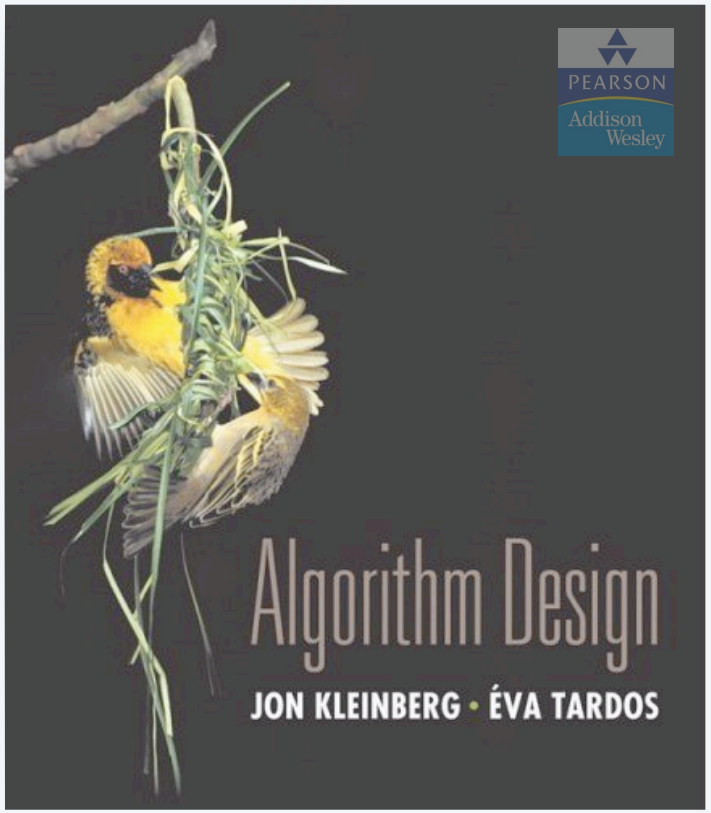


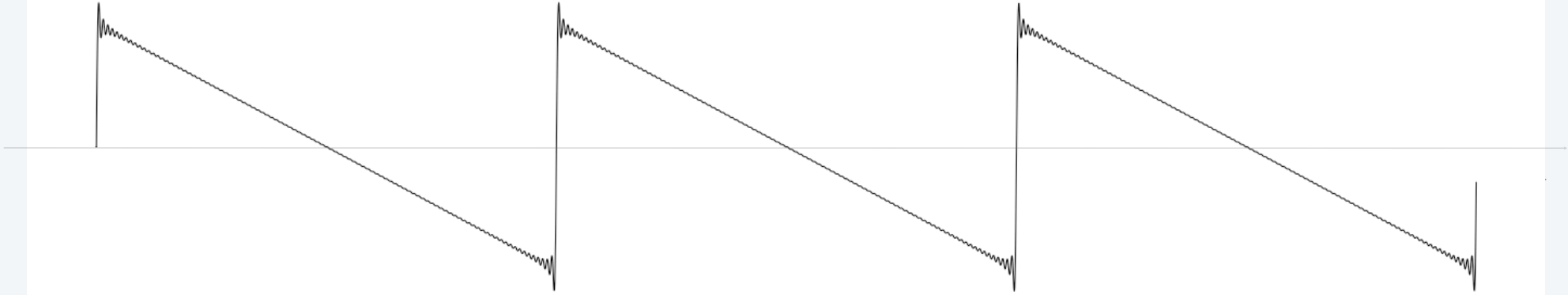
SECTION 5.6

DIVIDE AND CONQUER II

- *convolution and FFT*

Fourier analysis

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



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

Euler's identity

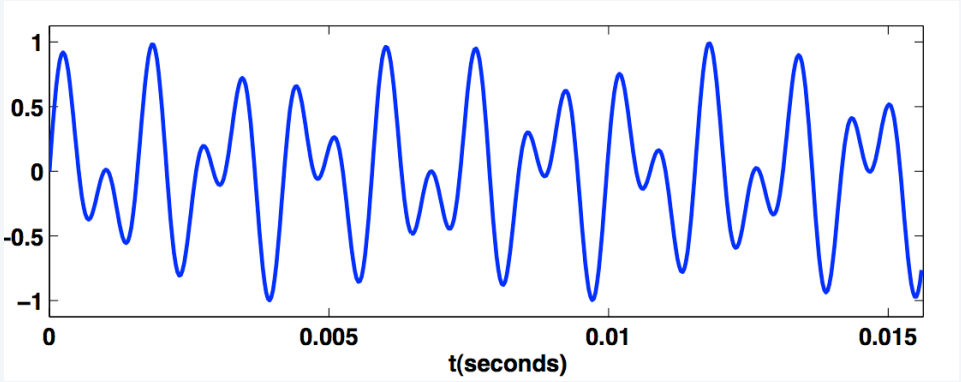
Euler's identity. $e^{ix} = \cos x + i \sin x$.

Sinusoids. Sum of sine and cosines = sum of complex exponentials.

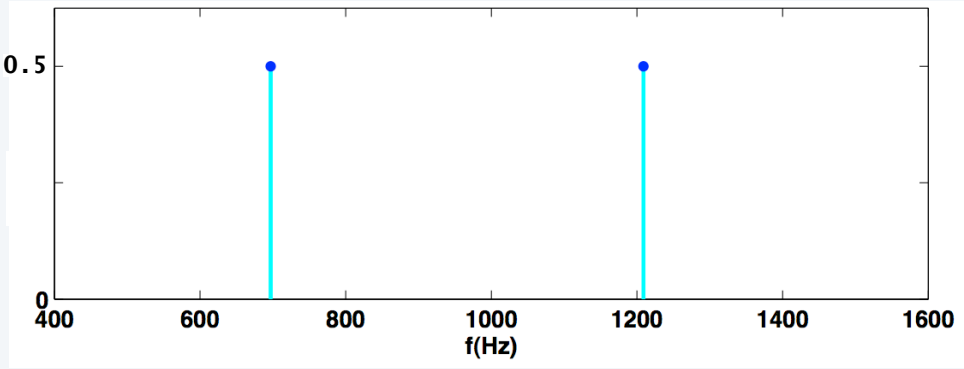
Time domain vs. frequency domain

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.



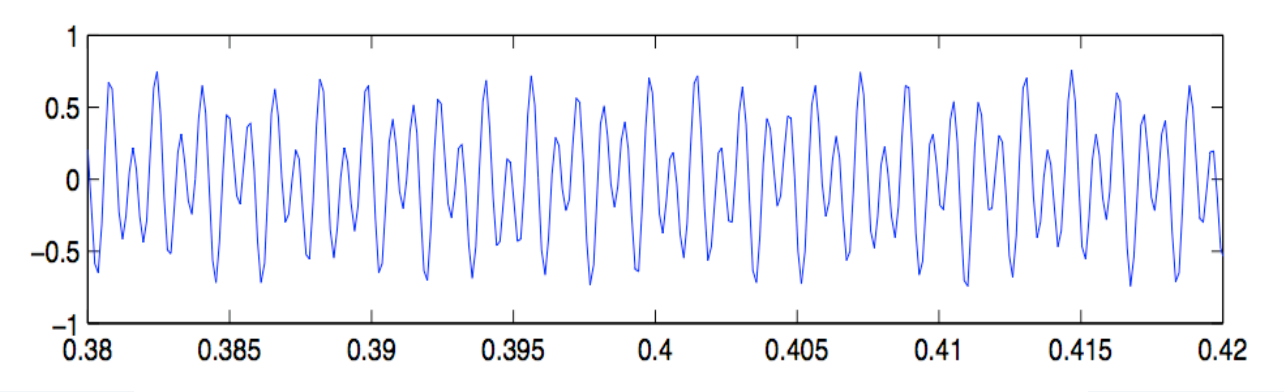
Frequency domain.



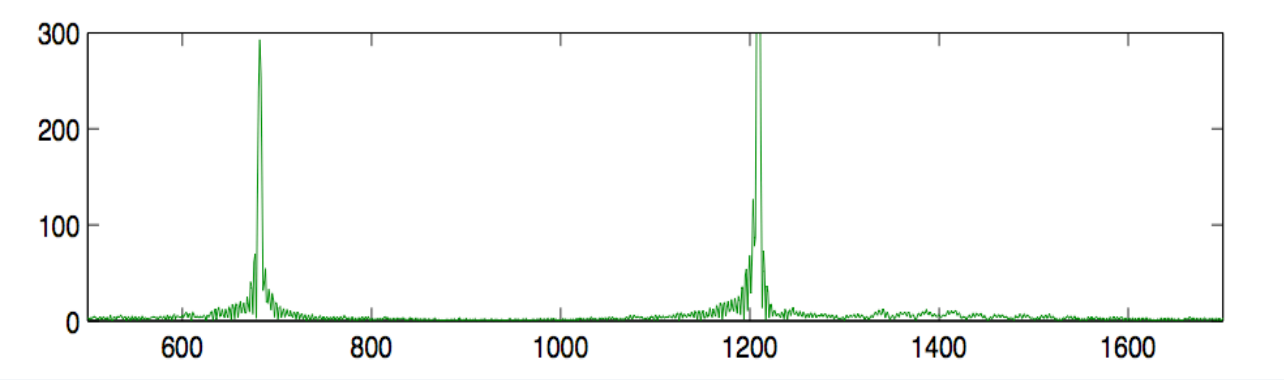
Reference: Cleve Moler, Numerical Computing with MATLAB

Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

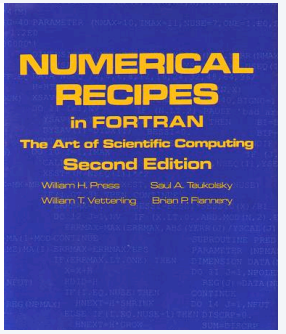
Fast Fourier transform

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

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

" 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*



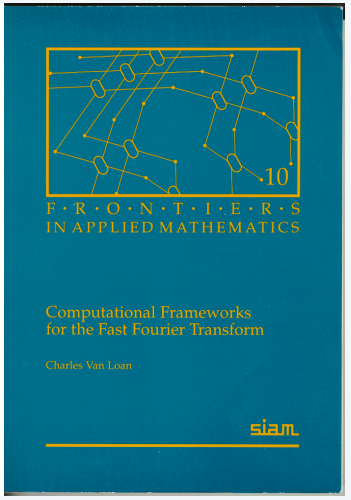
Fast Fourier transform: applications

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.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

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

— Charles van Loan



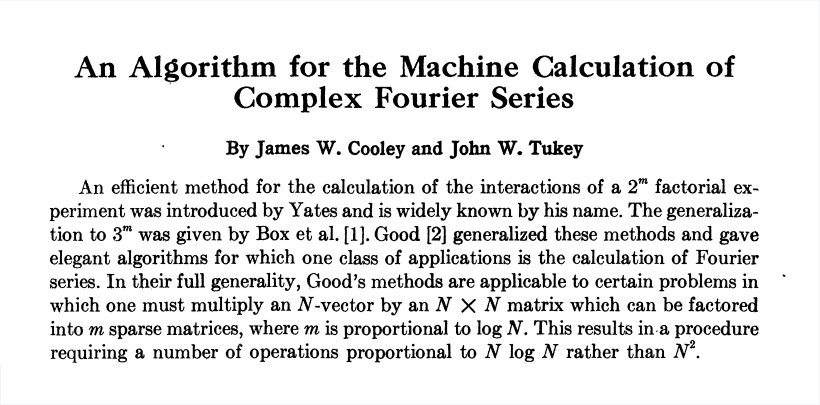
Fast Fourier transform: brief history

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

Runge-König (1924). Laid theoretical groundwork.

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

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



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2.

paper published only after IBM lawyers decided not to
set precedent of patenting numerical algorithms
(conventional wisdom: nobody could make money selling software!)

Importance not fully realized until advent of digital computers.

Polynomials: coefficient representation

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.

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

A modest Ph.D. dissertation title

DEMONSTRATIO NOVA
THEOREMATIS
OMNEM FVNCTIONEM ALGEBRAICAM
RATIONALEM INTEGRAM
VNIVS VARIABILIS
IN FACTORES REALES PRIMI VEL SECUNDI GRADVS
RESOLVI POSSE

AVCTORE
CAROLO FRIDERICO GAVSS
HELMSTADII
APVD C. G. FLECKEISEN. 1799

1.

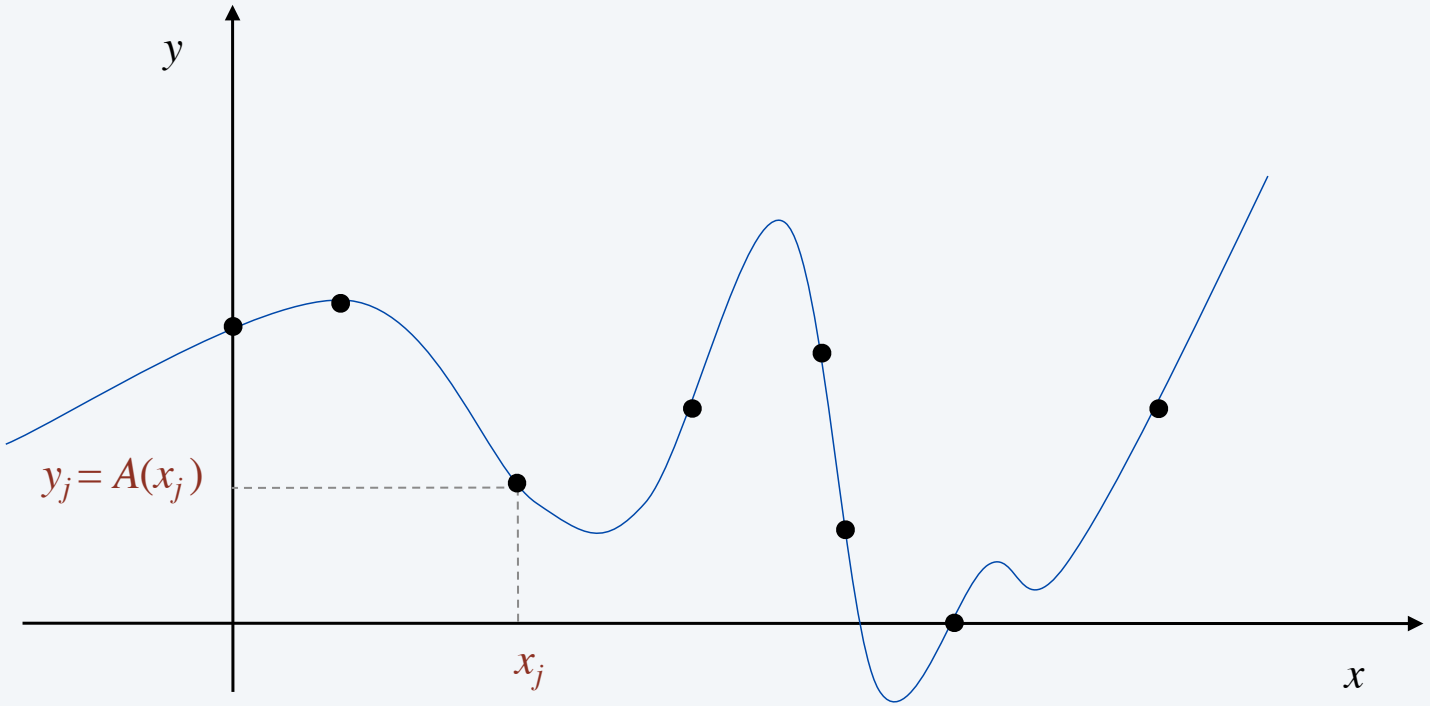
Quaelibet aequatio algebraica determinata reduci potest ad formam $x^m+Ax^{m-1}+Bx^{m-2}+$ etc. $+M=0$, ita vt m sit numerus integer positiuus. Si partem primam huius aequationis per X denotamus, aequationique $X=0$ per plures valores inaequales ipsius x satisfieri supponimus, puta ponendo $x=\alpha$, $x=\beta$, $x=\gamma$ etc. functio X per productum e factoribus $x-\alpha$, $x-\beta$, $x-\gamma$ etc. diuisibilis erit. Vice versa, si productum e pluribus factoribus simplicibus $x-\alpha$, $x-\beta$, $x-\gamma$ etc. functionem X metitur: aequationi $X=0$ satisfiet, aequando ipsam x cuicunque quantitatum α, β, γ etc. Denique si X producto ex m factoribus talibus simplicibus aequalis est (siue omnes diuersi sint, siue quidam ex ipsis identici): alii factores simplices praeter hos functionem X metiri non poterunt. Quamobrem aequatio m^{ti} gradus plures quam m radices habere nequit; simul vero patet, aequationem m^{ti} gradus *pauciores* radices habere posse, etsi X in m factores simplices resolubilis sit:

"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."

Polynomials: point-value representation

Fundamental theorem of algebra. 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]

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

Add. $O(n)$ arithmetic operations.

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

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

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

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

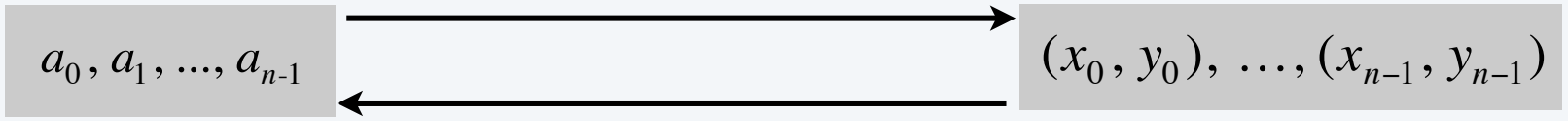
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

Converting between two 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.



coefficient representation **point-value representation**

Converting between two representations: brute force

Coefficient ⇒ 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 ⇒ 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 $O(n^{2.3727})$ via fast matrix multiplication

Divide-and-conquer

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

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
- $A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$
- $A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$
- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

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

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

Coefficient to point-value representation: intuition

Coefficient ⇒ 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 up polynomial into even and odd powers.

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

Intuition. Choose two points to be ± 1.

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

← **Can evaluate polynomial of degree ≤ n at 2 points by evaluating two polynomials of degree ≤ ½n at 1 point.**

Coefficient to point-value representation: intuition

Coefficient ⇒ 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 up polynomial into even and odd powers.

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

Intuition. Choose 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 ≤ n at 4 points by evaluating two polynomials of degree ≤ ½n at 2 point.**

Discrete Fourier transform

Coefficient ⇒ 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

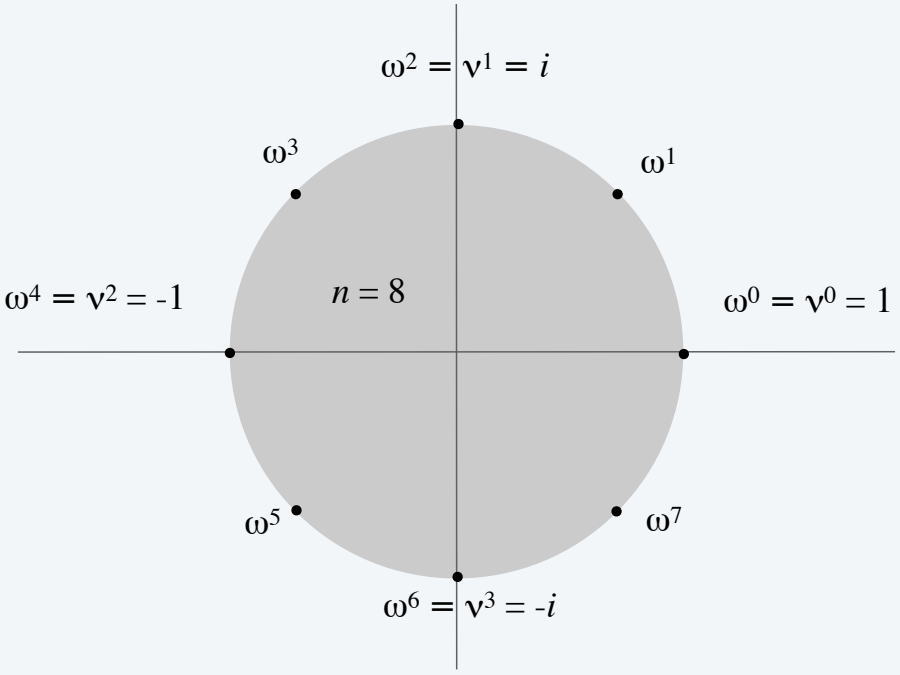
Roots of unity

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 $\nu = \omega^2 = e^{4\pi i / n}$.



Fast Fourier transform

Goal. Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \ldots + a_{n-1}\, x^{n-1}$ 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.

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

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

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

FFT: implementation

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

IF $(n = 1)$ RETURN a_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ FFT $(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ FFT $(n/2, a_1, a_3, a_4, \ldots, a_{n-1})$.

FOR $k = 0$ TO $n/2 - 1$.

$\omega^k \leftarrow e^{2\pi i k/n}$.

$y_k \leftarrow e_k + \omega^k d_k$.

$y_{k+n/2} \leftarrow e_k - \omega^k d_k$.

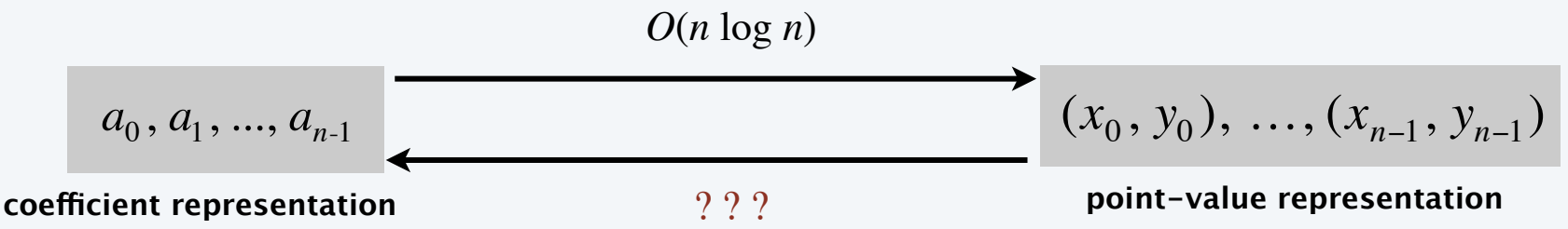
RETURN $(y_0, y_1, y_2, \ldots, y_{n-1})$.

FFT: summary

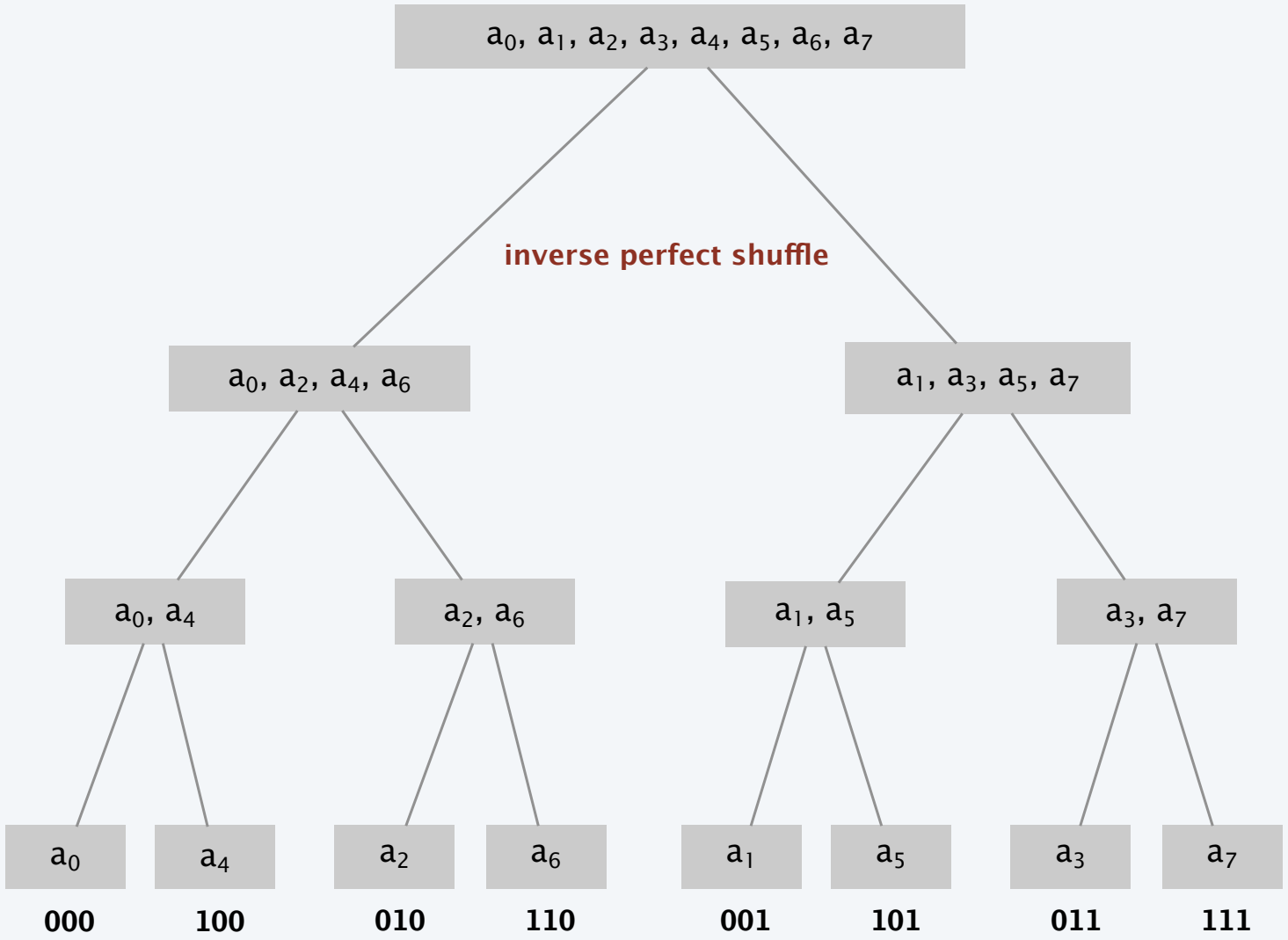
Theorem. The FFT algorithm evaluates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ steps and $O(n)$ extra space.

assumes n is a power of 2

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



FFT: recursion tree



"bit-reversed" order

Inverse discrete Fourier transform

Point-value ⇒ 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 $(F_n)^{-1}$

Inverse discrete Fourier transform

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

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

F_n / $\sqrt{n}$ is a unitary matrix

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 the result 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 (below)

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: implementation

Note. Need to divide result by n.

INVERSE-FFT $(n, a_0, a_1, a_2, \ldots, a_{n-1})$

IF $(n = 1)$ RETURN a_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ INVERSE-FFT $(n/2, a_0, a_2, a_4, \ldots, a_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ INVERSE-FFT $(n/2, a_1, a_3, a_4, \ldots, a_{n-1})$.

FOR $k = 0$ TO $n/2 - 1$.

  $\omega^k \leftarrow e^{-2\pi i k/n}$.

  $y_k \leftarrow (e_k + \omega^k d_k)$.

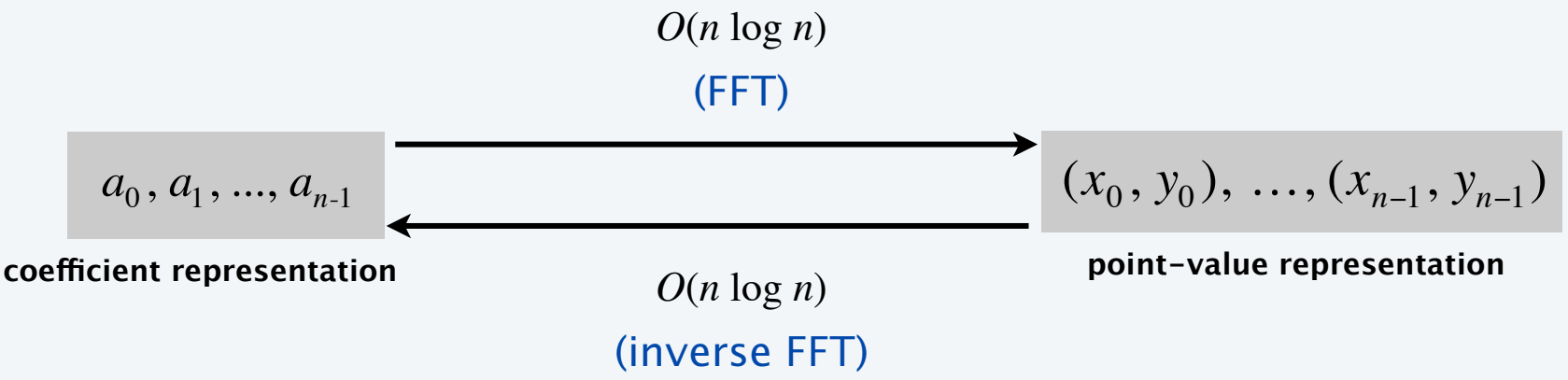
  $y_{k+n/2} \leftarrow (e_k - \omega^k d_k)$.

RETURN $(y_0, y_1, y_2, \ldots, y_{n-1})$.

Inverse FFT: summary

Theorem. The 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

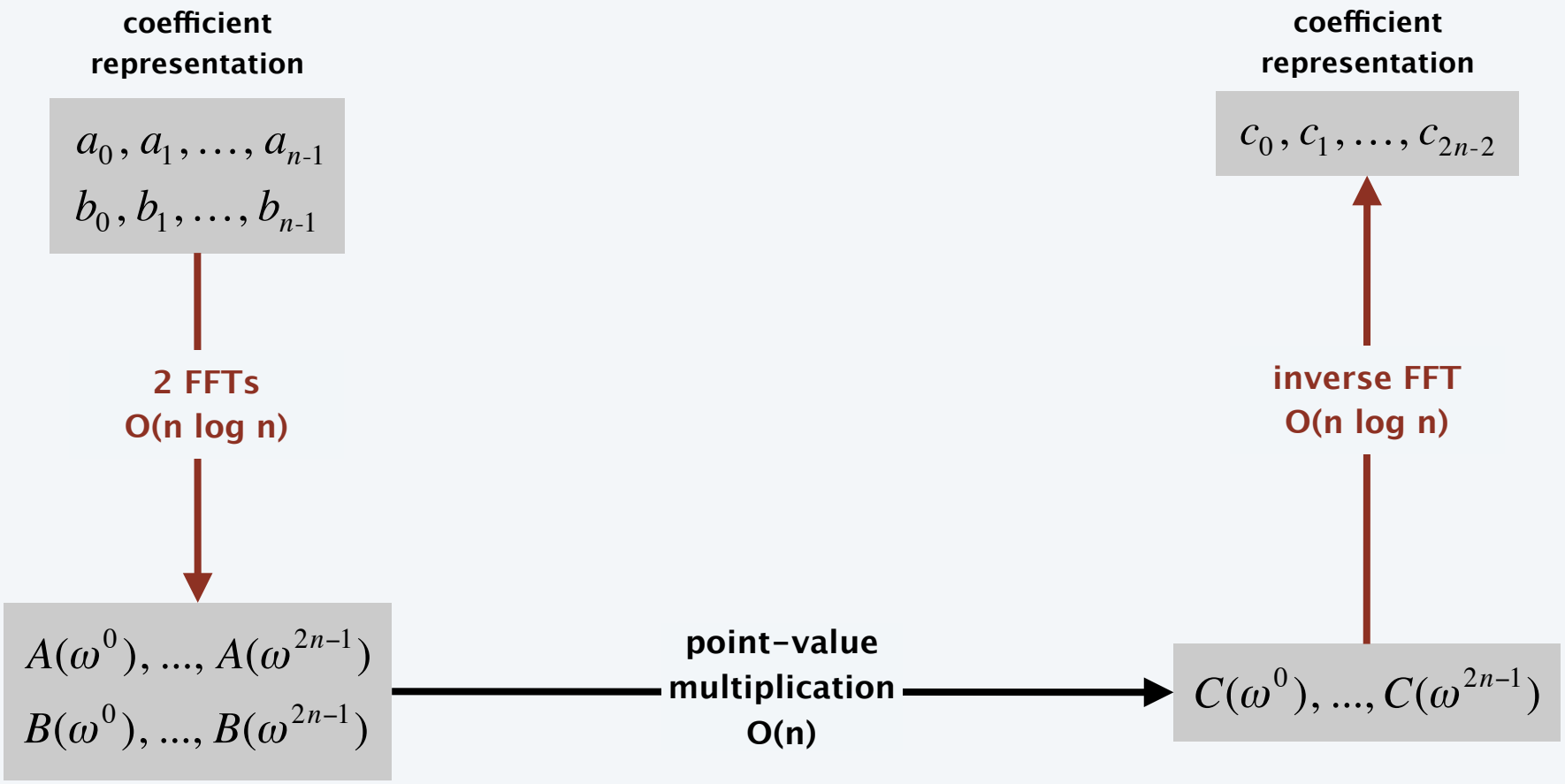


Polynomial multiplication

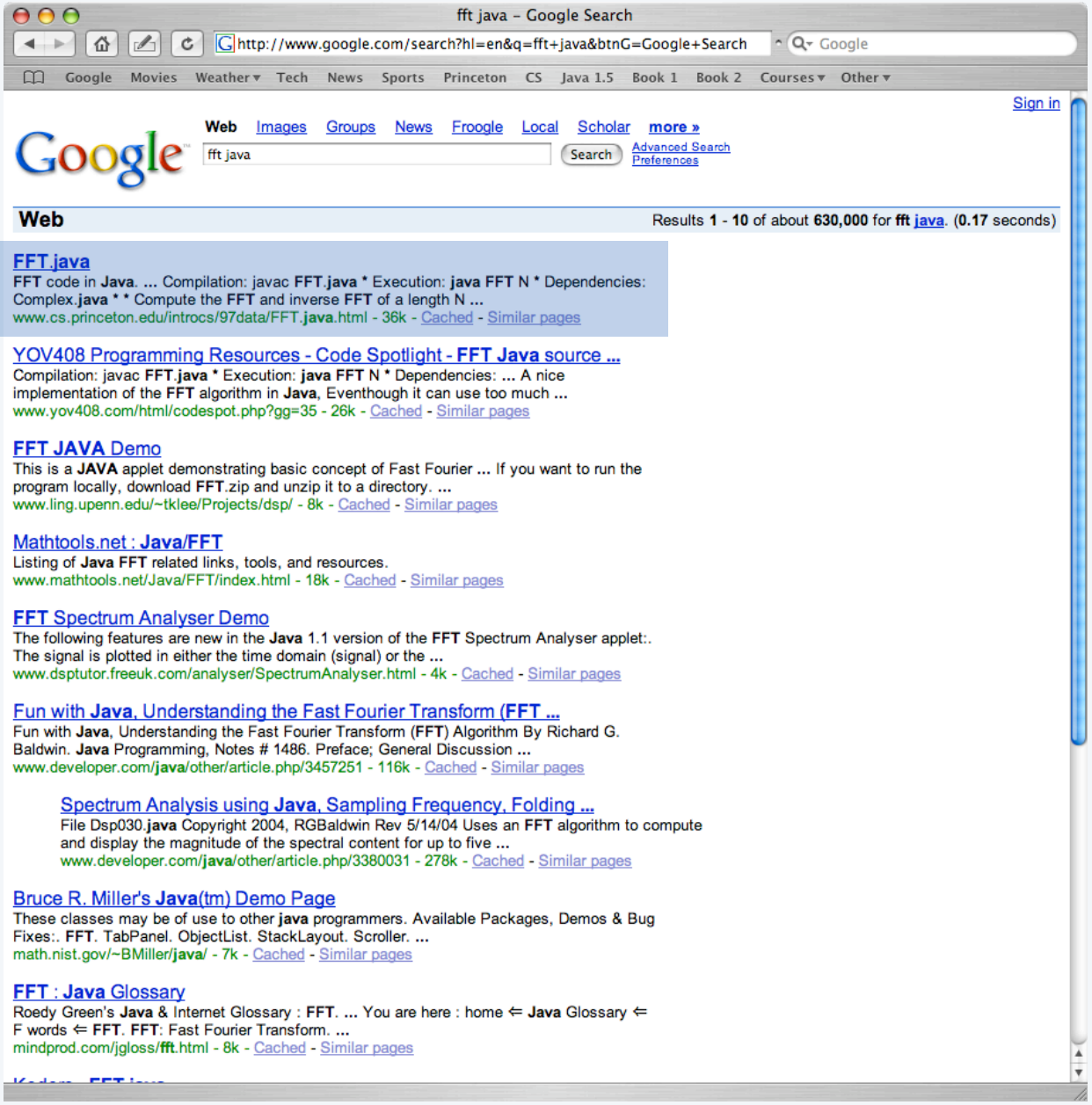
Theorem. Can multiply two degree $n-1$ polynomials in $O(n \log n)$ steps.

Pf.

pad with 0s to make n a power of 2



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.

- Core algorithm is nonrecursive version of Cooley-Tukey.
- 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.
- Runs in $O(n \log n)$ time, even when n is prime.
- Multidimensional FFTs.

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.

- 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.

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

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

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

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

Practice. [GNU Multiple Precision Arithmetic Library]

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