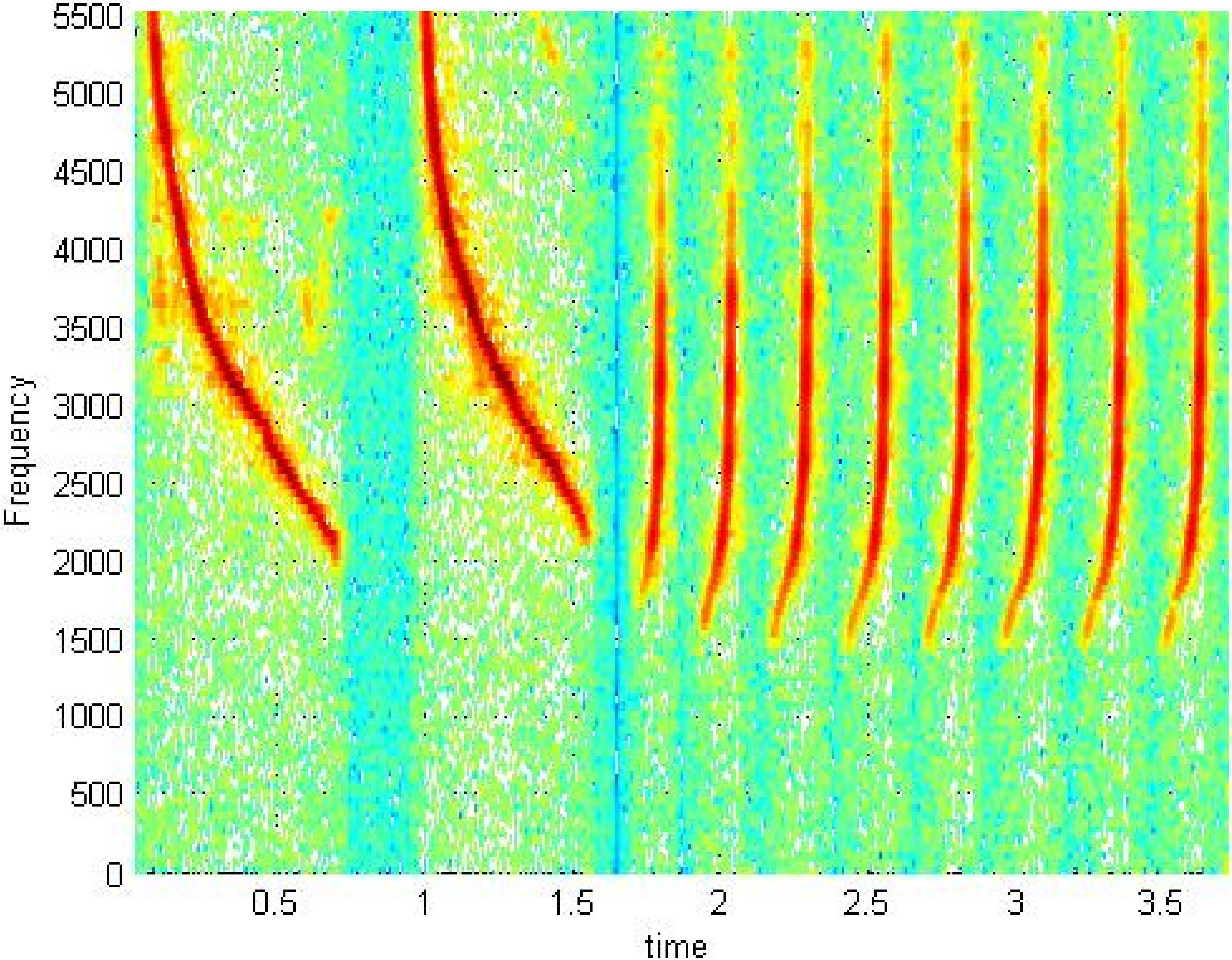
Fourier Transforms

COS 323

Life in the Frequency Domain

Jean Baptiste Joseph Fourier (1768-1830)

Spectrogram, Northern Cardinal



JPEG Image Compression

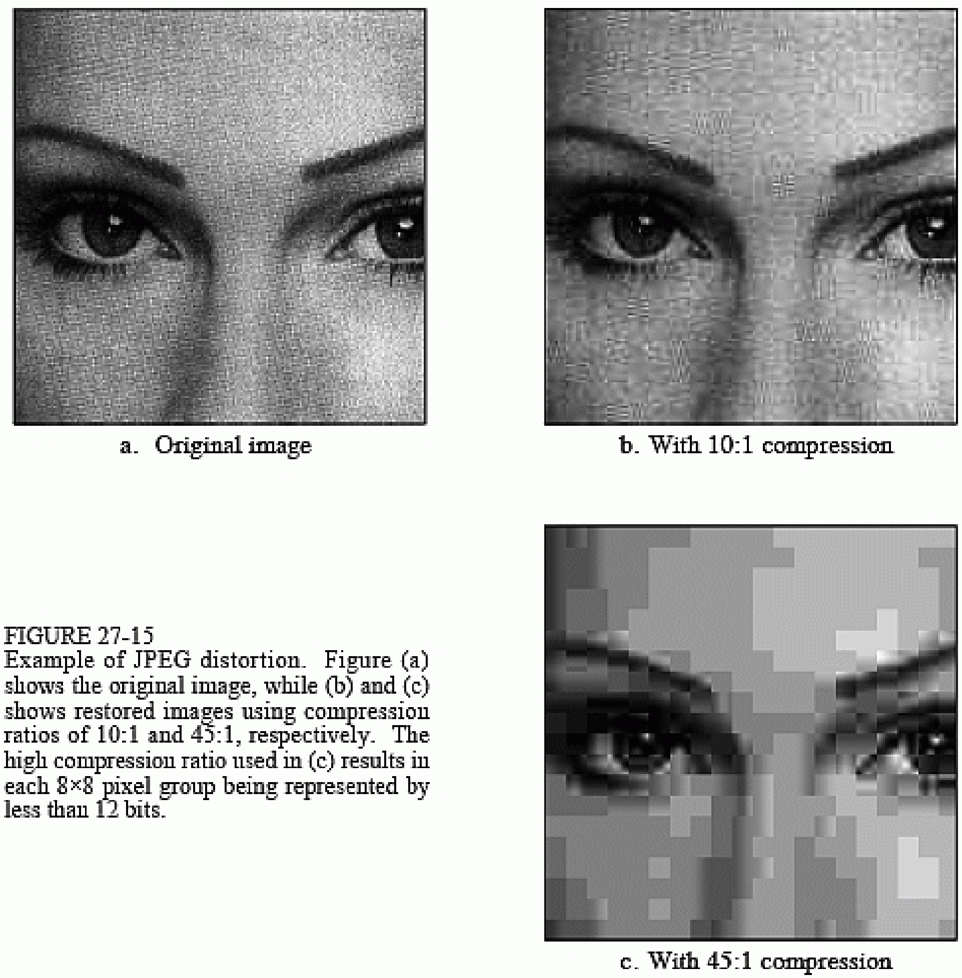


FIGURE 27-15
Example of JPEG distortion. Figure (a) shows the original image, while (b) and (c) shows restored images using compression ratios of 10:1 and 45:1, respectively. The high compression ratio used in (c) results in each 8×8 pixel group being represented by less than 12 bits.

Discrete Cosine Transform (DCT)

[Steven W. Smith 1997]

The Convolution Theorem

- Fourier transform turns convolution into multiplication:

$$\mathcal{F}\big(f(x) * g(x)\big) = \mathcal{F}\big(f(x)\big)\ \mathcal{F}\big(g(x)\big)$$

(and vice versa):

$$\mathcal{F}\big(f(x)\, g(x)\big) = \mathcal{F}\big(f(x)\big) * \mathcal{F}\big(g(x)\big)$$

Discrete Fourier Transform (DFT)

$$F_k = \sum_{x=0}^{n-1} f_x \, e^{-2\pi i \frac{k}{n} x}$$

- *F* is a function of frequency – describes how much of each frequency *f* contains
- Fourier transform is invertible

Inverse DFT (IDFT)

$$F_k = \sum_{x=0}^{n-1} f_x \, e^{-2\pi i \frac{k}{n} x}$$

$$f_x = \frac{1}{n} \sum_{k=0}^{n-1} F_k e^{2\pi i \frac{k}{n} x}$$

Computing Discrete Fourier Transform

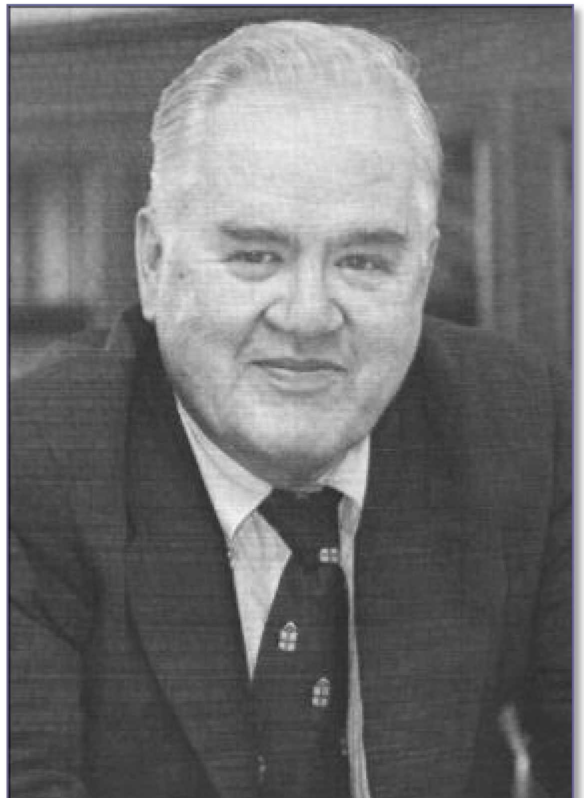
$$F_k = \sum_{x=0}^{n-1} f_x \, e^{-2\pi i \frac{k}{n} x}$$

- Straightforward computation: for each of *n* DFT values, loop over *n* input samples. Total: $O(n^2)$
- Fast Fourier Transform (FFT): $O(n \log_2 n)$ time
  - Revolutionized signal processing, filtering, compression, etc.

The FFT

Discovered by Johann Carl Friedrich Gauss (1777-1855)

The FFT



Rediscovered and popularized in 1965 by
J. W. Cooley and John Tukey (Princeton alum and faculty)

The FFT

$$F_k = \sum_{x=0}^{n-1} f_x \, e^{-2\pi i \frac{k}{n} x}$$

Let $\omega_n = e^{-2\pi i / n} = \cos(2\pi / n) - i \sin(2\pi / n)$

Then $F_k = \sum_{x=0}^{n-1} f_x \omega_n^{xk}$

$$= \sum_{x=0}^{n/2-1} f_{2x} \omega_n^{2xk} + \sum_{x=0}^{n/2-1} f_{2x+1} \omega_n^{(2x+1)k}$$

The FFT

Key idea: divide and conquer

– Separate computation on even and odd elements

$$F_k = \sum_{x=0}^{n/2-1} f_{2x}\omega_n^{2xk} + \sum_{x=0}^{n/2-1} f_{2x+1}\omega_n^{(2x+1)k}$$

$$= \sum_{x=0}^{n/2-1} f_{2x}\omega_{n/2}^{xk} + \omega_n^{k} \sum_{x=0}^{n/2-1} f_{2x+1}\omega_{n/2}^{xk}$$

Half-size FFT on even elements

Half-size FFT on odd elements

Example (n = 4)

- From the definition:

$$
\begin{aligned}
F_0 &= f_0\omega_n^{0\cdot 0} + f_1\omega_n^{1\cdot 0} + f_2\omega_n^{2\cdot 0} + f_3\omega_n^{3\cdot 0} \\
F_1 &= f_0\omega_n^{0\cdot 1} + f_1\omega_n^{1\cdot 1} + f_2\omega_n^{2\cdot 1} + f_3\omega_n^{3\cdot 1} \\
F_2 &= f_0\omega_n^{0\cdot 2} + f_1\omega_n^{1\cdot 2} + f_2\omega_n^{2\cdot 2} + f_3\omega_n^{3\cdot 2} \\
F_3 &= f_0\omega_n^{0\cdot 3} + f_1\omega_n^{1\cdot 3} + f_2\omega_n^{2\cdot 3} + f_3\omega_n^{3\cdot 3}
\end{aligned}
$$

Example (n = 4)

- Using the fact that $\omega_n^4 = 1$

$$F_0 = f_0\omega_n^0 + f_1\omega_n^0 + f_2\omega_n^0 + f_3\omega_n^0$$
$$F_1 = f_0\omega_n^0 + f_1\omega_n^1 + f_2\omega_n^2 + f_3\omega_n^3$$
$$F_2 = f_0\omega_n^0 + f_1\omega_n^2 + f_2\omega_n^0 + f_3\omega_n^2$$
$$F_3 = f_0\omega_n^0 + f_1\omega_n^3 + f_2\omega_n^2 + f_3\omega_n^5$$

Example (n = 4)

- Split even and odd terms, factor:

$$
\begin{aligned}
F_0 &= \left(f_0\omega_n{}^0 + f_2\omega_n{}^0\right) + \omega_n{}^0\left(f_1\omega_n{}^0 + f_3\omega_n{}^0\right)\\
F_1 &= \left(f_0\omega_n{}^0 + f_2\omega_n{}^2\right) + \omega_n{}^1\left(f_1\omega_n{}^0 + f_3\omega_n{}^2\right)\\
F_2 &= \left(f_0\omega_n{}^0 + f_2\omega_n{}^0\right) + \omega_n{}^2\left(f_1\omega_n{}^0 + f_3\omega_n{}^0\right)\\
F_3 &= \left(f_0\omega_n{}^0 + f_2\omega_n{}^2\right) + \omega_n{}^3\left(f_1\omega_n{}^0 + f_3\omega_n{}^2\right)
\end{aligned}
$$

Example (n = 4)

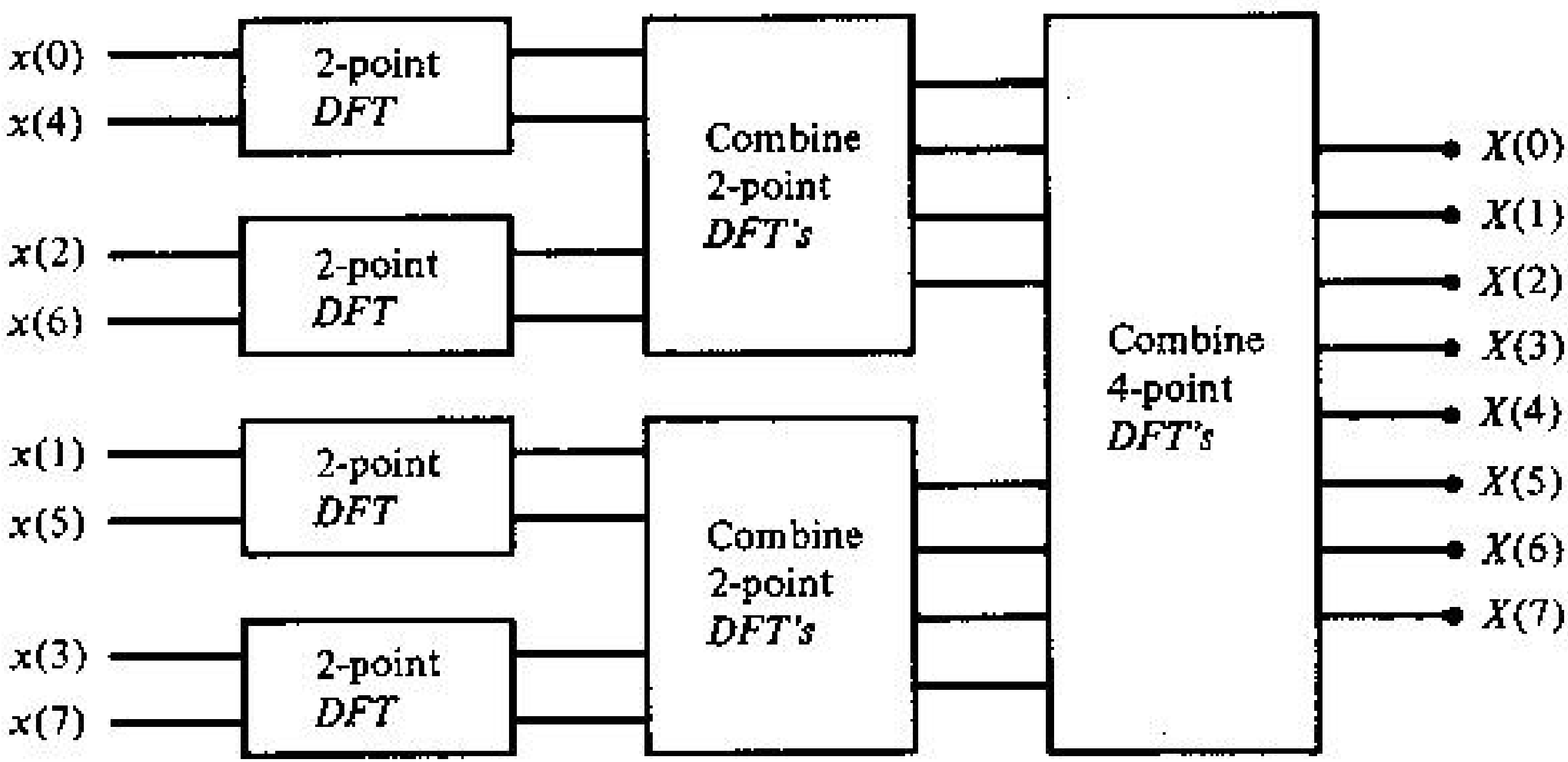
- This can be computed from two length-2 FFTs, with some "twiddle factors"

$$F_0 = \left(f_0\omega_n^{\ 0} + f_2\omega_n^{\ 0}\right) + \omega_n^{\ 0}\left(f_1\omega_n^{\ 0} + f_3\omega_n^{\ 0}\right)$$
$$F_1 = \left(f_0\omega_n^{\ 0} + f_2\omega_n^{\ 2}\right) + \omega_n^{\ 1}\left(f_1\omega_n^{\ 0} + f_3\omega_n^{\ 2}\right)$$
$$F_2 = \left(f_0\omega_n^{\ 0} + f_2\omega_n^{\ 0}\right) + \omega_n^{\ 2}\left(f_1\omega_n^{\ 0} + f_3\omega_n^{\ 0}\right)$$
$$F_3 = \left(f_0\omega_n^{\ 0} + f_2\omega_n^{\ 2}\right) + \omega_n^{\ 3}\left(f_1\omega_n^{\ 0} + f_3\omega_n^{\ 2}\right)$$

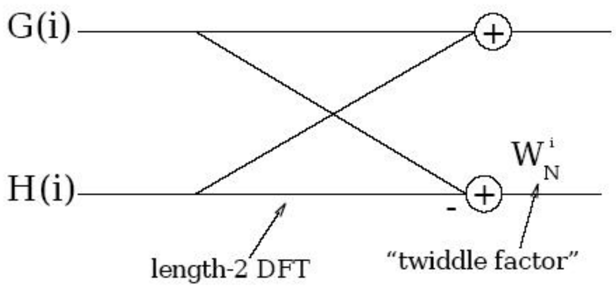
$$\begin{matrix} f_0 \\ f_2 \end{matrix} \xrightarrow{FFT} \begin{matrix} f_0\omega_{n/2}^{\ 0} + f_2\omega_{n/2}^{\ 0} \\ f_0\omega_{n/2}^{\ 0} + f_2\omega_{n/2}^{\ 1} \end{matrix} \qquad \begin{matrix} f_1 \\ f_3 \end{matrix} \xrightarrow{FFT} \begin{matrix} f_1\omega_{n/2}^{\ 0} + f_3\omega_{n/2}^{\ 0} \\ f_1\omega_{n/2}^{\ 0} + f_3\omega_{n/2}^{\ 1} \end{matrix}$$

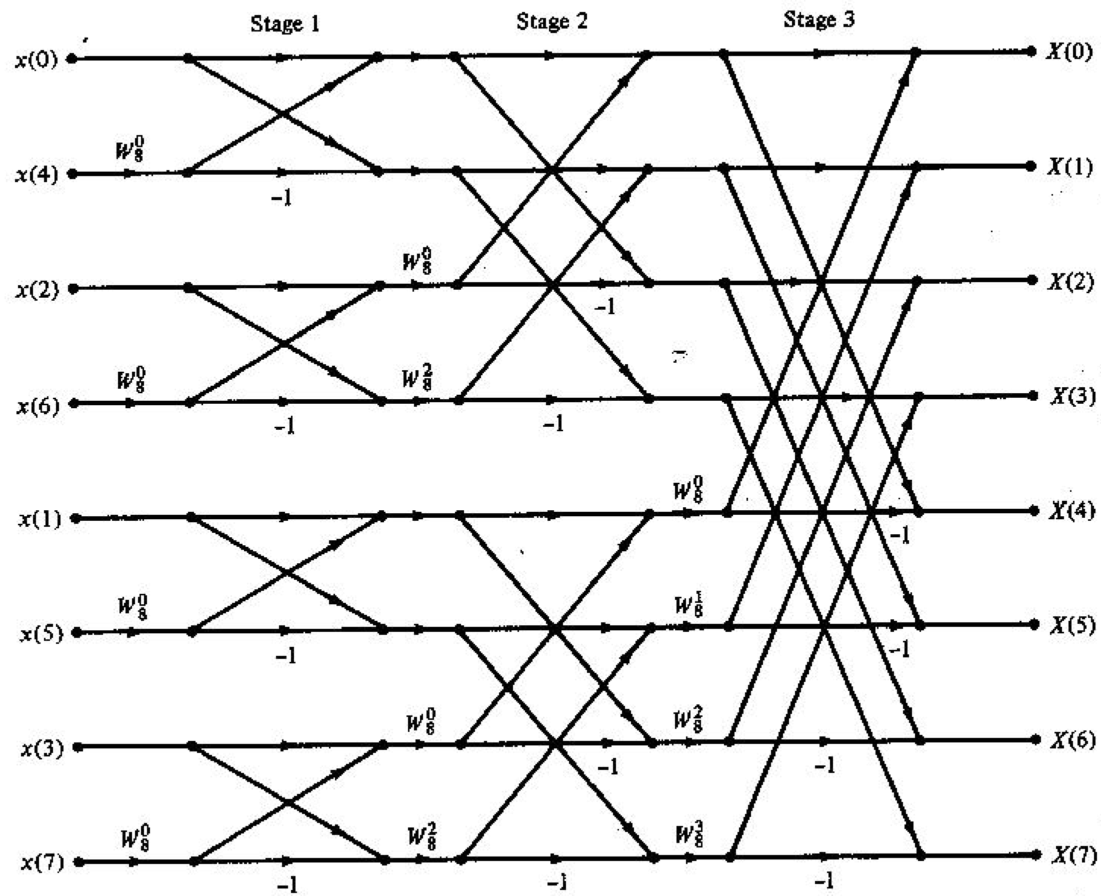
The FFT

- Now apply algorithm recursively!



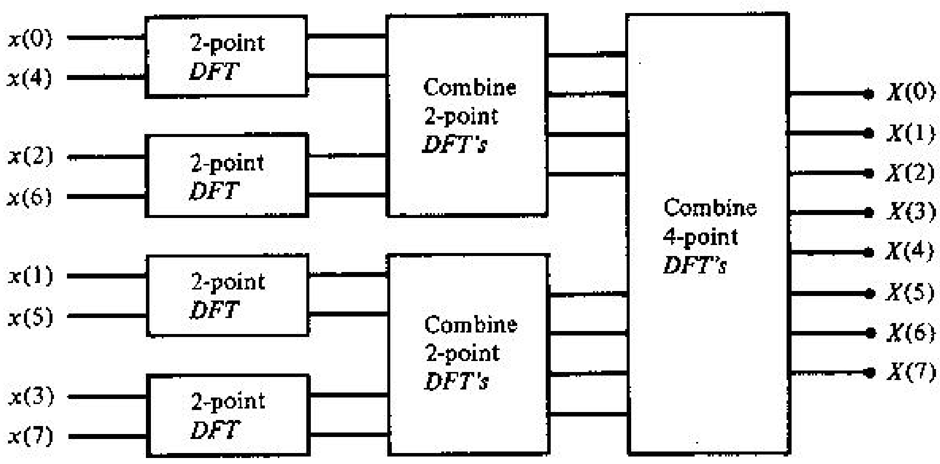
FFT Butterfly





The FFT

- Final detail: how to find elements involved in initial size-2 FFTs?



- Bit reversal!

$$
\begin{aligned}
0 &\to 000 \to 000 \to 0\\
1 &\to 001 \to 100 \to 4\\
2 &\to 010 \to 010 \to 2\\
3 &\to 011 \to 110 \to 6\\
4 &\to 100 \to 001 \to 1\\
5 &\to 101 \to 101 \to 5\\
6 &\to 110 \to 011 \to 3\\
7 &\to 111 \to 111 \to 7
\end{aligned}
$$

FFT Running Time

- Time to compute FFT of length *n*:
  - Solve two subproblems of length *n*/2
  - Additional processing proportional to *n*

$$T(n) = 2T(n/2) + cn$$

- Recurrence relation with solution

$$T(n) = c\,n\log_2 n$$

FFT Running Time

- Proof:

$$T(n) = 2T(n/2) + cn$$

$$c\,n\log_2 n \stackrel{?}{=} 2\left(c\,\tfrac{n}{2}\log_2 \tfrac{n}{2}\right) + cn$$

$$c\,n\log_2 n \stackrel{?}{=} c\,n\left((\log_2 n) - 1\right) + cn$$

$$c\,n\log_2 n \stackrel{\checkmark}{=} c\,n\log_2 n - cn + cn$$

DFT of Real Signals

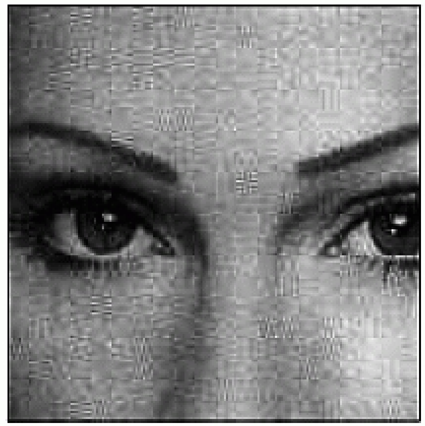
- Standard FFT is complex $\rightarrow$ complex
  - n real numbers as input yields n complex numbers
  - But: symmetry relation for real inputs $F_{n-k} = (F_k)^*$
  - Variants of FFT to compute this efficiently
- Discrete Cosine Transform (DCT)
  - Reflect real input to get signal of length $2n$
  - Resulting FFT real and symmetric
  - n real numbers as input, n real numbers as output

Application: JPEG Image Compression

- Perceptually-based lossy compression of images
- Algorithm
  - Transform colors
  - Divide into 8×8 blocks
  - 2-dimensional DCT on each block
  - Perceptually-guided quantization
  - Lossless run-length and Huffman encoding

Application: JPEG Image Compression

a. Original image



b. With 10:1 compression

c. With 45:1 compression

FIGURE 27-15
Example of JPEG distortion. Figure (a) shows the original image, while (b) and (c) shows restored images using compression ratios of 10:1 and 45:1, respectively. The high compression ratio used in (c) results in each 8×8 pixel group being represented by less than 12 bits.

Discrete Cosine Transform (DCT)

[Steven W. Smith 1997]

Application: Polynomial Multiplication

- Usual algorithm for multiplying two polynomials of degree *n* is O(n^2)
- Observation: can use DFT to efficiently go between polynomial coefficients f_x

$$f(t) = \sum_{x=0}^{n-1} f_x \, t^x$$

and polynomial evaluated at $\omega_n{}^k$

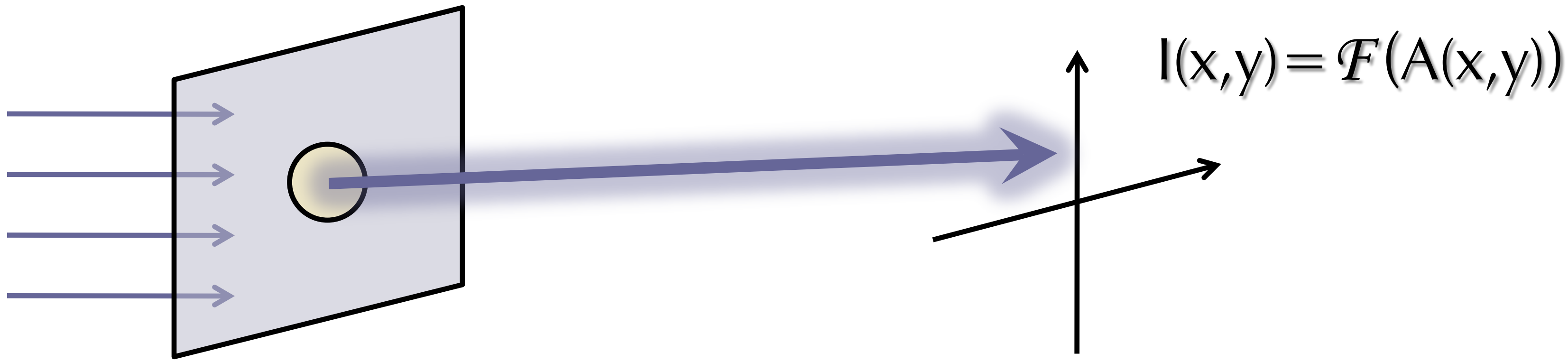
$$f(\omega_n{}^k) = F_k = \sum_{x=0}^{n-1} f_x \, \omega_n{}^{kx}$$

Application: Polynomial Multiplication

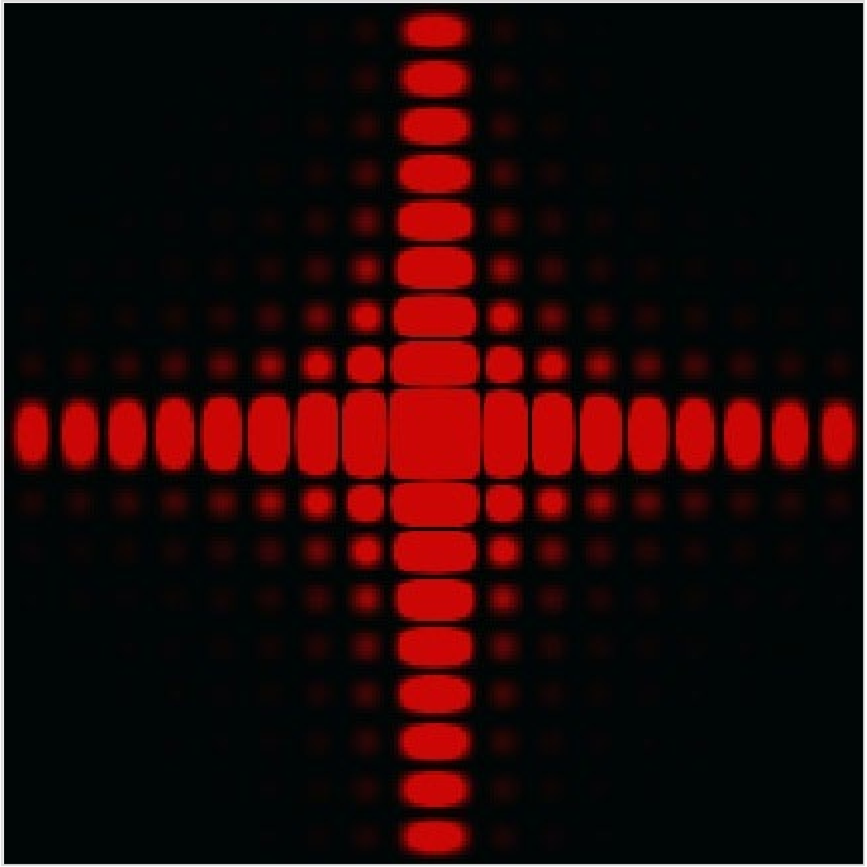
- So, we have an O(n log n) algorithm for multiplying two degree-n polynomials:
  - DFT on coefficients
  - Multiply
  - Inverse DFT
- Polynomial multiplication is convolution!

Application: Diffraction

- (Far-field) diffraction pattern of parallel light passing through an aperture is Fourier transform of aperture

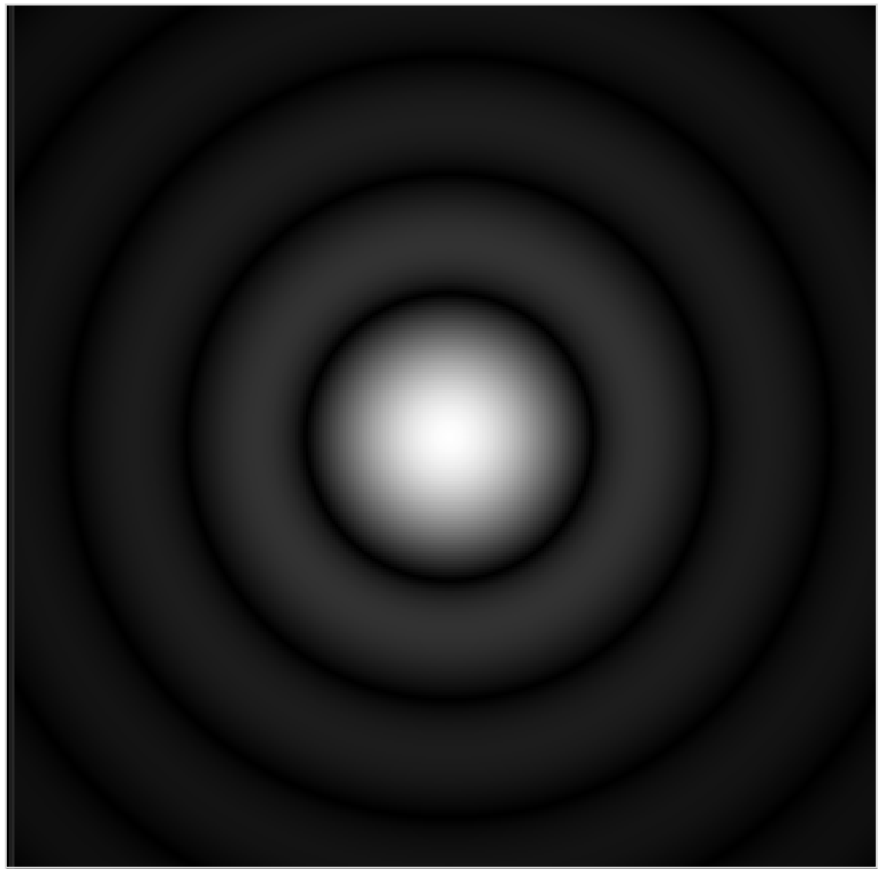


Application: Diffraction



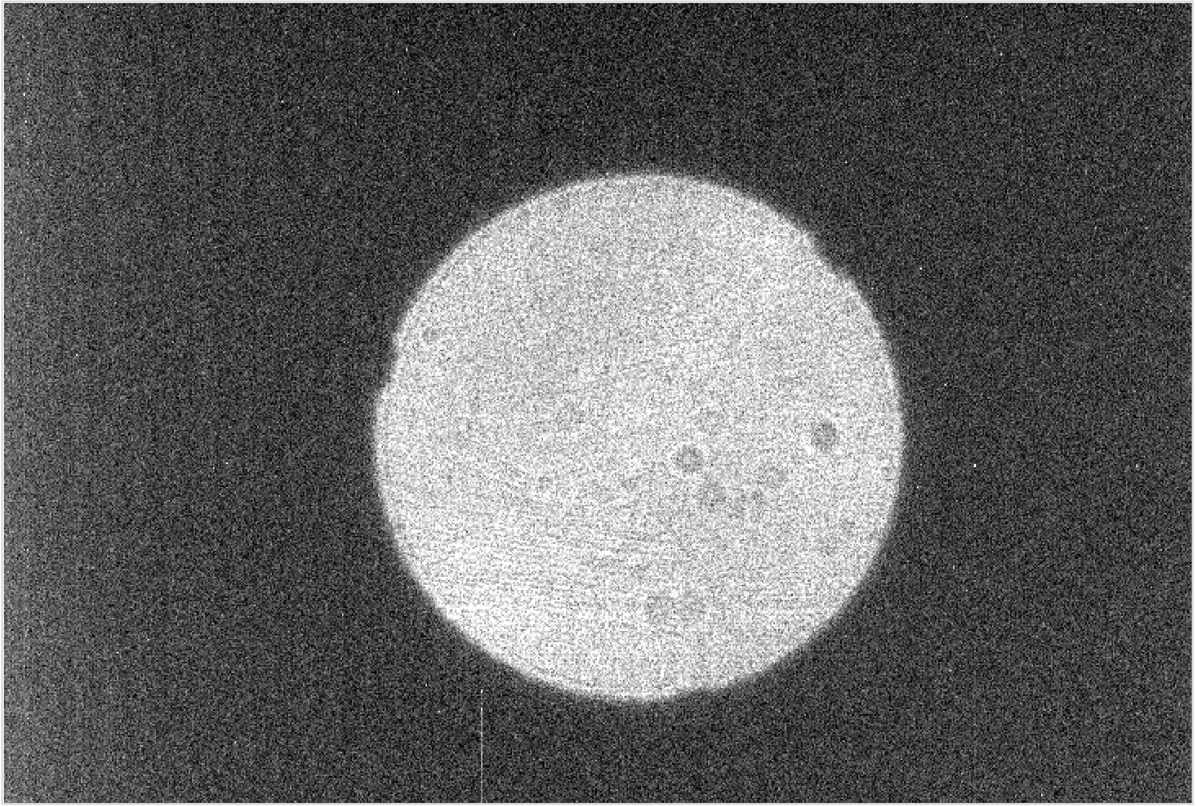
Square aperture

Application: Diffraction



Circular aperture: Airy disk

Application: Diffraction



Diffraction + defocus in telescope image