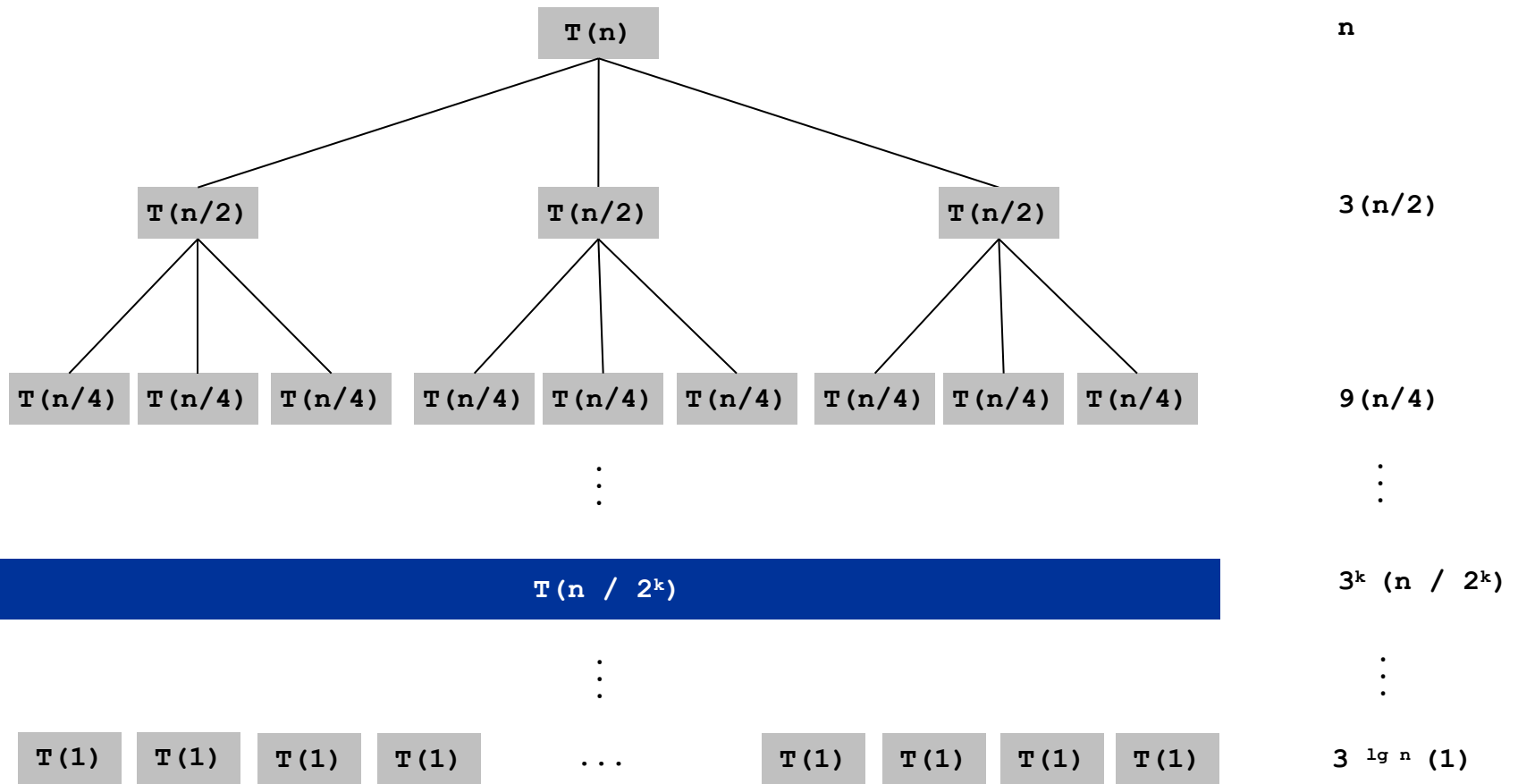
**Theorem.** [Karatsuba-Ofman 1962] Can multiply two  $n$ -bit integers in  $O(n^{1.585})$  bit operations.

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

# Karatsuba: Recursion Tree

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

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

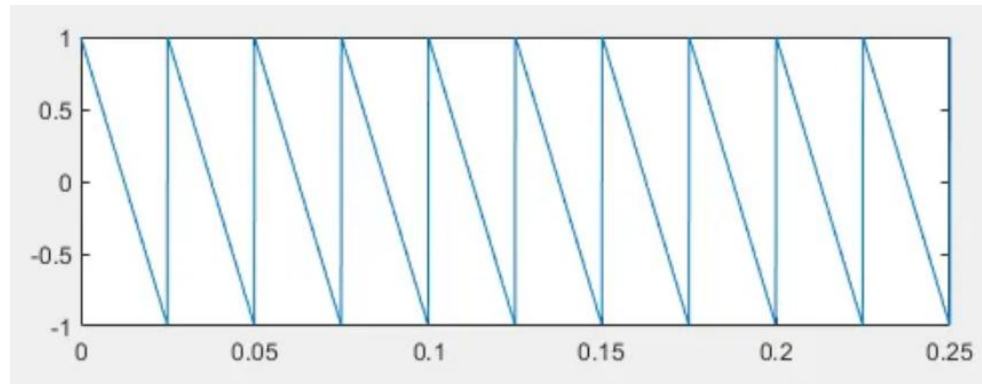


## 5.6 Convolution and FFT

---

# Fourier Analysis

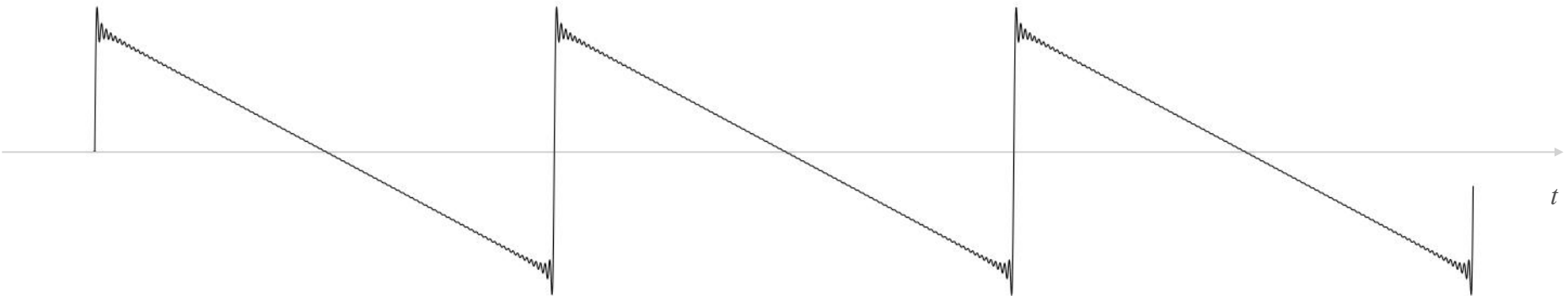
**Fourier theorem.** [Fourier, Dirichlet, Riemann] Any periodic function can be expressed as the sum of a series of sinusoids. ← sufficiently smooth



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

# Fourier Analysis

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



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

## Euler's Identity

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

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

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

$$\bullet e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$$

$$\bullet e^{j\omega t} = \cos(\omega t) + j\sin(\omega t)$$

*Euler's identity*

**Sinusoids.** Sum of complex exponentials.

$$\cos(\omega t) = \frac{1}{2}(e^{j\omega t} + e^{-j\omega t})$$

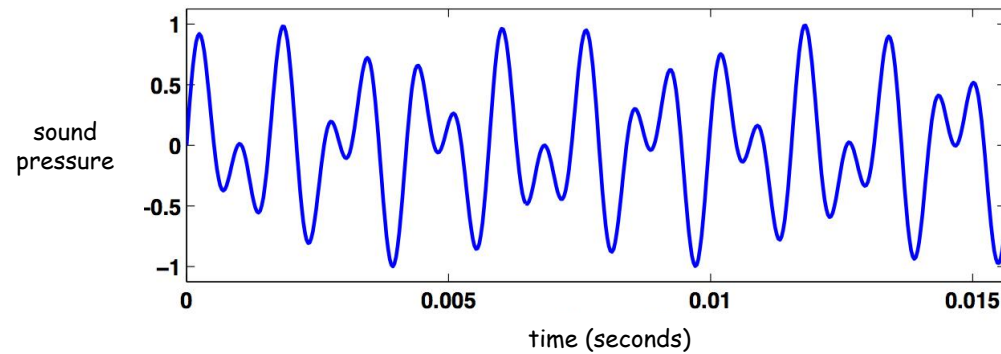
$$\sin(\omega t) = \frac{1}{2j}(e^{j\omega t} - e^{-j\omega t})$$



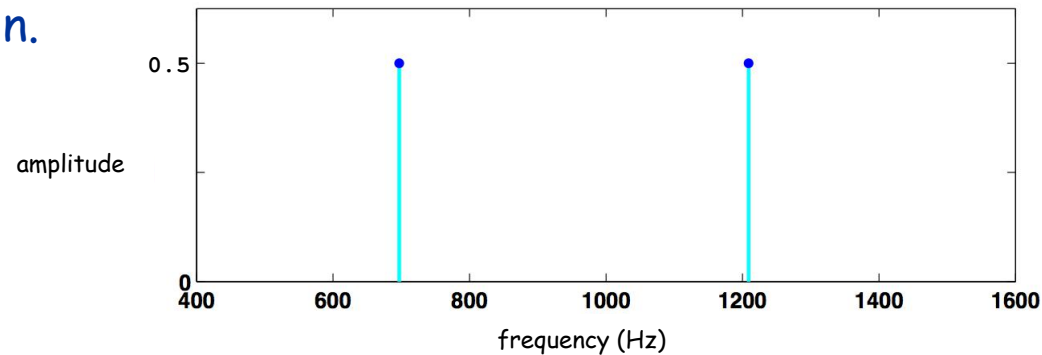
# Time Domain vs. Frequency Domain

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.



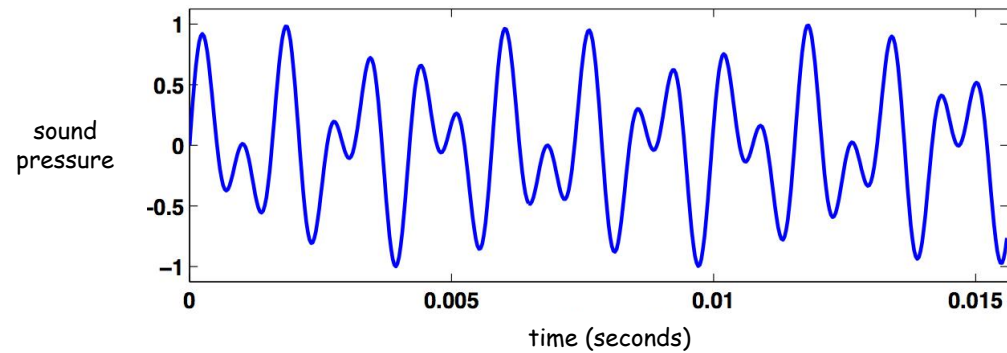
Frequency domain.



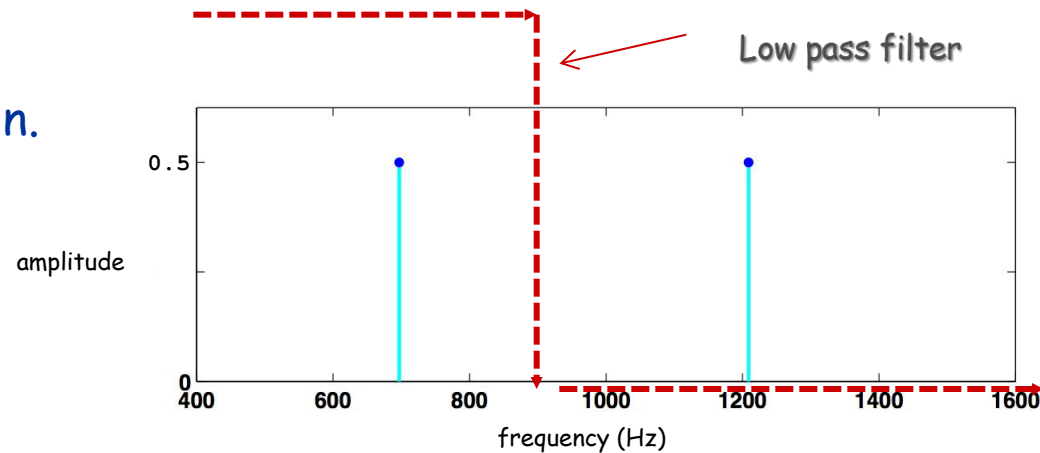
# Time Domain vs. Frequency Domain

Signal. [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.

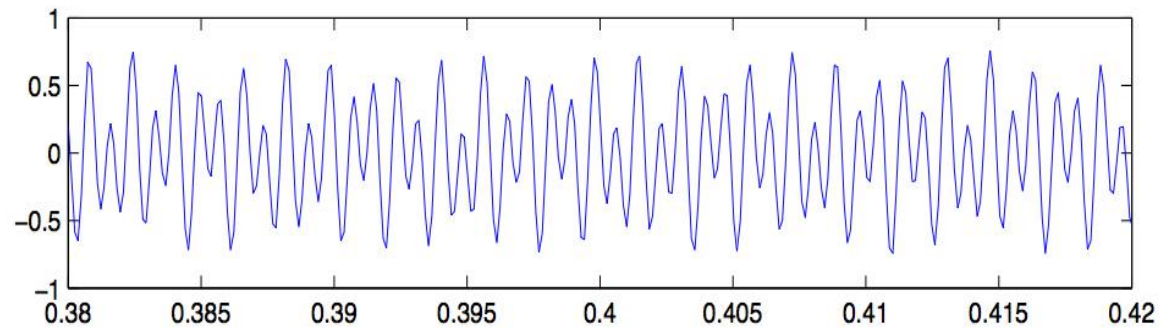


Frequency domain.

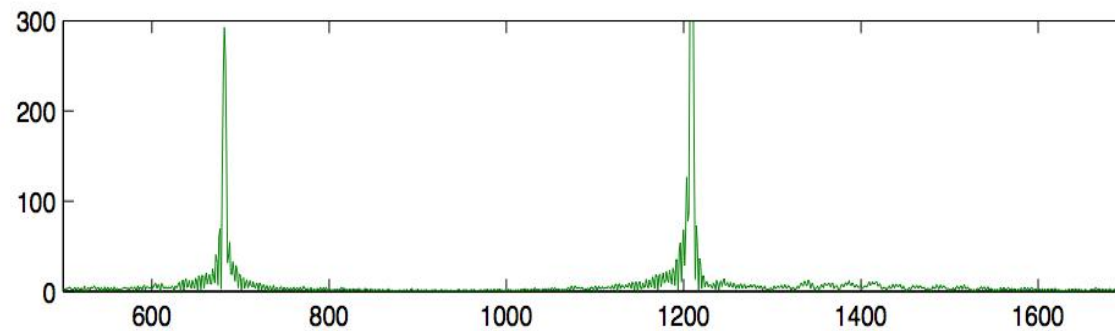


# Time Domain vs. Frequency Domain

Signal. [recording, 8192 samples per second]

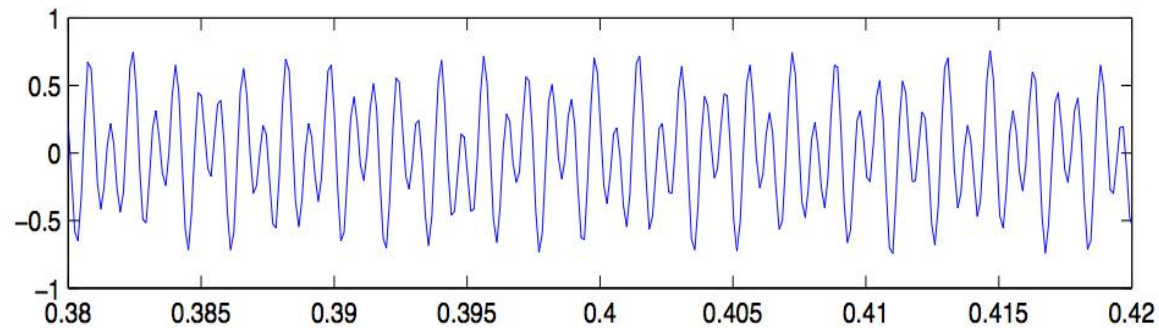


Magnitude of discrete Fourier transform.

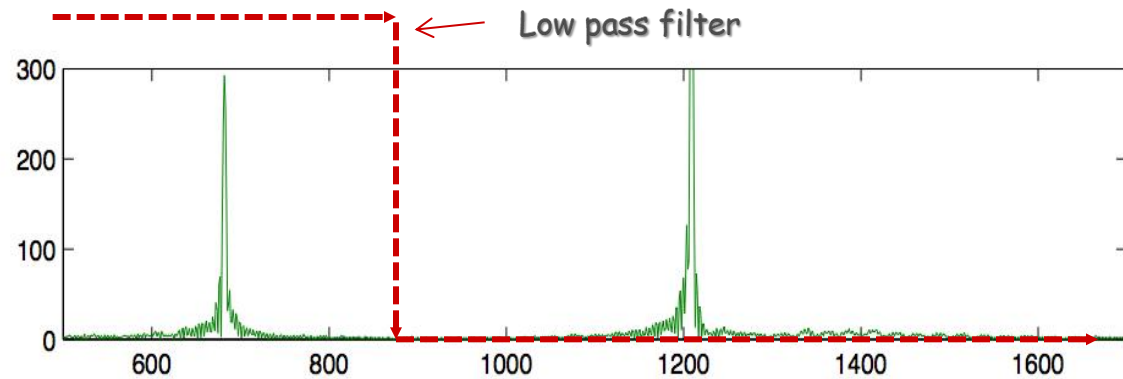


# Time Domain vs. Frequency Domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.

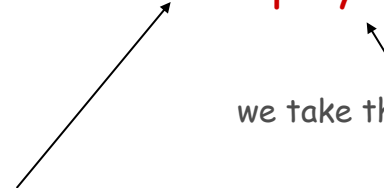


# Fast Fourier Transform

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

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

we take this approach


$$\begin{aligned}A(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_{m-1}x^{m-1} \\B(x) &= b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1} \\C(x) &= A(x)B(x) = c_0 + c_1x + c_2x^2 + \cdots + c_kx^k + \cdots + c_{n+m-2}x^{n+m-2} \\c_k &= \sum_{(i,j): i+j=k} a_ib_j\end{aligned}$$

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

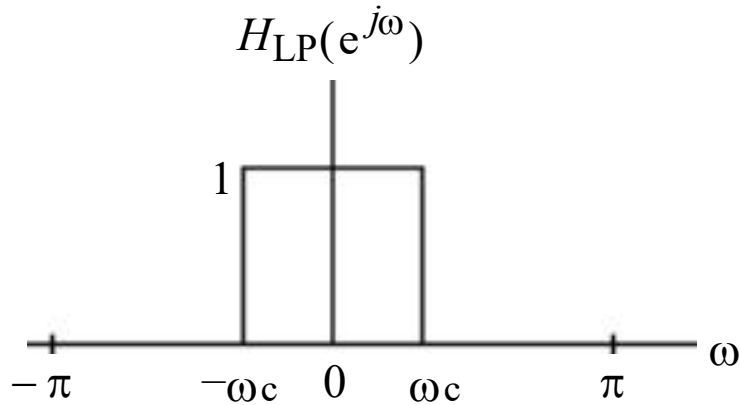
## Applications.

- 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

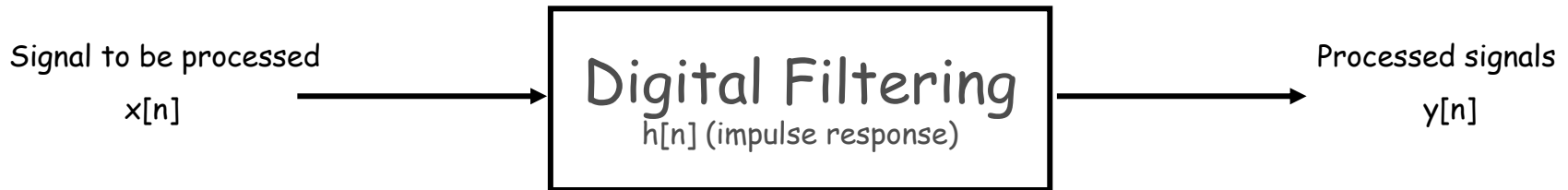
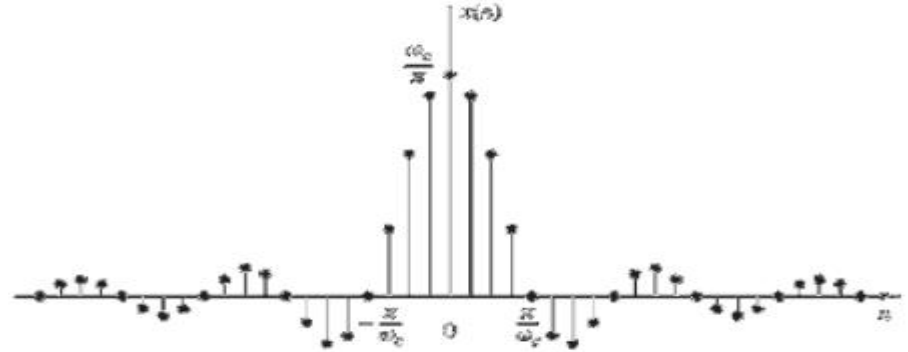
# Digital filtering

## *Ideal low-pass filter*



$$h_{LP}[n] = \frac{\sin \omega_c n}{\pi n}, \quad -\infty \leq n \leq \infty$$

impulse response

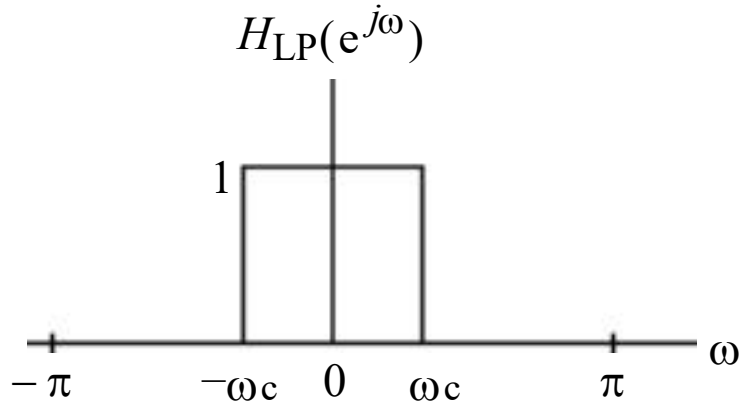


$$Y(\omega) = X(\omega) H(\omega) \longrightarrow y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n-k] = \sum_{k=-\infty}^{\infty} x[n-k] h[k]$$

↑  
Convolution

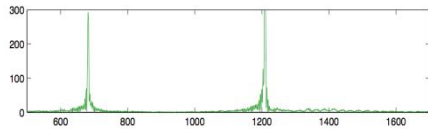
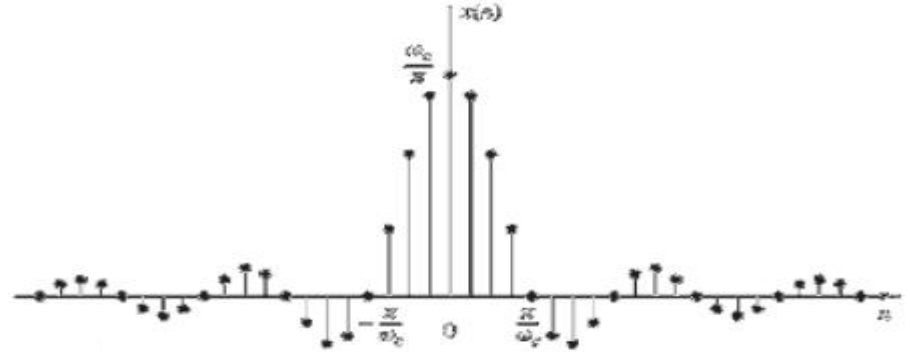
# Digital filtering

## *Ideal low-pass filter*

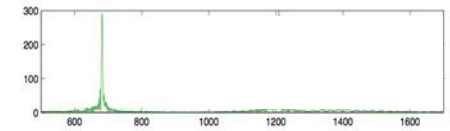
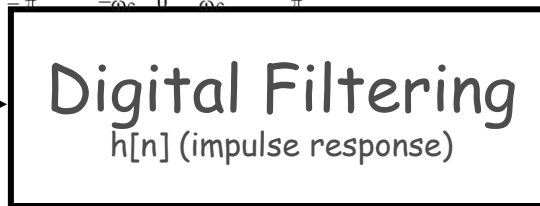
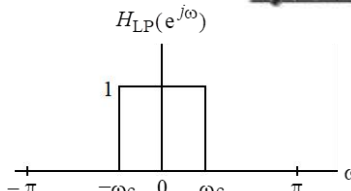


$$h_{LP}[n] = \frac{\sin \omega_c n}{\pi n}, \quad -\infty \leq n \leq \infty$$

impulse response



Signal to be processed  
 $x[n]$



Processed signals  
 $y[n]$

$$Y(w) = X(w) H(w) \longrightarrow y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n-k] = \sum_{k=-\infty}^{\infty} x[n-k] h[k]$$

Convolution

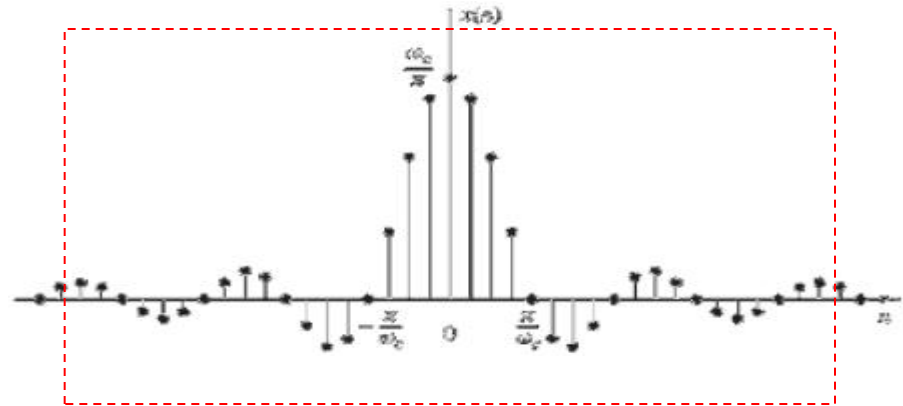
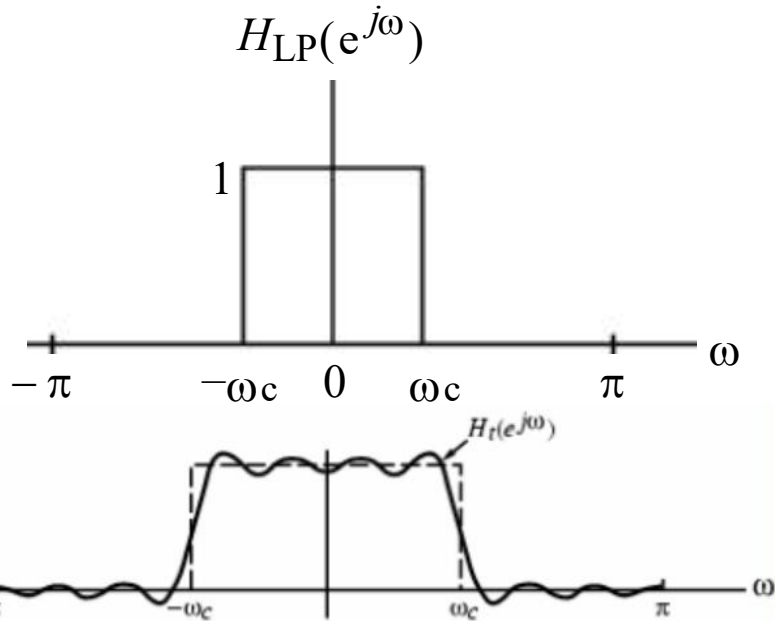


# Digital filtering

## *Ideal low-pass filter*

impulse response

$$h_{LP}[n] = \frac{\sin \omega_c n}{\pi n}, \quad -\infty \leq n \leq \infty$$



Signal to be processed  
 $x[n]$

Digital Filtering  
 $h[n]$  (impulse response)

Processed signals  
 $y[n]$

$$Y(\omega) = X(\omega) H(\omega)$$



$$y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n-k] = \sum_{k=-\infty}^{\infty} x[n-k] h[k]$$

Convolution

Truncate to limited number

# 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.

## Fourier Series (FS)

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Fourier Series (FS)

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Continuous Time Fourier Transform (CTFT)

- Extension of Fourier series to non-periodic functions: Any continuous aperiodic function can be represented as an infinite sum (integral) of sinusoids. The sinusoids are no longer integer multiples of a specific frequency.

## Fourier Series (FS)

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Continuous Time Fourier Transform (CTFT)

- Extension of Fourier series to non-periodic functions: Any continuous aperiodic function can be represented as an infinite sum (integral) of sinusoids. The sinusoids are no longer integer multiples of a specific frequency.

## Discrete Time Fourier Transform (DTFT)

- Extension of FT to discrete sequences. Any discrete function can also be represented as an infinite sum (integral) of sinusoids. While time domain is discretized, frequency domain is still continuous.

## Fourier Series (FS)

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Continuous Time Fourier Transform (CTFT)

- Extension of Fourier series to non-periodic functions: Any continuous aperiodic function can be represented as an infinite sum (integral) of sinusoids. The sinusoids are no longer integer multiples of a specific frequency.

## Discrete Time Fourier Transform (DTFT)

- Extension of FT to discrete sequences. Any discrete function can also be represented as an infinite sum (integral) of sinusoids. While time domain is discretized, frequency domain is still continuous.

## Discrete Fourier Transform (DFT)

- Because DTFT is defined as an infinite sum, the frequency representation is not discrete. An extension to DTFT is DFT, where the frequency variable is also discretized.

## Fourier Series (FS)

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Continuous Time Fourier Transform (CTFT)

- Extension of Fourier series to non-periodic functions: Any continuous aperiodic function can be represented as an infinite sum (integral) of sinusoids. The sinusoids are no longer integer multiples of a specific frequency.

## Discrete Time Fourier Transform (DTFT)

- Extension of FT to discrete sequences. Any discrete function can also be represented as an infinite sum (integral) of sinusoids. While time domain is discretized, frequency domain is still continuous.

## Discrete Fourier Transform (DFT)

- Because DTFT is defined as an infinite sum, the frequency representation is not discrete. An extension to DTFT is DFT, where the frequency variable is also discretized.

## Fast Fourier Transform (FFT)

- Mathematically identical to DFT, however a significantly more efficient implementation. FFT is what signal processing made possible today!

## Fourier Series (FS)

Continuous vs Discrete

- Fourier's original work: A periodic function can be represented as a finite, weighted sum of sinusoids that are integer multiples of the fundamental frequency  $\Omega_0$  of the signal. These frequencies are said to be harmonically related, or simply harmonics.

## Continuous Time Fourier Transform (CTFT)

Continuous vs Continuous

- Extension of Fourier series to non-periodic functions: Any continuous aperiodic function can be represented as an infinite sum (integral) of sinusoids. The sinusoids are no longer integer multiples of a specific frequency.

## Discrete Time Fourier Transform (DTFT)

Discrete vs Continuous

- Extension of FT to discrete sequences. Any discrete function can also be represented as an infinite sum (integral) of sinusoids. While time domain is discretized, frequency domain is still continuous.

## Discrete Fourier Transform (DFT)

Discrete vs Discrete

- Because DTFT is defined as an infinite sum, the frequency representation is not discrete. An extension to DTFT is DFT, where the frequency variable is also discretized.

## Fast Fourier Transform (FFT)

- Mathematically identical to DFT, however a significantly more efficient implementation. FFT is what signal processing made possible today!



# 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.

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

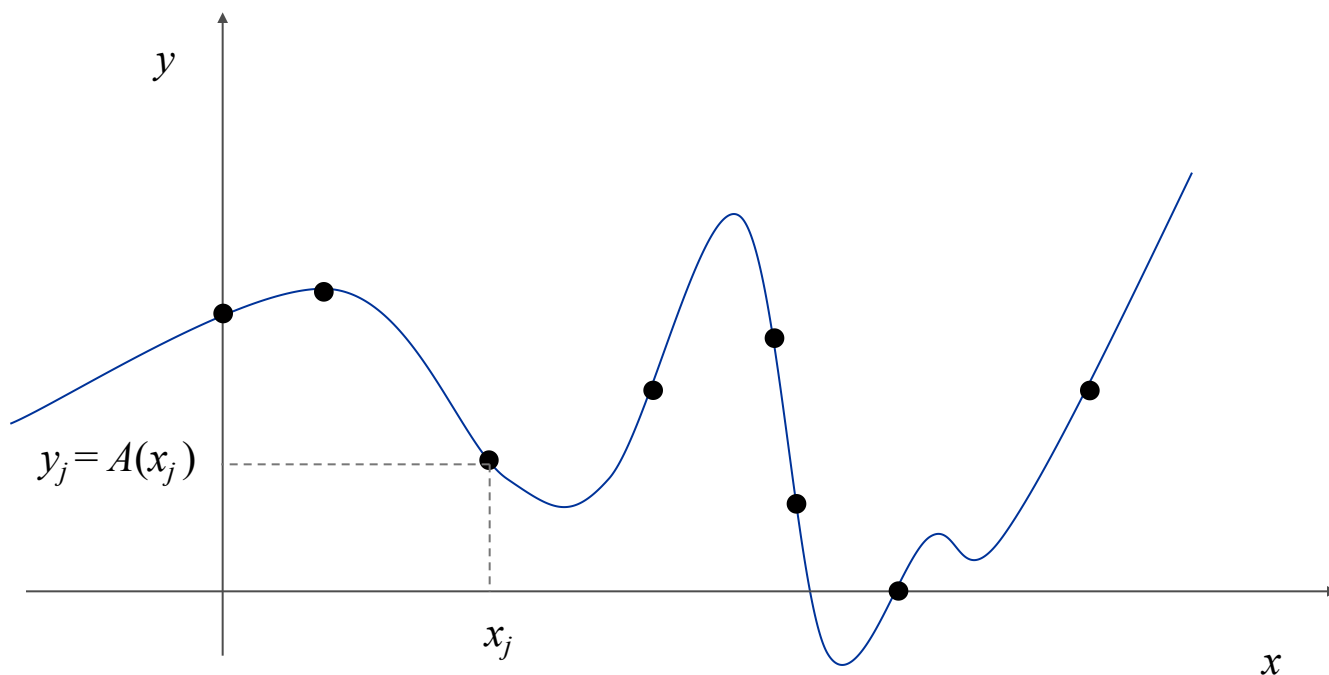
**Multiply (convolve).**  $O(n^2)$  using brute force.

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

# Polynomials: Point-Value Representation

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

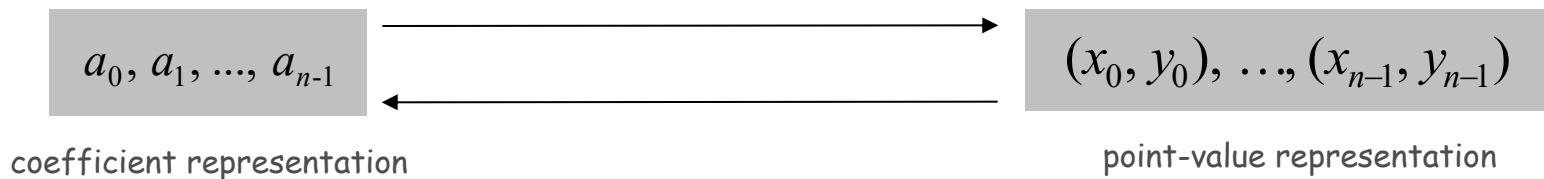
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

# 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  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , 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  $F_n$

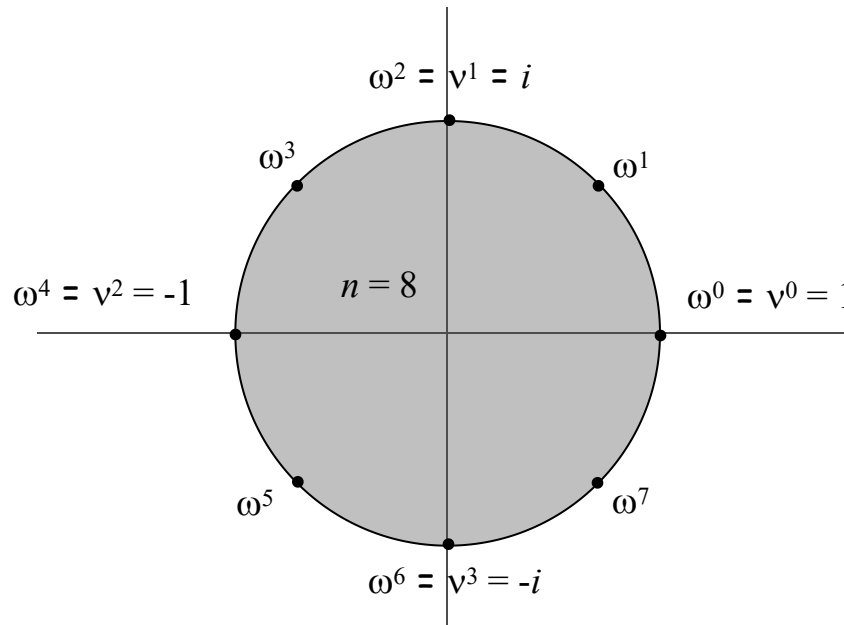
# Roots of Unity

**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:  $v^0, v^1, \dots, v^{n/2-1}$ .

$2T(n/2)$

**Combine.**

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

$v^k = (\omega^k)^2$

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

$\omega^{k+\frac{1}{2}n} = -\omega^k$

$O(n)$

# 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)  
}
```

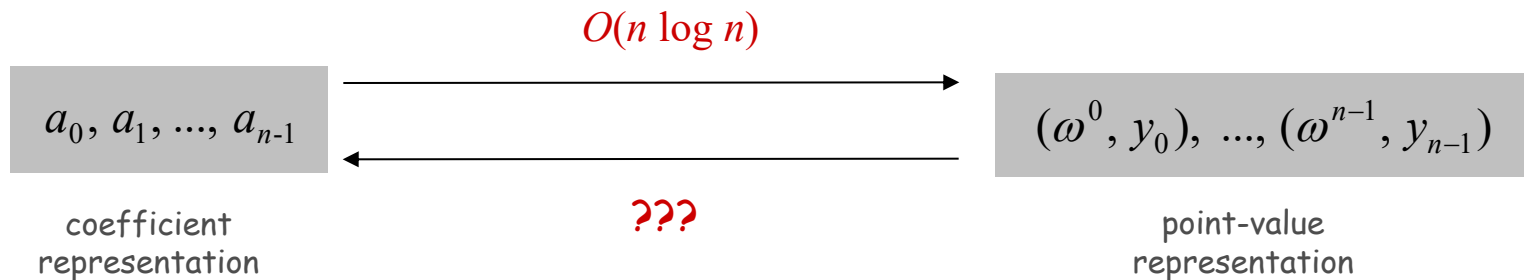
# FFT Summary

**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.

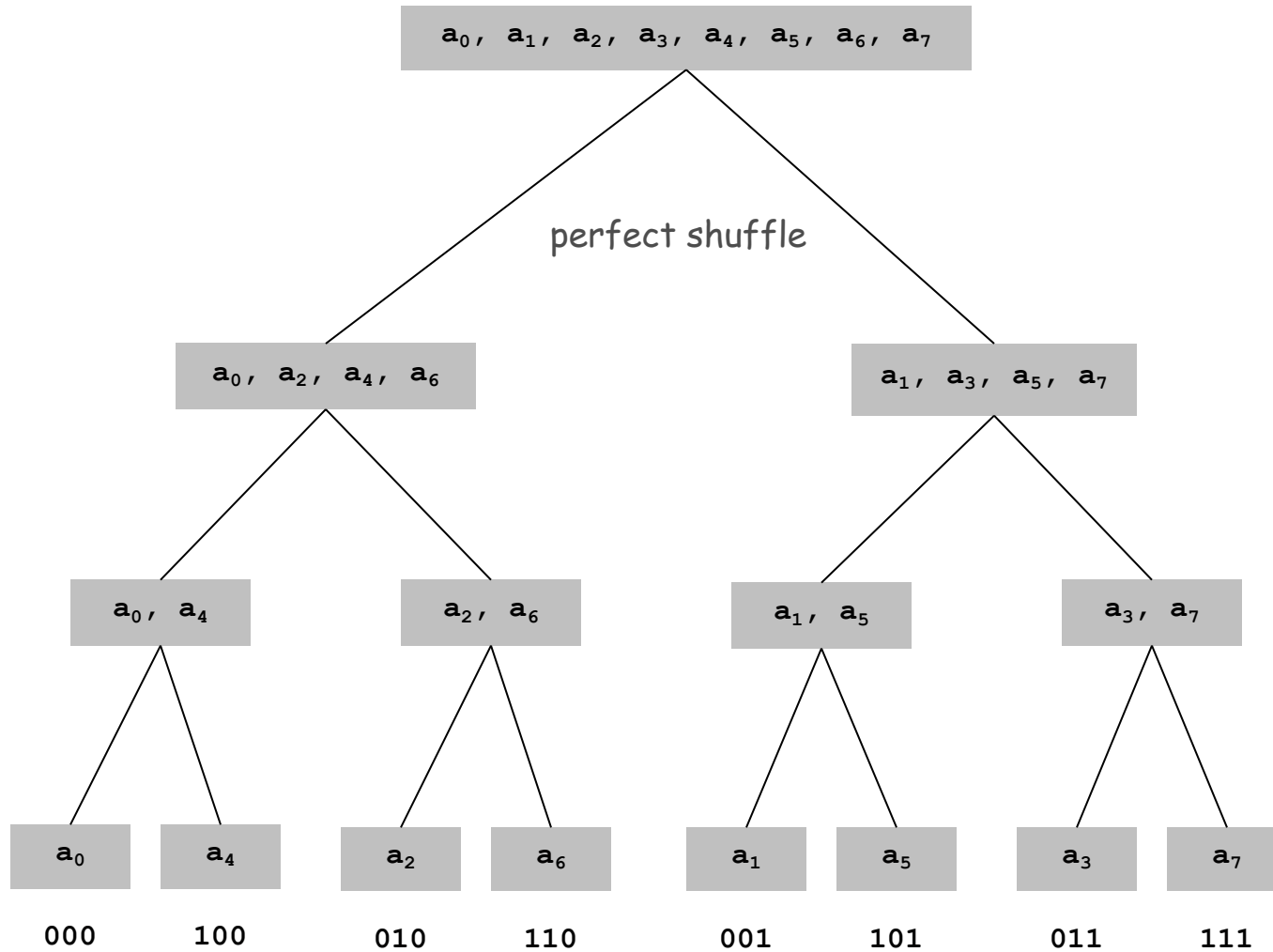
↖  
assumes  $n$  is a power of 2

Running time.

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$



# Recursion Tree

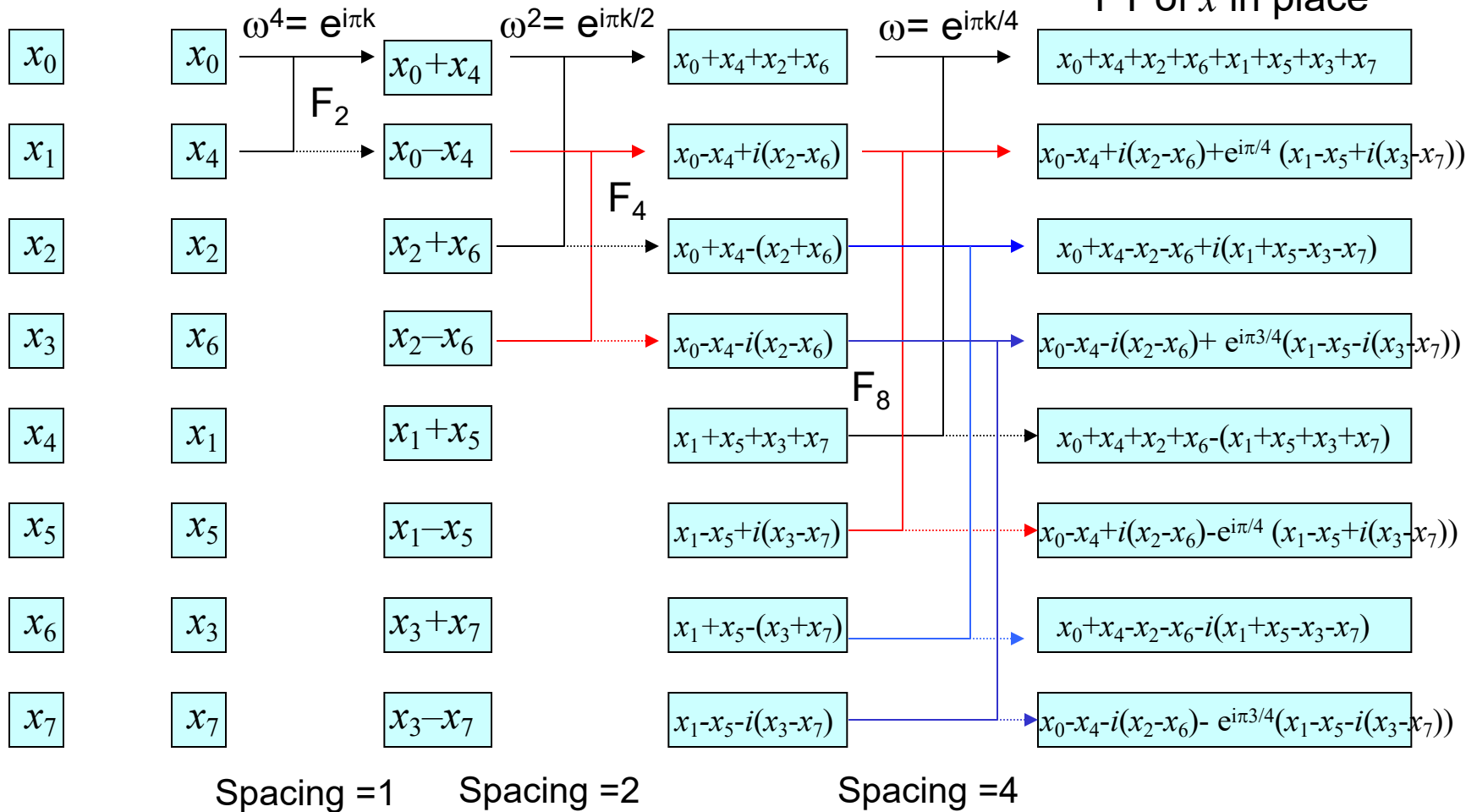


"bit-reversed" order

Backtracking

# Example of FFT

Swap data  
according to bit  
reversal





# 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.

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

↑  
Inverse DFT

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

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

# Inverse DFT

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

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

$\frac{1}{\sqrt{n}} F_n$  is unitary

**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  $\omega^k = 1 \Rightarrow$  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. ■

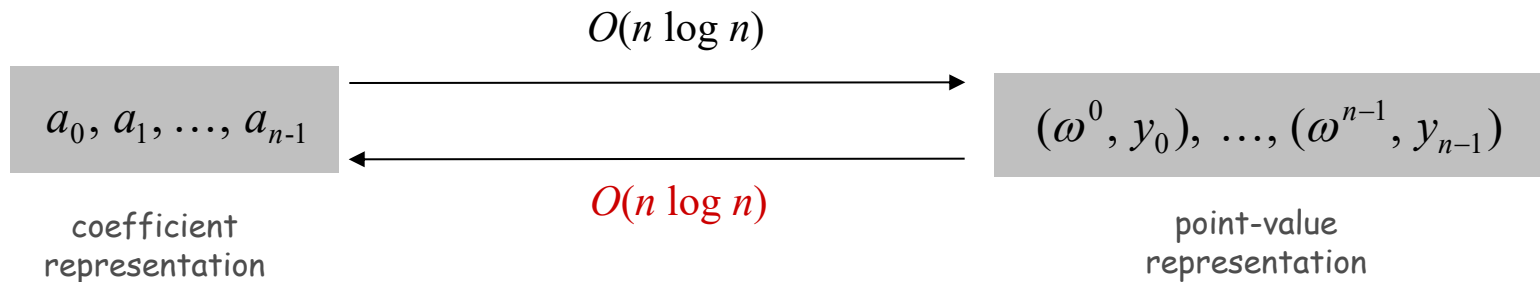
# Inverse FFT: Algorithm

```
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

