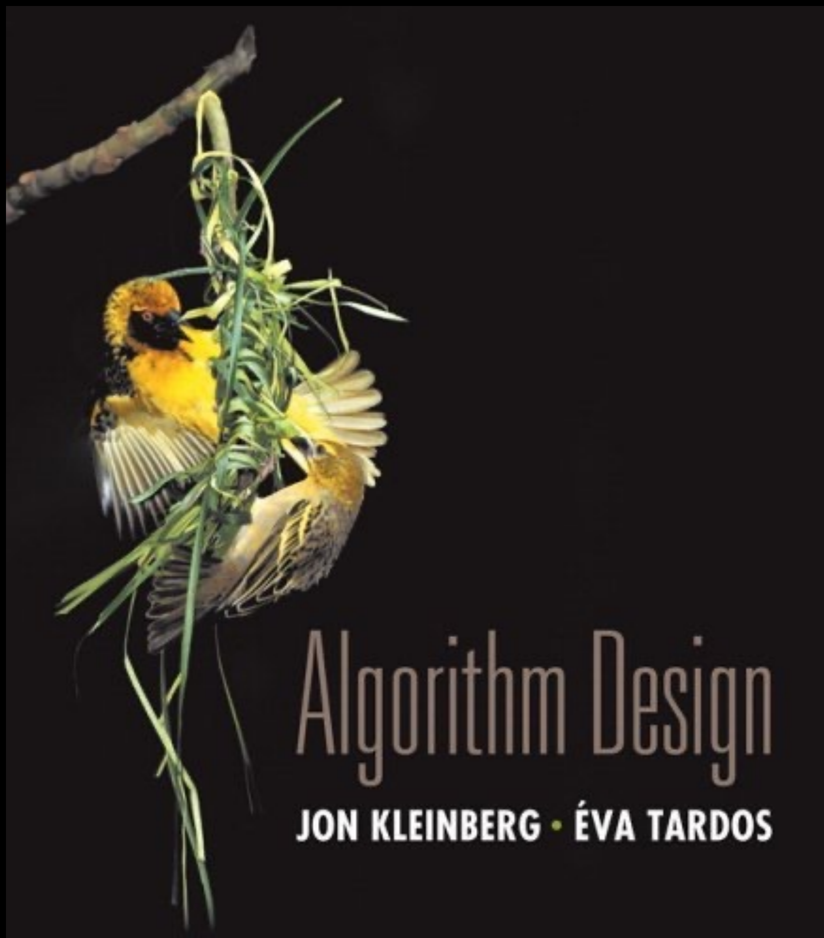


Chapter 5

Divide and Conquer

Slides by Kevin Wayne.

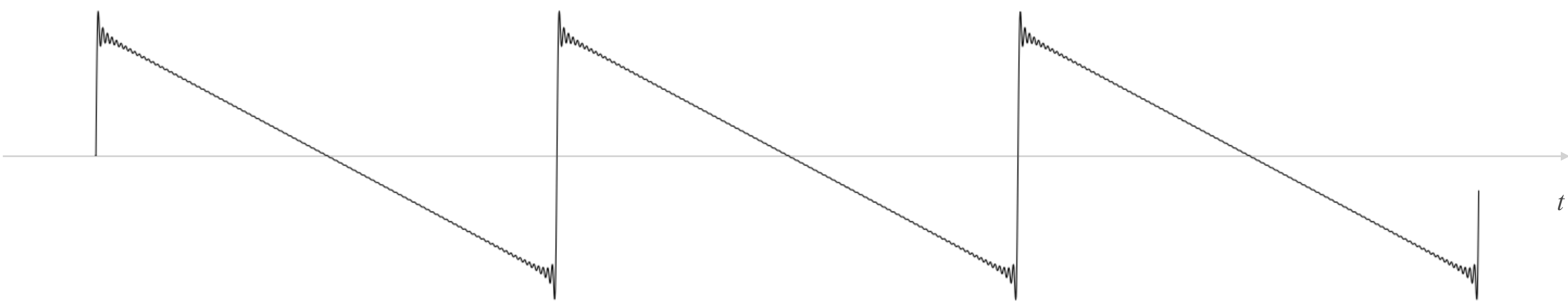
Annotated and slightly changed by Yang Xu (徐炀)
Contact: xuyang@sustech.edu.cn

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



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

Euler's Identity

Sinusoids. Sum of sine an cosines.

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

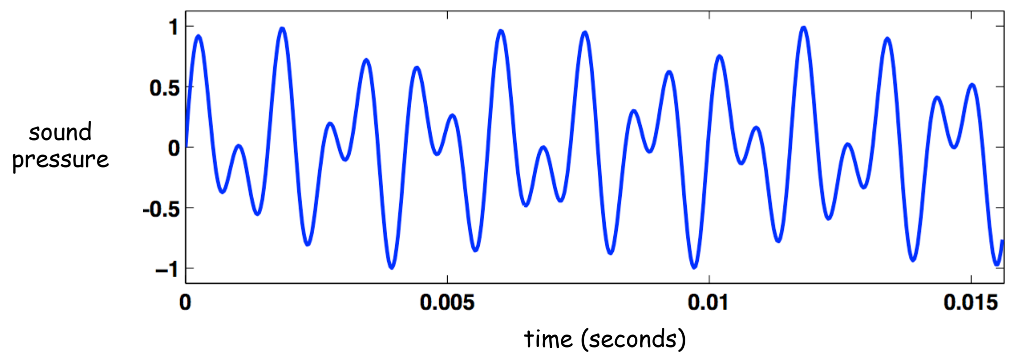
Euler's identity

Sinusoids. Sum of complex exponentials.

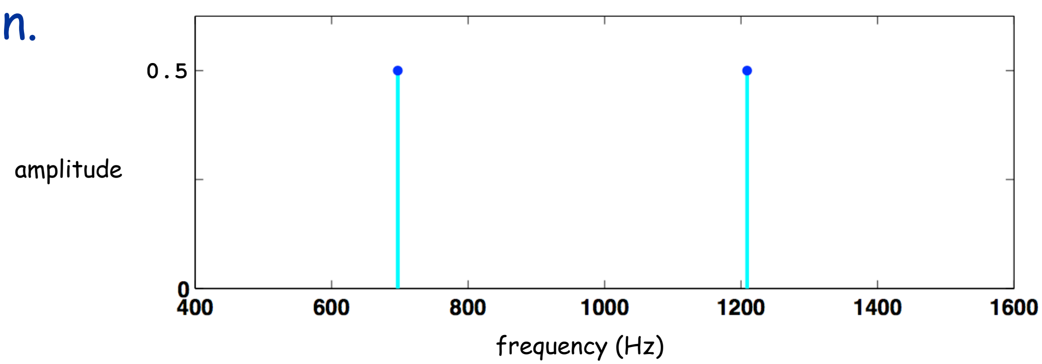
Time Domain vs. Frequency Domain

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.



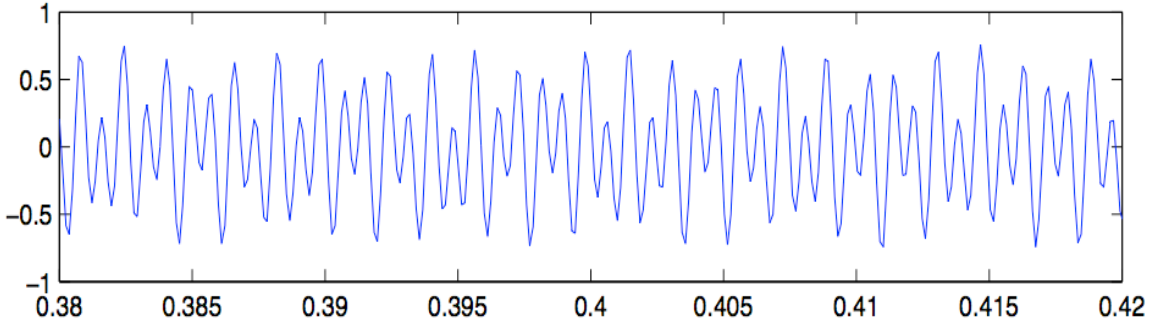
Frequency domain.



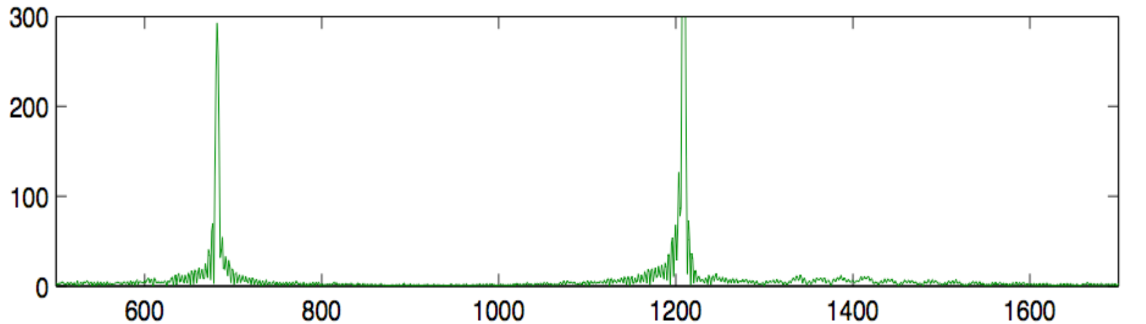
Reference: Cleve Moler, Numerical Computing with MATLAB

Time Domain vs. Frequency Domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

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

Evaluate: v. 求值；求……的数值；词根：value

> 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

Fourier (1822). Any function can be expanded into sine series.

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

Runge-König (1924). Laid theoretical groundwork.

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

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

系数

Polynomial. [coefficient representation]

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

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

Add. $O(n)$ arithmetic operations.

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

Evaluate. $O(n)$ using Horner's method.

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

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

卷积

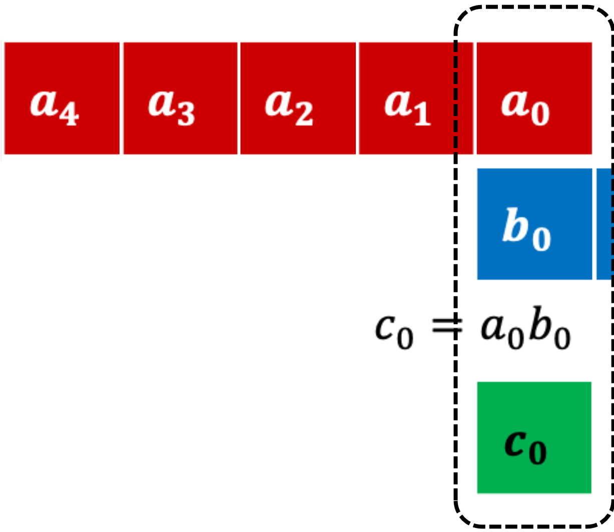
$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

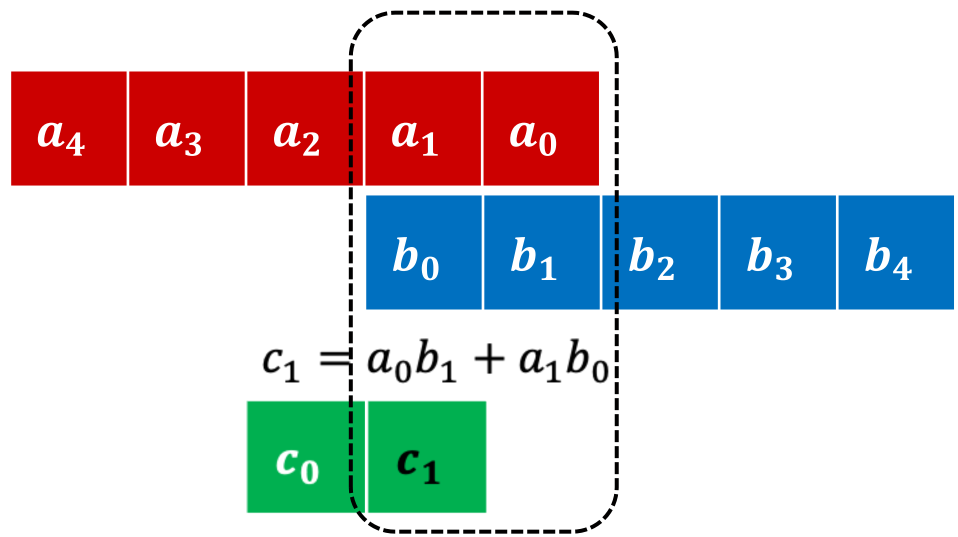
$$\begin{cases} i = 0,\ c_0 = a_0 b_0 \\ i = 1,\ c_1 = a_0 b_1 + a_1 b_0 \\ i = 2,\ c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0 \end{cases}$$

Notes on Convolve

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

$$c_k = \sum_{(i,j): i+j=k} a_i b_j$$

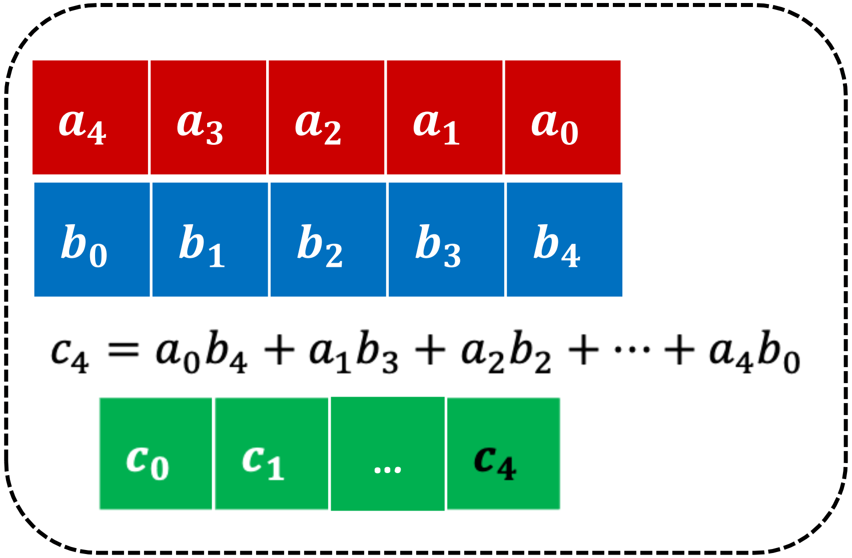


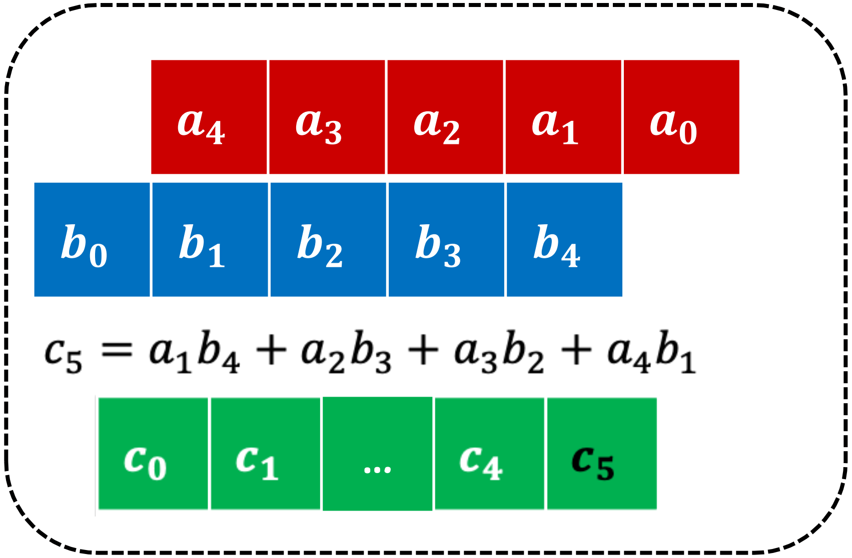


Notes on Convolve

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

$$c_k = \sum_{(i,j):i+j=k} a_i b_j$$





Convolve and Polynomials

Convolution between two vectors a and b: $a * b$, with $2n-1$ coordinates

$$a * b = (a_0b_0, a_0b_1 + a_1b_0, a_0b_2 + a_1b_1 + a_2b_0, \ldots, a_{n-2}b_{n-1} + a_{n-1}b_{n-2}, a_{n-1}b_{n-1})$$

$$\begin{array}{ccccc}
a_0b_0 & a_0b_1 & \cdots & a_0b_{n-2} & a_0b_{n-1} \\
a_1b_0 & a_1b_1 & \cdots & a_1b_{n-2} & a_1b_{n-1} \\
a_2b_0 & a_2b_1 & \cdots & a_2b_{n-2} & a_2b_{n-1} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
a_{n-1}b_0 & a_{n-1}b_1 & \cdots & a_{n-1}b_{n-2} & a_{n-1}b_{n-1}
\end{array}$$

Summing along $2n-1$ diagnals

When a and b are of unequal lengths

$$a = (a_0, a_1, \dots, a_{m-1})$$

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

A vector of $m + n - 1$ coordinates

$$a * b = (a_0 b_0, a_0 b_1 + a_1 b_0, \dots, a_{m-1} b_{n-1})$$

$$m \left\{ \underbrace{\begin{array}{ccccc}
a_0 b_0 & a_0 b_1 & \cdots & a_0 b_{n-2} & a_0 b_{n-1} \\
a_1 b_0 & a_1 b_1 & \cdots & a_1 b_{n-2} & a_1 b_{n-1} \\
a_2 b_0 & a_2 b_1 & \cdots & a_2 b_{n-2} & a_2 b_{n-1} \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
a_{m-1} b_0 & a_{m-1} b_1 & \cdots & a_{m-1} b_{n-2} & a_{m-1} b_{n-1}
\end{array}}_{n} \right.$$

Why convolve? Relates to polynomials

Polynomial multiplication

Given $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1}$ and $B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$

$$C(x) = A(x)B(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{m+n-2} x^{m+n-2}$$

$$c_k = \sum_{(i,j):i+j=k} a_i b_j \qquad (k = 0,1, \ldots m+n-2)$$

The coefficient vector c of $C(x)$ is the convolution of the coefficient vectors of $A(x)$ and $B(x)$

The computation of c_k needs $O(n^2)$ multiplications of $a_i b_j$

Question: Is there an efficient way to obtain the coefficients c_k?

A Modest PhD Dissertation Title

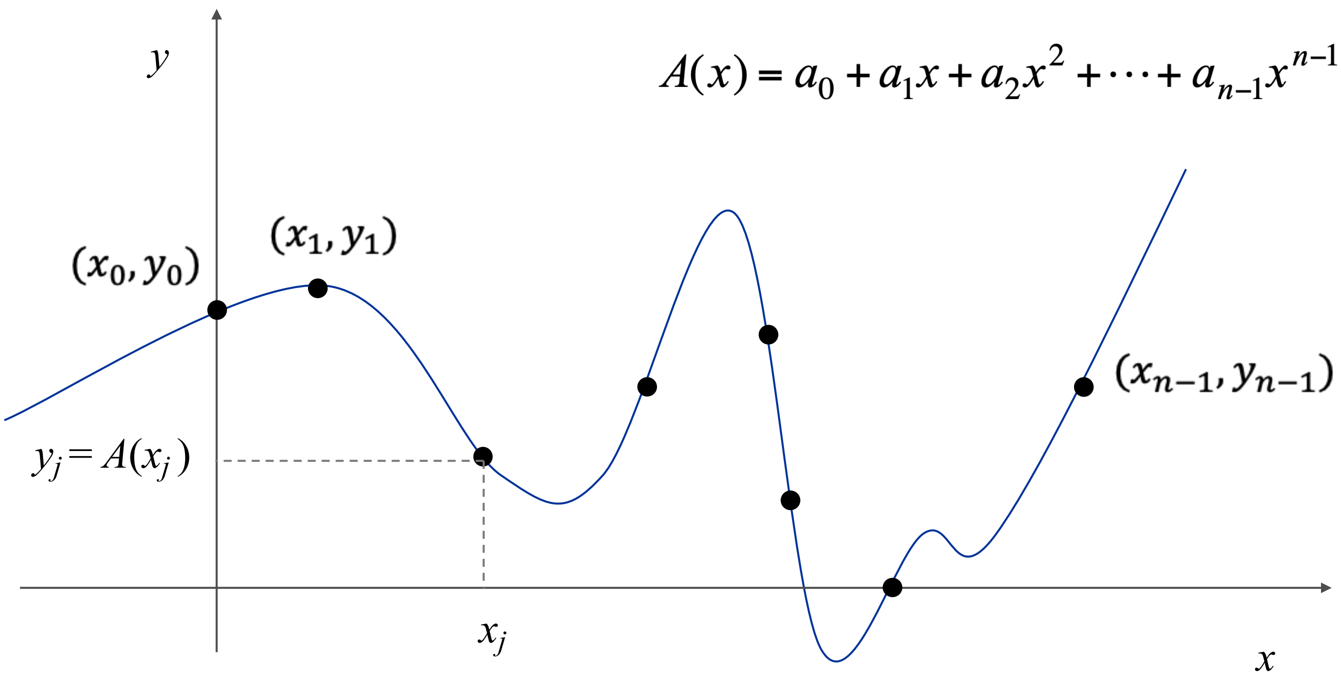
"New Proof of the Theorem That Every Algebraic Rational Integral Function In One Variable can be Resolved into Real Factors of the First or the Second Degree."

- 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

点值

Polynomial. [point-value representation]

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

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

Add. $O(n)$ arithmetic operations.

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

Multiply (convolve). $O(n)$, but need $2n\text{-}1$ points.

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

$$C(x) = A(x) \times B(x)$$

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i$$

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.

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

coefficient representation

point-value representation

Converting Between Two Representations: Brute Force

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

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

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

Converting Between Two Representations: Brute Force

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

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

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 $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

we get to choose which ones!

Divide. Break polynomial up into even and odd powers.

- $A(x) \quad = \ a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2).$

Intuition. Choose two points to be ± 1.

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

2 distinct points evaluated:
$x_0 = 1, x_1 = -1$

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 $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

we get to choose which ones!

Divide. Break polynomial up into even and odd powers.

- $A(x) \quad = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
- $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2).$

4 distinct points evaluated:
$x_0 = 1, x_1 = i, x_2 = -1, x_3 = -i,$

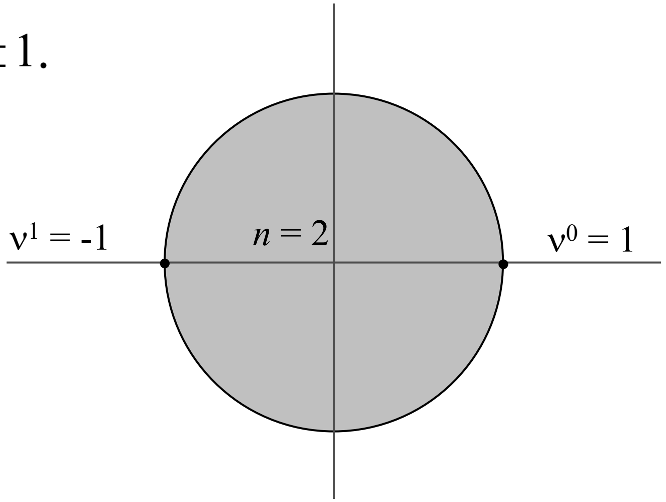
Intuition. Choose four complex points to be $\pm 1, \pm i$.

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

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

What are the distinct points like?

Intuition 1. Choose two points to be ± 1.

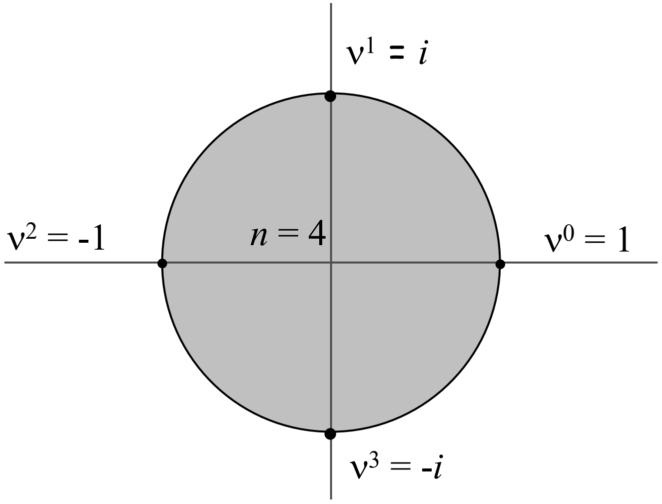


Intuition 2. Choose four complex points to be ± 1, $\pm i$.

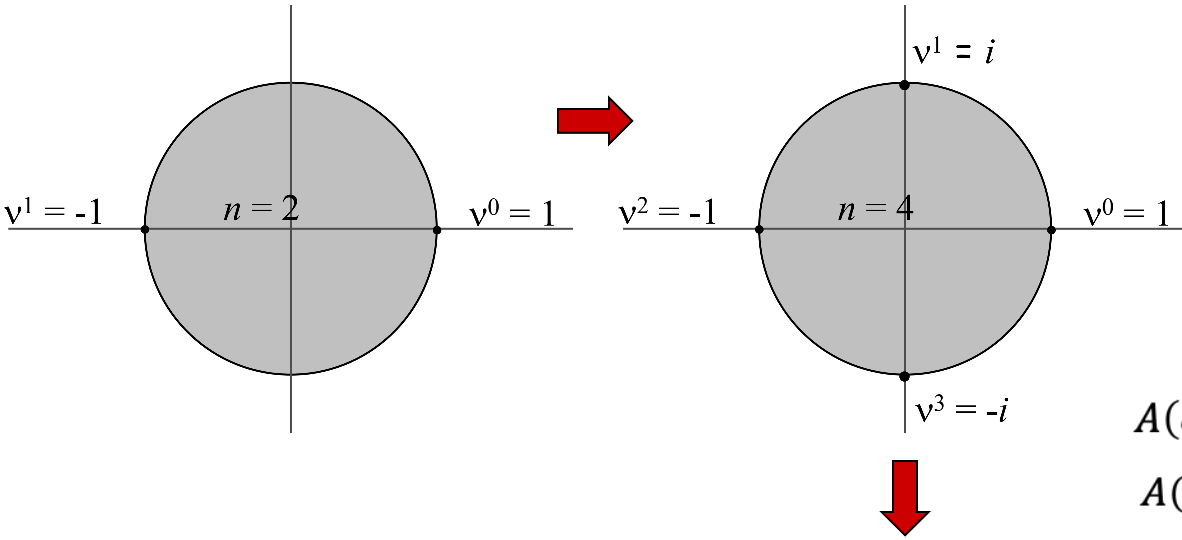
Question: How many points do we need?

Recall the goal: To find the point-value representation of $A(x)$, who is of degree $n-1$.

Thus, n points needed



How to find the n points?

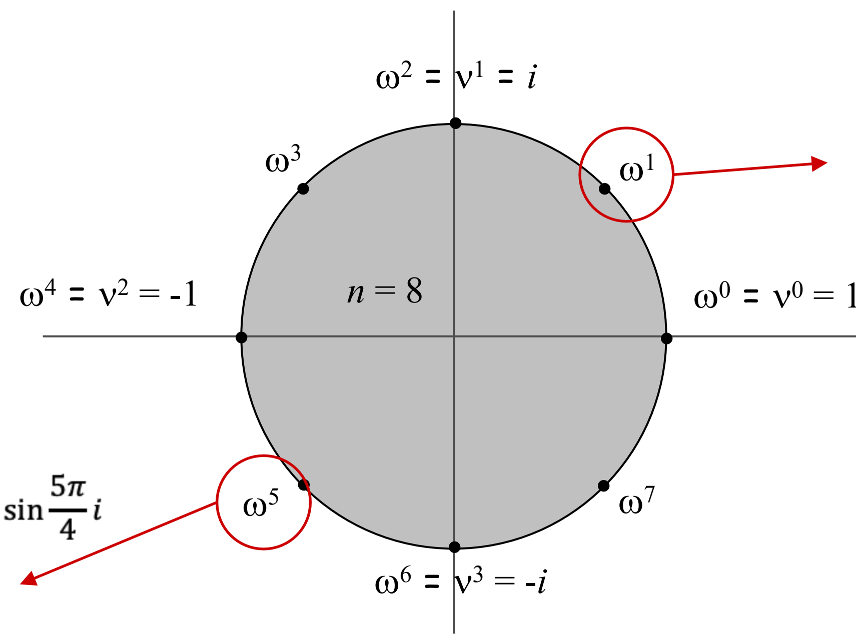


Substitute into

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

$$A(\omega^1) = A_{even}\left((\omega^1)^2\right) + \omega^1 A_{odd}\left((\omega^1)^2\right)$$

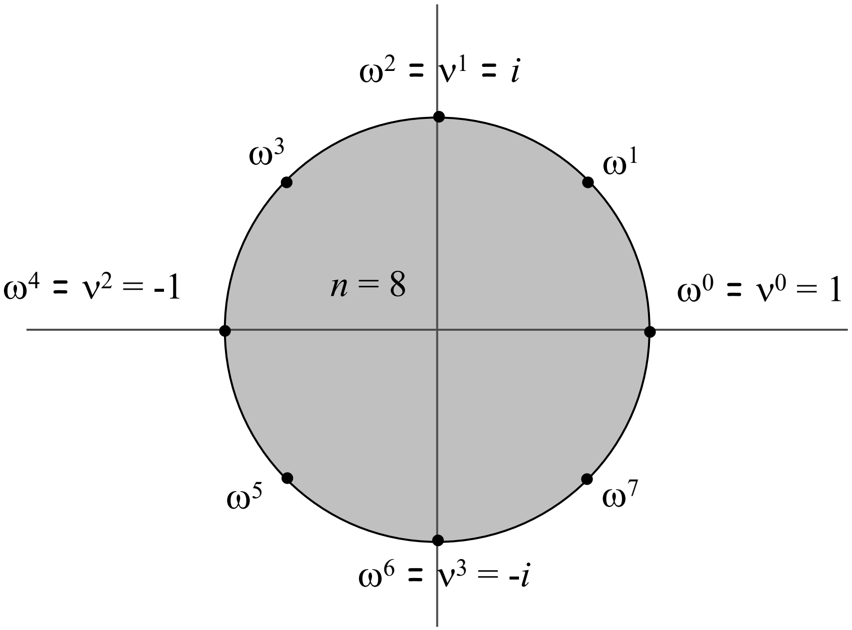
$$A(\omega^5) = A_{even}\left((\omega^1)^2\right) - \omega^1 A_{odd}\left((\omega^1)^2\right)$$



$$\omega^1 = e^{2\pi i \frac{1}{8}} = e^{\pi i \frac{1}{4}} = \cos\frac{\pi}{4} + \sin\frac{\pi}{4} i$$
$$= \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} i$$

$$\omega^5 = e^{2\pi i \frac{5}{8}} = e^{\pi i \frac{5}{4}} = \cos\frac{5\pi}{4} + \sin\frac{5\pi}{4} i$$
$$= -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2} i = -\omega^1$$

Find n=8 points



$$A(\omega^0) = A_{even}\left((\omega^0)^2\right) + \omega^0 A_{odd}\left((\omega^0)^2\right)$$
$$A(\omega^4) = A_{even}\left((\omega^0)^2\right) - \omega^0 A_{odd}\left((\omega^0)^2\right)$$

$$A(\omega^1) = A_{even}\left((\omega^1)^2\right) + \omega^1 A_{odd}\left((\omega^1)^2\right)$$
$$A(\omega^5) = A_{even}\left((\omega^1)^2\right) - \omega^1 A_{odd}\left((\omega^1)^2\right)$$

$$A(\omega^2) = A_{even}\left((\omega^2)^2\right) + \omega^1 A_{odd}\left((\omega^2)^2\right)$$
$$A(\omega^6) = A_{even}\left((\omega^2)^2\right) - \omega^1 A_{odd}\left((\omega^2)^2\right)$$

$$A(\omega^3) = A_{even}\left((\omega^3)^2\right) + \omega^1 A_{odd}\left((\omega^3)^2\right)$$
$$A(\omega^7) = A_{even}\left((\omega^3)^2\right) - \omega^1 A_{odd}\left((\omega^3)^2\right)$$

Can evaluate polynomial of degree $\leq n$ at 8 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 4 points.

Roots of Unity

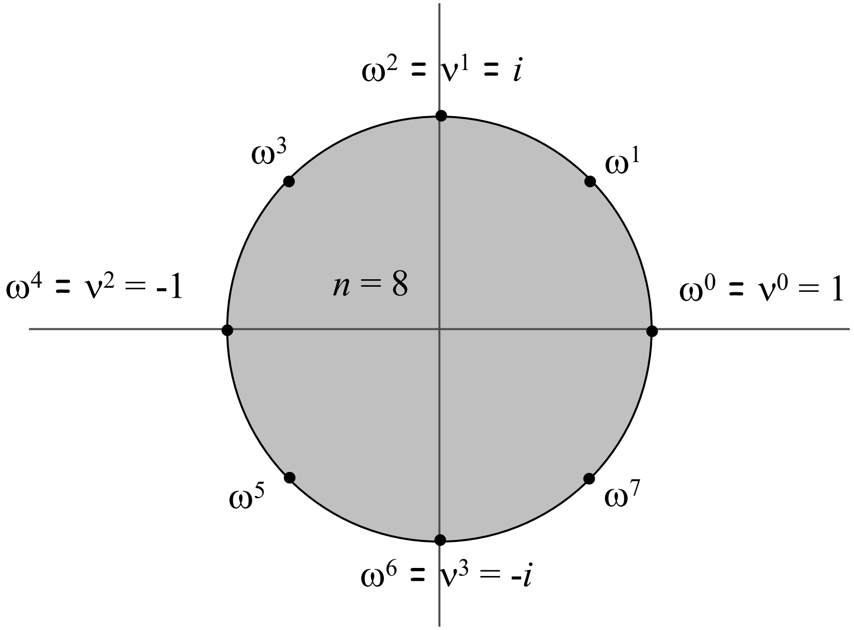
Def. An n^{th} **root of unity** is a complex number x such that $x^n = 1$.

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

$$\omega^k = e^{2\pi i \frac{k}{n}}$$

$$(k = 0, 1, \ldots, n-1)$$



Discrete Fourier Transform

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

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

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

DFT

Fourier matrix F_n

Entry at (k,j)

$$\omega^{kj} = \left(\omega^k\right)^j = \left(e^{2\pi i \frac{k}{n}}\right)^j$$
$$= e^{2\pi i \frac{kj}{n}}$$

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

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

Divide. Break up polynomial into even and odd powers.

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

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

Combine.

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

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

$\nu^k = (\omega^{k+\frac{1}{2}n})^2$ $\qquad$ $\omega^{k+\frac{1}{2}n} = -\omega^k$

Fast Fourier Transform - recursive

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

Output $\qquad$ Input

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{bmatrix} \longleftarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

RECURSIVE-FFT(a)

```
n = a.length
if n == 1
    return a
ω_n = e^{2πi/n}
ω = 1
a^[0] = (a_0, a_2, ..., a_{n-2})
a^[1] = (a_1, a_3, ..., a_{n-1})
y^[0] = RECURSIVE-FFT(a^[0])
y^[1] = RECURSIVE-FFT(a^[1])
for k = 0 to n/2 - 1
    y_k = y_k^[0] + ω y_k^[1]
    y_{k+(n/2)} = y_k^[0] - ω y_k^[1]
    ω = ω ω_n
return y
```

Divide: $A(x) = A_{even}(x^2) + x\, A_{odd}(x^2)$.
Let $A^{[0]} = A_{even}$ and $A^{[1]} = A_{odd}$ for short

Conquer: evaluate $A^{[0]}$ and $A^{[1]}$ at
$(\omega^0)^2, (\omega^1)^2, \ldots, (\omega^{n-1})^2$

Only n/2 distinct values

FFT Algorithm - alternative

```
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 ← e^(2πik/n)
      yₖ     ← eₖ + ωᵏ dₖ
      yₖ₊ₙ/₂ ← eₖ - ωᵏ dₖ
   }

   return (y₀,y₁,…,yₙ₋₁)
}
```

Fast Fourier Transform - analysis

RECURSIVE-FFT(a)

```
1   n = a.length
2   if n == 1
3       return a
4   ω_n = e^{2πi/n}
5   ω = 1
6   a^[0] = (a_0, a_2, ..., a_{n-2})
7   a^[1] = (a_1, a_3, ..., a_{n-1})
8   y^[0] = RECURSIVE-FFT(a^[0])
9   y^[1] = RECURSIVE-FFT(a^[1])
10  for k = 0 to n/2 - 1
11      y_k = y_k^[0] + ω y_k^[1]
12      y_{k+(n/2)} = y_k^[0] - ω y_k^[1]
13      ω = ω ω_n
14  return y
```

Lines 2–3 represent the basis of the recursion;

$$
\begin{aligned}
y_0 &= a_0\,\omega_1^0 \\
&= a_0 \cdot 1 \\
&= a_0 \,.
\end{aligned}
$$

Line 13: ω is updated properly
Keeping a running value of ω from iteration to iteration saves time over computing ω^k from scratch

Line 8-9: recursive $DFT_{n/2}$

$$
\begin{aligned}
y_k^{[0]} &= A^{[0]}(\omega_n^{2k}) \\
y_k^{[1]} &= A^{[1]}(\omega_n^{2k})
\end{aligned}
\qquad \Longleftrightarrow \qquad
\begin{aligned}
y_k^{[0]} &= A^{[0]}(\omega_{n/2}^{k}) \\
y_k^{[1]} &= A^{[1]}(\omega_{n/2}^{k})
\end{aligned}
$$

Fast Fourier Transform - analysis

RECURSIVE-FFT(a)

1 $n = a.length$
2 **if** $n == 1$
3 **return** a
4 $\omega_n = e^{2\pi i/n}$
5 $\omega = 1$
6 $a^{[0]} = (a_0, a_2, \ldots, a_{n-2})$
7 $a^{[1]} = (a_1, a_3, \ldots, a_{n-1})$
8 $y^{[0]} =$ RECURSIVE-FFT$(a^{[0]})$
9 $y^{[1]} =$ RECURSIVE-FFT$(a^{[1]})$
10 **for** $k = 0$ **to** $n/2 - 1$
11 $y_k = y_k^{[0]} + \omega\, y_k^{[1]}$
12 $y_{k+(n/2)} = y_k^{[0]} - \omega\, y_k^{[1]}$
13 $\omega = \omega\, \omega_n$
14 **return** y

Why minus?

Line 11-12:
combine the results of the recursive $DFT_{n/2}$ calculations

First half:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{\frac{n}{2}-1} \\ y_{\frac{n}{2}} \\ \vdots \\ y_{n-1} \end{bmatrix}$$

$$\begin{aligned} y_k &= y_k^{[0]} + \omega_n^k y_k^{[1]} \\ &= A^{[0]}(\omega_n^{2k}) + \omega_n^k A^{[1]}(\omega_n^{2k}) \\ &= A(\omega_n^k) \end{aligned}$$

$$\begin{aligned} y_{k+(n/2)} &= y_k^{[0]} - \omega_n^k y_k^{[1]} \\ &= y_k^{[0]} + \omega_n^{k+(n/2)} y_k^{[1]} \end{aligned}$$

Second half: $= A(\omega_n^{k+(n/2)})$

Use the fact: $\omega_n^{k+(n/2)} = -\omega_n^k$

ω_n^k is used in both positive and negative forms, called twiddle factor

FFT Summary

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

assumes n is a power of 2

Running time.

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

$$a_0, a_1, \ldots, a_{n\text{-}1} \; \underset{???}{\overset{O(n \log n)}{\rightleftarrows}} \; (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

Recall: Discrete Fourier Transform

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

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

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

Entry at (k,j)

$$\omega^{kj} = \left(\omega^k\right)^j = \left(e^{2\pi i \frac{k}{n}}\right)^j = e^{2\pi i \frac{kj}{n}}$$

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

coefficient representation

point-value representation

Now: Inverse Discrete Fourier Transform

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

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

Inverse DFT

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

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

coefficient representation

point-value representation

Inverse DFT

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

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

Each entry (k,j):

$$\omega^{-kj} = e^{-2\pi i\frac{kj}{n}}$$

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

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

Inverse FFT: Proof of Correctness

Claim. F_n and G_n are inverses.

Pf.

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

summation lemma

Summation lemma. Let ω be a principal n^{th} root of unity. Then

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

Pf.

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

Inverse FFT: Proof of Correctness

$$F_n$$

$$\begin{array}{c} \\ \\ \\ k \\ \\ \\ \end{array}
\begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}$$

$$G_n$$

$$\frac{1}{n}
\begin{array}{c}
\begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\
1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)}
\end{bmatrix} \\
\begin{array}{cccccc} & \phantom{\omega^{-(n-1)}} & \phantom{\omega^{-2(n-1)}} & k' & & \phantom{\omega^{-(n-1)(n-1)}} \end{array}
\end{array}$$

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

Thus, $F_n G_n = I_n$

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse FFT: Algorithm

```
ifft(n, a_0,a_1,…,a_{n-1}) {
   if (n == 1) return a_0

   (e_0,e_1,…,e_{n/2-1}) ← FFT(n/2, a_0,a_2,a_4,…,a_{n-2})
   (d_0,d_1,…,d_{n/2-1}) ← FFT(n/2, a_1,a_3,a_5,…,a_{n-1})

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

   return (y_0,y_1,…,y_{n-1})
}
```

Inverse FFT Summary

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

assumes n is a power of 2

$$a_0, a_1, \ldots, a_{n\text{-}1} \quad \underset{O(n \log n)}{\overset{O(n \log n)}{\rightleftarrows}} \quad (\omega^0, y_0), \ldots, (\omega^{n-1}, y_{n-1})$$

coefficient representation

point-value representation

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

coefficient representation

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

2 FFTs $O(n \log n)$

$A(\omega^0), \ldots, A(\omega^{2n-1})$
$B(\omega^0), \ldots, B(\omega^{2n-1})$

point-value multiplication $O(n)$

$C(\omega^0), \ldots, C(\omega^{2n-1})$

inverse FFT $O(n \log n)$

coefficient representation

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

Convolution theorem

MIT Book p. 914

For any two vectors a and b of length n, where n is the power of 2

$$a * b = DFT_{2n}^{-1}(DFT_{2n}(a) \cdot DFT_{2n}(b))$$

where the vectors a and b are padded with 0s to length $2n$.
"·"denotes point-wise product of two vectors.

Efficient FFT implementations

An iterative FFT Implementation

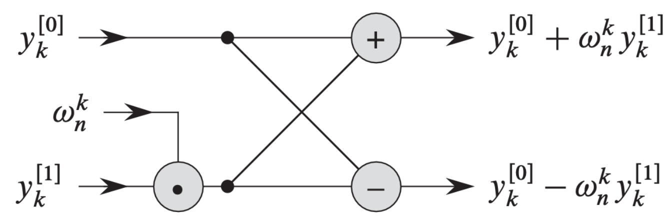
RECURSIVE-FFT(a)

$n = a.length$
if $n == 1$
    return a
$\omega_n = e^{2\pi i/n}$
$\omega = 1$
$a^{[0]} = (a_0, a_2, \ldots, a_{n-2})$
$a^{[1]} = (a_1, a_3, \ldots, a_{n-1})$
$y^{[0]} =$ RECURSIVE-FFT$(a^{[0]})$
$y^{[1]} =$ RECURSIVE-FFT$(a^{[1]})$
for $k = 0$ **to** $n/2 - 1$
    $y_k = y_k^{[0]} + \omega\, y_k^{[1]}$
    $y_{k+(n/2)} = y_k^{[0]} - \omega\, y_k^{[1]}$
    $\omega = \omega\, \omega_n$
return y

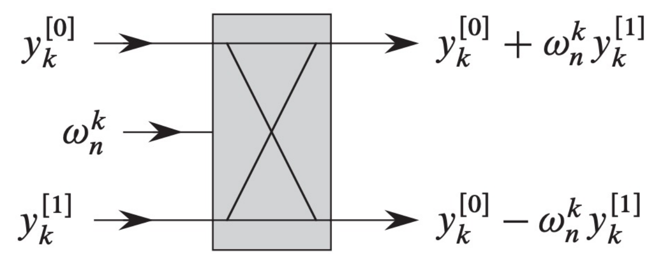
for $k = 0$ **to** $n/2 - 1$
    $t = \omega\, y_k^{[1]}$
    $y_k = y_k^{[0]} + t$
    $y_{k+(n/2)} = y_k^{[0]} - t$
    $\omega = \omega\, \omega_n$

$\omega y_k^{[1]}$ computed twice

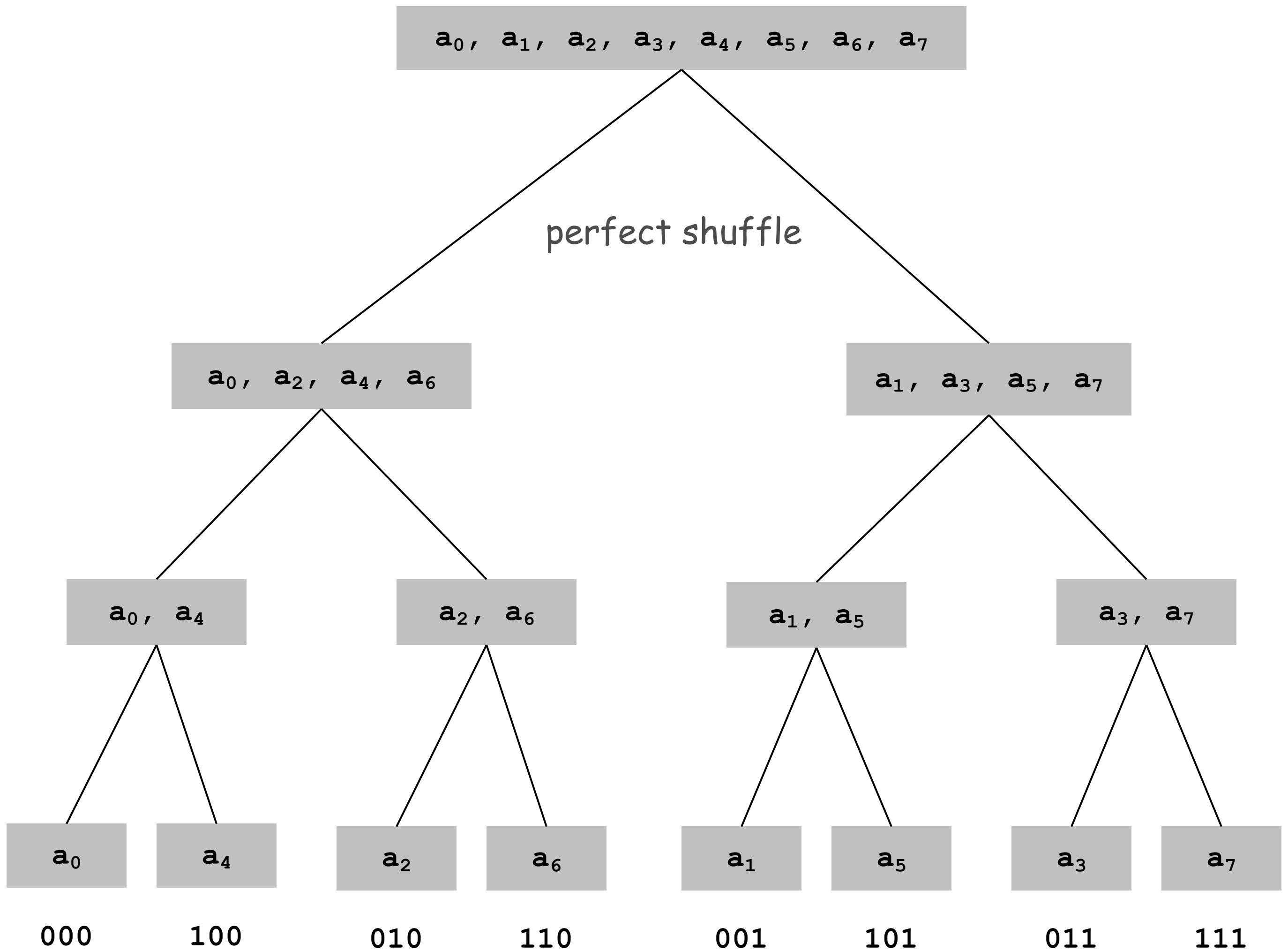
Butterfly op



or



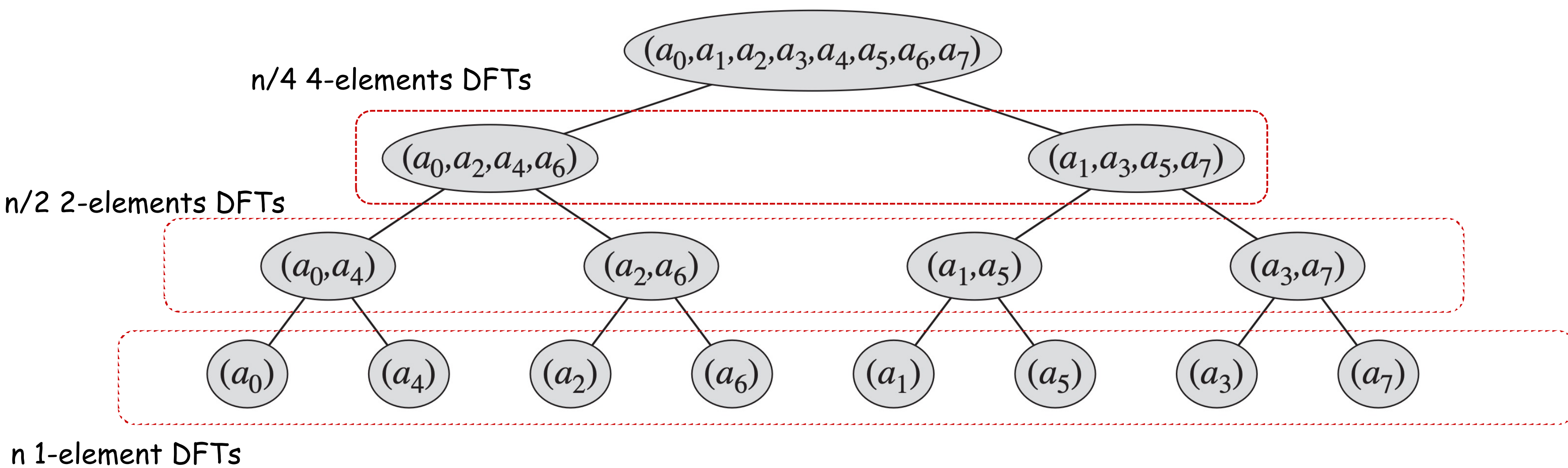
Observe the Recursion Tree



"bit-reversed" order

Trace the execution of RECURSIVE-FFT

If we could arrange the elements of the initial vector a into the order in which they appear in the leaves, we could trace the execution of the RECURSIVE-FFT procedure, but bottom up instead of top down.




```
for s = 1 to lg n
    for k = 0 to n − 1 by 2^s
        combine the two 2^(s−1)-element DFTs in
            A[k .. k + 2^(s−1) − 1] and A[k + 2^(s−1) .. k + 2^s − 1]
            into one 2^s-element DFT in A[k .. k + 2^s − 1]
```

MIT Book p. 917

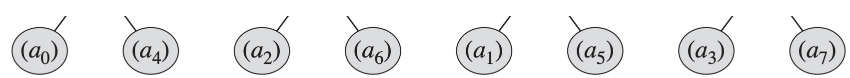
Iterative FFT

Copy the for loop from the RECURSIVE-FFT procedure
Compute the size of DFT for each level s: $m = 2^s$
First calls an auxiliary procedure to copy vector a into array A in the initial order in which we need the values

```
ITERATIVE-FFT(a)
BIT-REVERSE-COPY(a, A)
n = a.length                    // n is a power of 2
for s = 1 to lg n
    m = 2^s
    ω_m = e^{2πi/m}
    for k = 0 to n − 1 by m
        ω = 1
        for j = 0 to m/2 − 1
            t = ω A[k + j + m/2]
            u = A[k + j]
            A[k + j] = u + t
            A[k + j + m/2] = u − t
            ω = ω ω_m
return A
```

```
BIT-REVERSE-COPY(a, A)
n = a.length
for k = 0 to n − 1
    A[rev(k)] = a_k
```

let rev[k] be the lg n-bit integer formed by reversing the bits of the binary representation of k



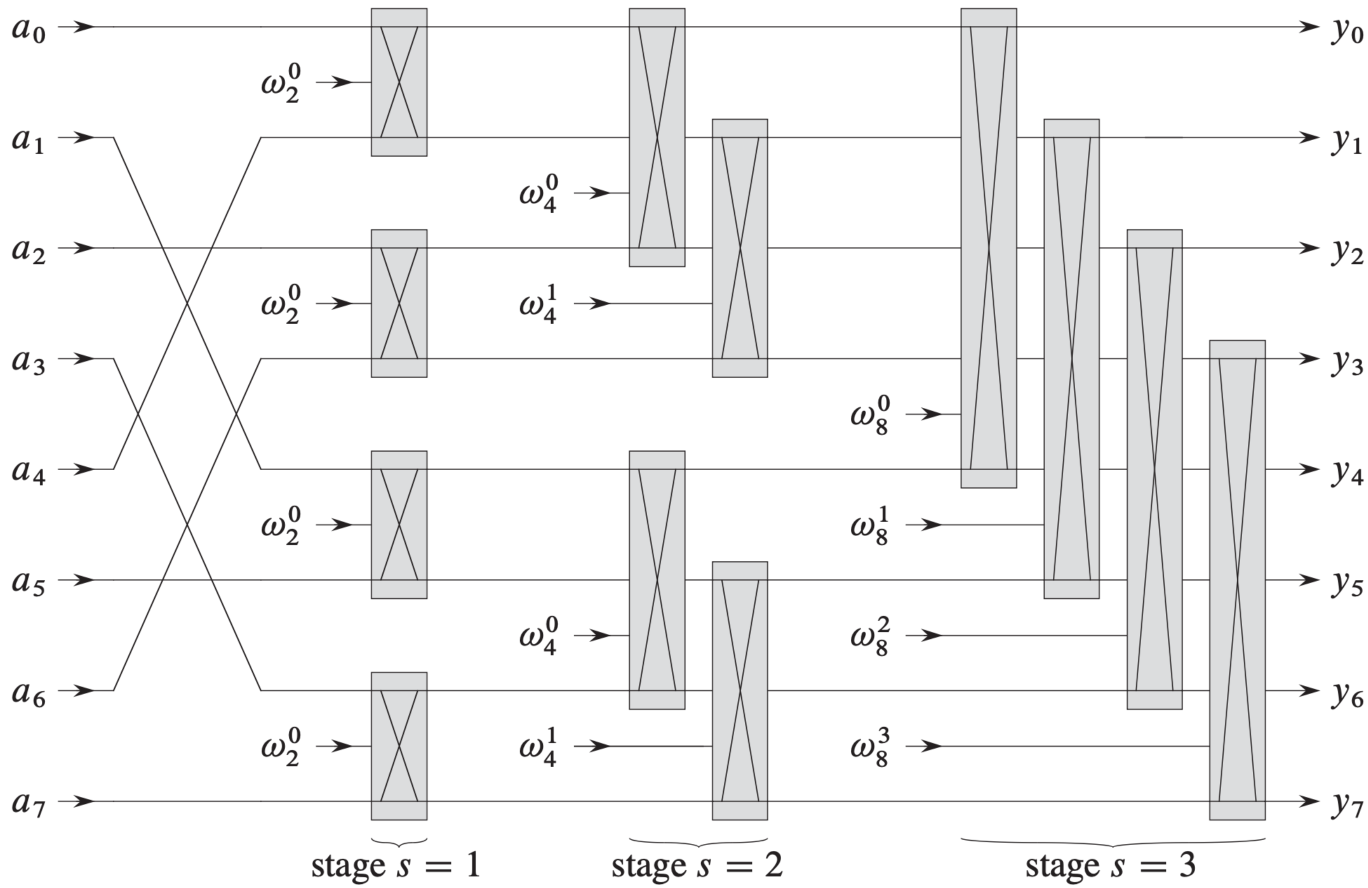
0;4;2;6;1;5;3;7

k: 000; 100; 010; 110; 001; 101; 011; 111

rev(k): 000; 001; 010; 011; 100; 101; 110; 111

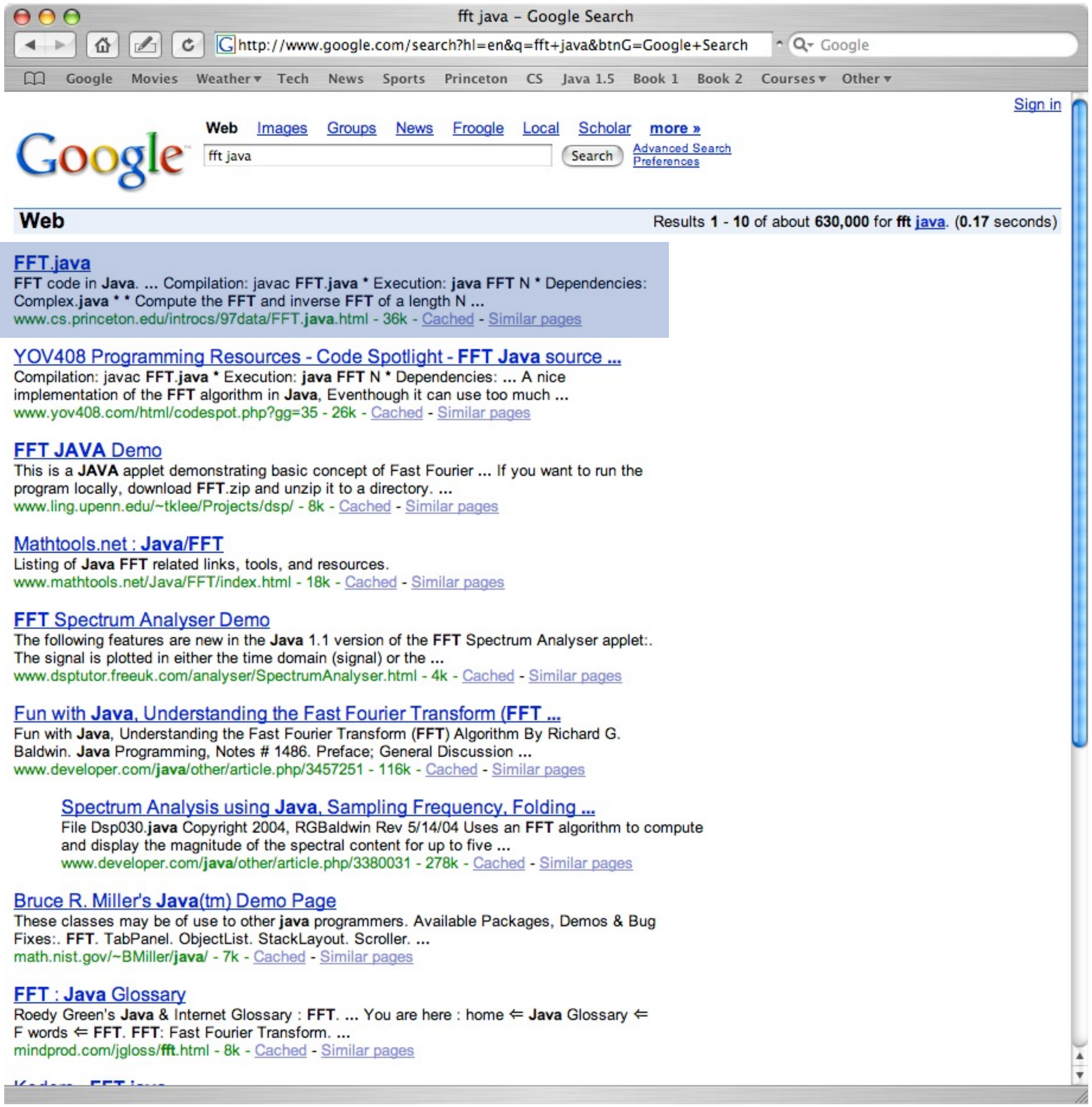
MIT Book p. 917-918

A parallel FFT circuit



MIT Book p. 919

FFT in Practice ?



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 $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

Convolution algorithm.

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

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

- 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 $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, 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 × 59
```

```
2^67-1 = 147573952589676412927 = 193707721 × 761838257287
```

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

```
740375634795617128280467960974295731425931888892312890849
362326389727650340282662768919964196251178439958943305021
275853701189680982867331732731089309005525051168770632990
72396380786710086096962537934650563796359
```

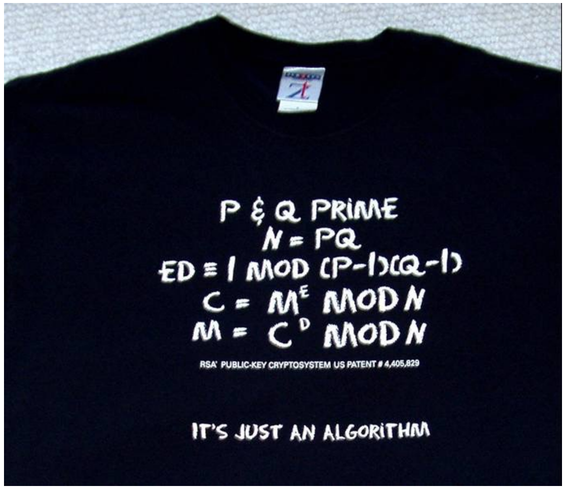
RSA-704
($30,000 prize if you can factor)

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.

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 $(p\text{-}1)\,(q\text{-}1)$:

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

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

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

Extra Slides

Fourier Matrix Decomposition

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

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} \omega^0 & 0 & 0 & 0 \\ 0 & \omega^1 & 0 & 0 \\ 0 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & \omega^3 \end{bmatrix}$$

$$a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$y = F_n a = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2}\, a_{even} \\ F_{n/2}\, a_{odd} \end{bmatrix}$$