DIVIDE AND CONQUER II

‣ master theorem

‣ integer multiplication

‣ matrix multiplication

‣ convolution and FFT

THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN

INTRODUCTION TO
ALGORITHMS

THIRD EDITION

SECTIONS 4.3–4.6

DIVIDE AND CONQUER II

‣ *master theorem*

‣ integer multiplication

‣ matrix multiplication

‣ convolution and FFT

Goal. Recipe for solving common divide-and-conquer recurrences:

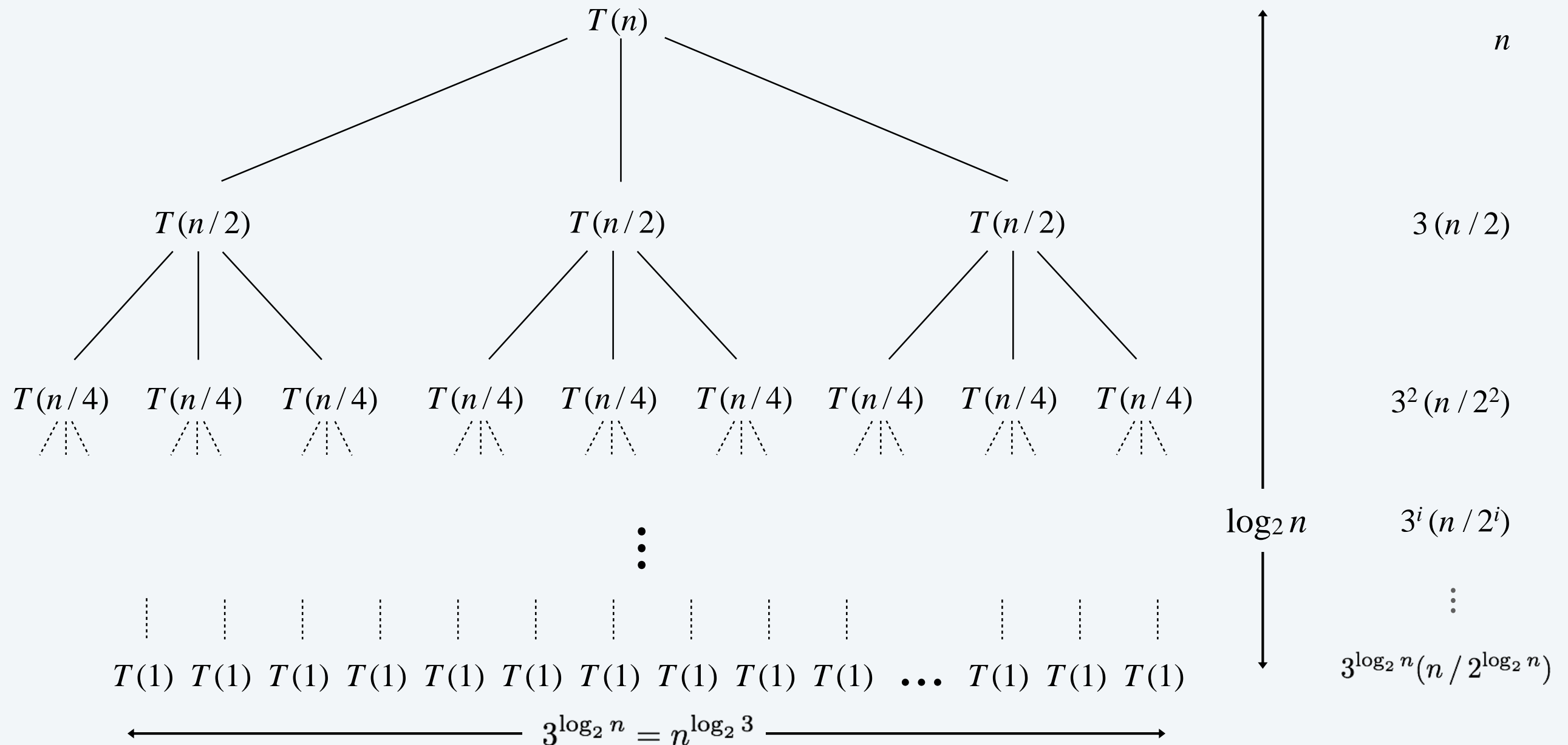$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

Terms.
- $a \geq 1$ is the number of subproblems.
- $b > 0$ is the factor by which the subproblem size decreases.
- $f(n)$ = work to divide/merge subproblems.

Recursion tree.
- $k = \log_b n$ levels.
- a^i = number of subproblems at level i.
- n / b^i = size of subproblem at level i.
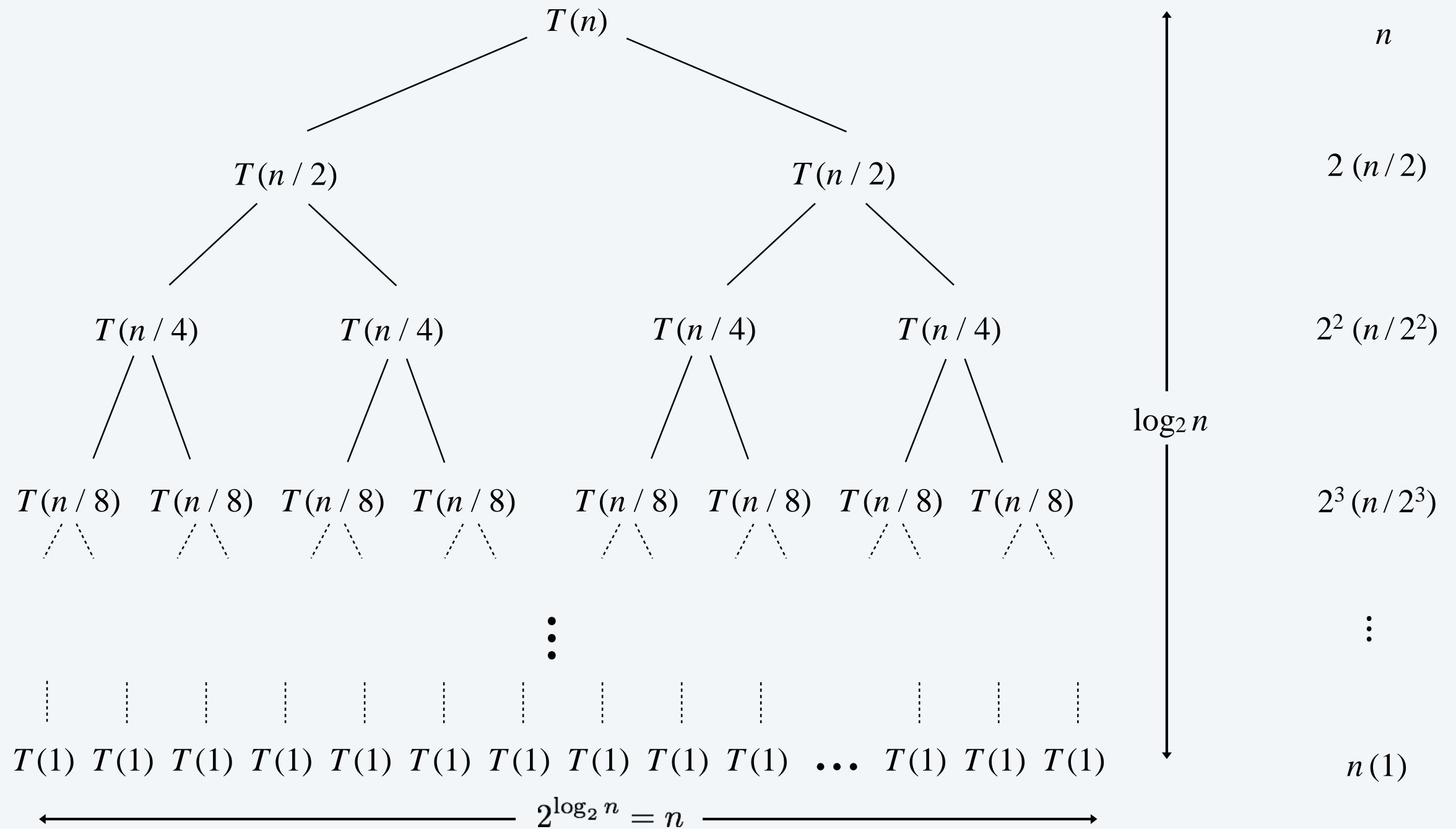
Case 1: total cost dominated by cost of leaves

Ex 1. If $T(n)$ satisfies $T(n) = 3\, T(n\,/\,2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n^{\lg 3})$.

$T(n)$ n

$T(n/2)$ $T(n/2)$ $T(n/2)$ $3\,(n\,/\,2)$

$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $3^2\,(n\,/\,2^2)$

$\vdots$

$\log_2 n$ $3^i\,(n\,/\,2^i)$

$\vdots$

$T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ \cdots\ T(1)\ T(1)\ T(1)$ $3^{\log_2 n}(n\,/\,2^{\log_2 n})$

$\overleftrightarrow{\qquad 3^{\log_2 n} = n^{\log_2 3} \qquad}$

$$r = 3\,/\,2\, > 1 \qquad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\, n \;=\; \frac{r^{1 + \log_2 n} - 1}{r - 1}\, n \;=\; 3n^{\log_2 3} - 2n$$

4

Case 2: total cost evenly distributed among levels

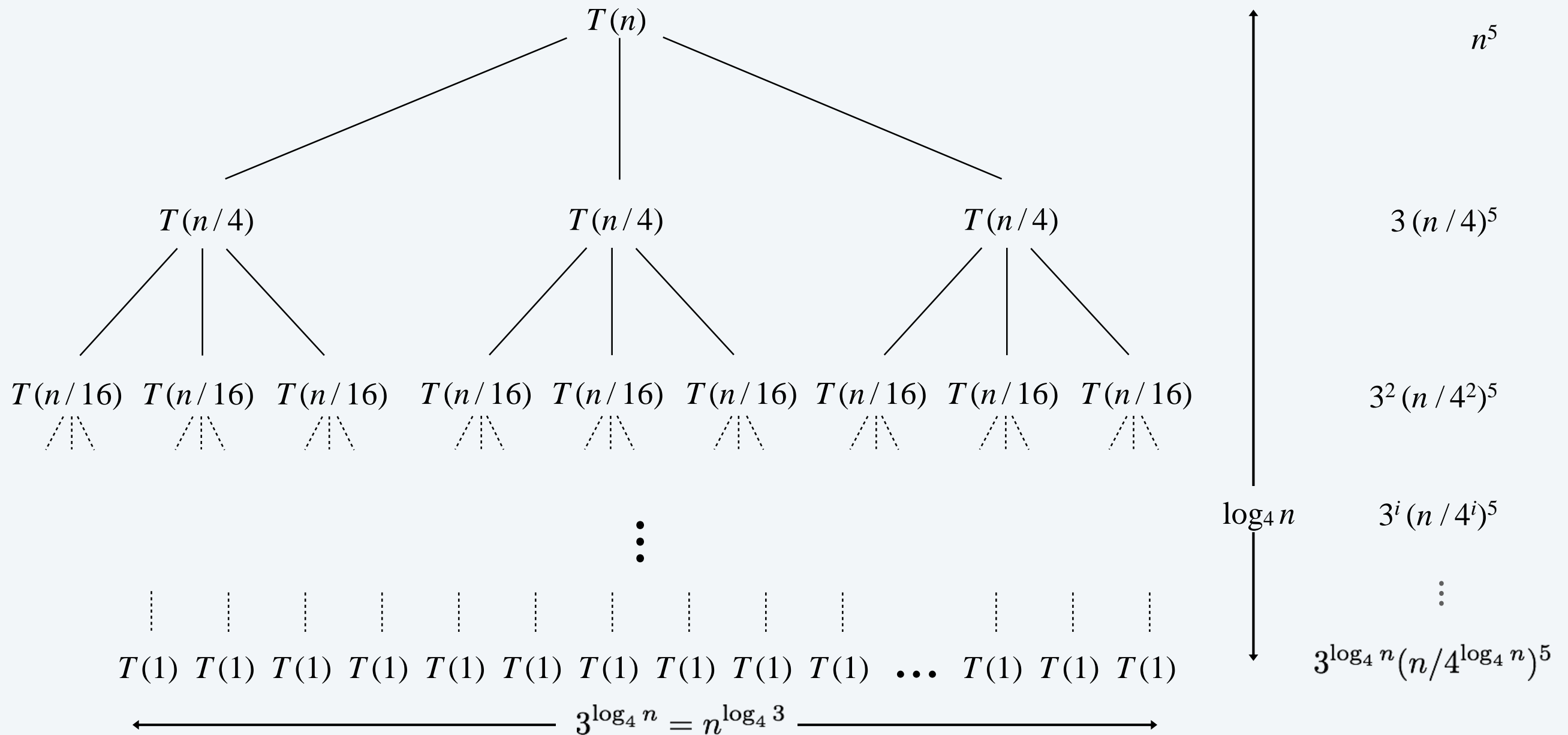**Ex 2.** If $T(n)$ satisfies $T(n) = 2\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log n)$.

$r = 1$ $\qquad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\, n \ = \ n\,(\log_2 n + 1)$

Ex 3. If $T(n)$ satisfies $T(n) = 3\,T(n\,/\,4) + n^5$, with $T(1) = 1$, then $T(n) = \Theta(n^5)$.

$T(n)$ n^5

$T(n/4)$ $T(n/4)$ $T(n/4)$ $3\,(n\,/\,4)^5$

$T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $T(n/16)$ $3^2\,(n\,/\,4^2)^5$

$\log_4 n$ $3^i\,(n\,/\,4^i)^5$

$T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ T(1)\ \cdots\ T(1)\ T(1)\ T(1)$ $3^{\log_4 n}(n/4^{\log_4 n})^5$

$$3^{\log_4 n} = n^{\log_4 3}$$

$$r = 3\,/\,4^5 < 1 \qquad n^5 \leq T(n) \leq (1 + r + r^2 + r^3 + \ldots)\,n^5 \leq \frac{1}{1-r}\,n^5$$

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Ex. $T(n) = 3\,T(n/2) + n$.

- $a = 3,\ b = 2,\ f(n) = n,\ k = \log_2 3$.
- $T(n) = \Theta(n^{\lg 3})$.

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where $n\,/\,b$ means either $\lfloor n\,/\,b\rfloor$ or $\lceil n\,/\,b\rceil$. Let $k = \log_b a$. Then,

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Ex. $T(n) = 2\,T(n\,/\,2) + \Theta(n \log n)$.
- $a = 2,\ b = 2,\ f(n) = 17\,n,\ \ k = \log_2 2 = 1,\ p = 1$.
- $T(n) = \Theta(n \log^2 n)$.

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\, T\left(\frac{n}{b}\right) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

regularity condition holds
if f(n) = Θ(n$^{k + \varepsilon}$)

Case 3. If $f(n) = \Omega(n^{k + \varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Ex. $T(n) = 3\,T(n/4) + n^5$.

- $a = 3,\ b = 4,\ f(n) = n^5,\ k = \log_4 3$.
- $T(n) = \Theta(n^5)$.

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

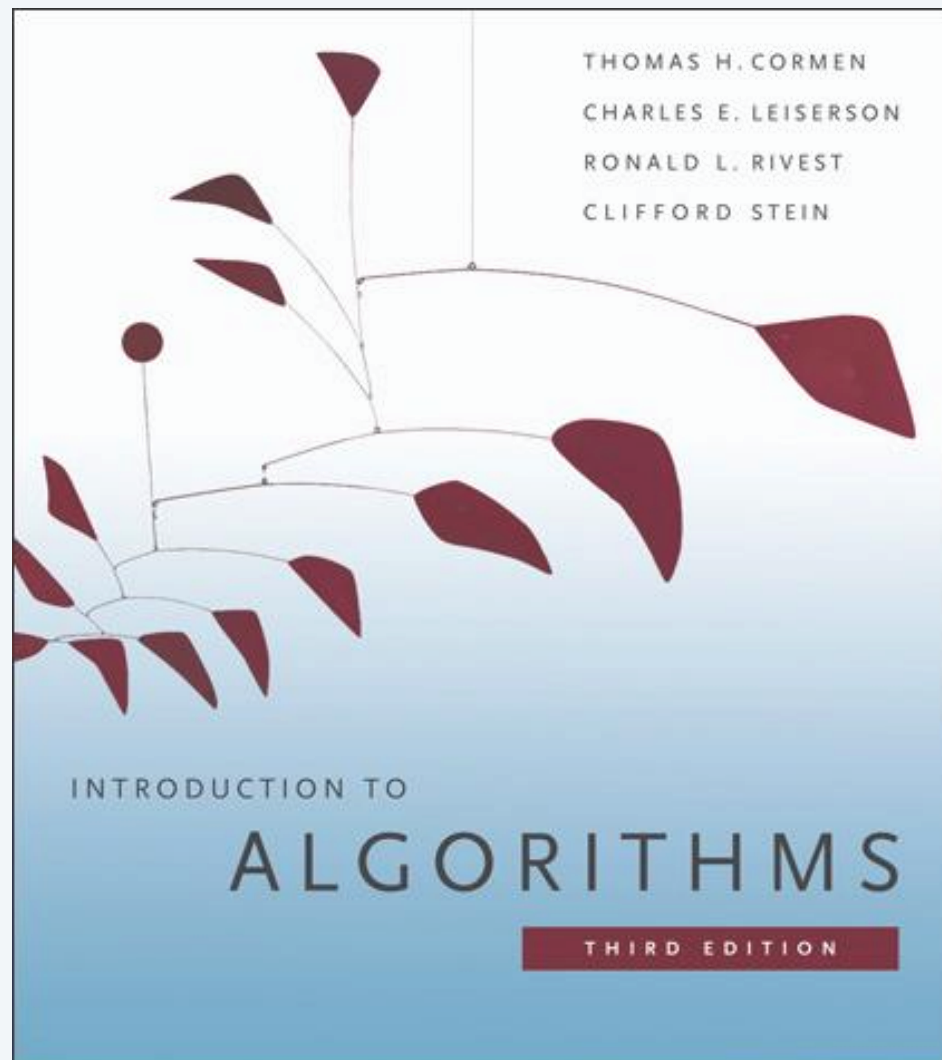where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Case 3. If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Pf sketch.

- Use recursion tree to sum up terms (assuming n is an exact power of b).
- Three cases for geometric series.
- Deal with floors and ceilings.

DIVIDE AND CONQUER II

‣ *master theorem*

‣ *integer multiplication*

‣ **matrix multiplication**

‣ *convolution and FFT*

SECTION 4.2

Dot product. Given two length n vectors a and b, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

$$a = \begin{bmatrix} .70 & .20 & .10 \end{bmatrix}$$
$$b = \begin{bmatrix} .30 & .40 & .30 \end{bmatrix}$$
$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Remark. Grade-school dot product algorithm is asymptotically optimal.

Matrix multiplication. Given two n-by-n matrices A and B, compute $C = AB$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

$$
\begin{bmatrix}
c_{11} & c_{12} & \cdots & c_{1n} \\
c_{21} & c_{22} & \cdots & c_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix}
=
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\times
\begin{bmatrix}
b_{11} & b_{12} & \cdots & b_{1n} \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
$$

$$
\begin{bmatrix}
.59 & .32 & .41 \\
.31 & .36 & .25 \\
.45 & .31 & .42
\end{bmatrix}
=
\begin{bmatrix}
.70 & .20 & .10 \\
.30 & .60 & .10 \\
.50 & .10 & .40
\end{bmatrix}
\times
\begin{bmatrix}
.80 & .30 & .50 \\
.10 & .40 & .10 \\
.10 & .30 & .40
\end{bmatrix}
$$

Q. Is grade-school matrix multiplication algorithm asymptotically optimal?

Block matrix multiplication

$$C_{11} \qquad A_{11} \qquad A_{12} \qquad B_{11}$$

$$
\begin{bmatrix}
152 & 158 & 164 & 170 \\
504 & 526 & 548 & 570 \\
856 & 894 & 932 & 970 \\
1208 & 1262 & 1316 & 1370
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{bmatrix}
\times
\begin{bmatrix}
16 & 17 & 18 & 19 \\
20 & 21 & 22 & 23 \\
24 & 25 & 26 & 27 \\
28 & 29 & 30 & 31
\end{bmatrix}
$$

$$B_{11}$$

$$
C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} =
\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}
\times
\begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix}
+
\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}
\times
\begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix}
=
\begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}
$$

Matrix multiplication: warmup

To multiply two n-by-n matrices A and B:

- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$
\begin{aligned}
C_{11} &= \left(A_{11} \times B_{11}\right) + \left(A_{12} \times B_{21}\right) \\
C_{12} &= \left(A_{11} \times B_{12}\right) + \left(A_{12} \times B_{22}\right) \\
C_{21} &= \left(A_{21} \times B_{11}\right) + \left(A_{22} \times B_{21}\right) \\
C_{22} &= \left(A_{21} \times B_{12}\right) + \left(A_{22} \times B_{22}\right)
\end{aligned}
$$

Running time. Apply case 1 of Master Theorem.

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

Key idea. multiply 2-by-2 blocks with only 7 multiplications.
(plus 11 additions and 7 subtractions)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$P_1 \leftarrow A_{11} \square (B_{12} - B_{22})$

$P_2 \leftarrow (A_{11} + A_{12}) \square B_{22}$

$P_3 \leftarrow (A_{21} + A_{22}) \square B_{11}$

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_1 + P_5 - P_3 - P_7$

$P_4 \leftarrow A_{22} \square (B_{21} - B_{11})$

$P_5 \leftarrow (A_{11} + A_{22}) \square (B_{11} + B_{22})$

$P_6 \leftarrow (A_{12} - A_{22}) \square (B_{21} + B_{22})$

$P_7 \leftarrow (A_{11} - A_{21}) \square (B_{11} + B_{12})$

Pf. $C_{12} = P_1 + P_2$

$= A_{11} \square (B_{12} - B_{22}) + (A_{11} + A_{12}) \square B_{22}$

$= A_{11} \square B_{12} + A_{12} \square B_{22}$. ✔

STRASSEN (n, A, B)

IF $(n = 1)$ RETURN $A \cdot B$.

Partition A and B into 2-by-2 block matrices.

$P_1 \leftarrow$ STRASSEN $(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow$ STRASSEN $(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow$ STRASSEN $(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow$ STRASSEN $(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow$ STRASSEN $(n / 2, (A_{11} + A_{22}) \cdot (B_{11} + B_{22}))$.

$P_6 \leftarrow$ STRASSEN $(n / 2, (A_{12} - A_{22}) \cdot (B_{21} + B_{22}))$.

$P_7 \leftarrow$ STRASSEN $(n / 2, (A_{11} - A_{21}) \cdot (B_{11} + B_{12}))$.

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

RETURN C.

assume n is a power of 2

keep track of indices of submatrices (don't copy matrix entries)

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Q. What if n is not a power of 2?

A. Could pad matrices with zeros.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm when n is "small" .

Common misperception. *"Strassen is only a theoretical curiosity."*
- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2{,}048$.
- Range of instances where it's useful is a subject of controversy.

Matrix multiplication. Given two n-by-n matrices, compute their product.

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$|A|$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L\,U$	$\Theta(MM(n))$
least squares	$\min \|Ax - b\|_2$	$\Theta(MM(n))$

numerical linear algebra problems with the same complexity as matrix multiplication

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?

A. Yes! [Strassen 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.807})$$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?

A. Impossible. [Hopcroft and Kerr 1971]

$$\Theta(n^{\log_2 6}) = O(n^{2.59})$$

Q. Multiply two 3-by-3 matrices with 21 scalar multiplications?

A. Unknown.

$$\Theta(n^{\log_3 21}) = O(n^{2.77})$$

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, …]

- Two 20-by-20 matrices with 4,460 scalar multiplications. $O(n^{2.805})$
- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$
- A year later. $O(n^{2.7799})$
- December 1979. $O(n^{2.521813})$
- January 1980. $O(n^{2.521801})$

History of asymptotic complexity of matrix multiplication

year	algorithm	order of growth
?	brute force	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith-Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith-Winograd	$O(n^{2.376})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.3727})$
?	?	$O(n^{2+\varepsilon})$

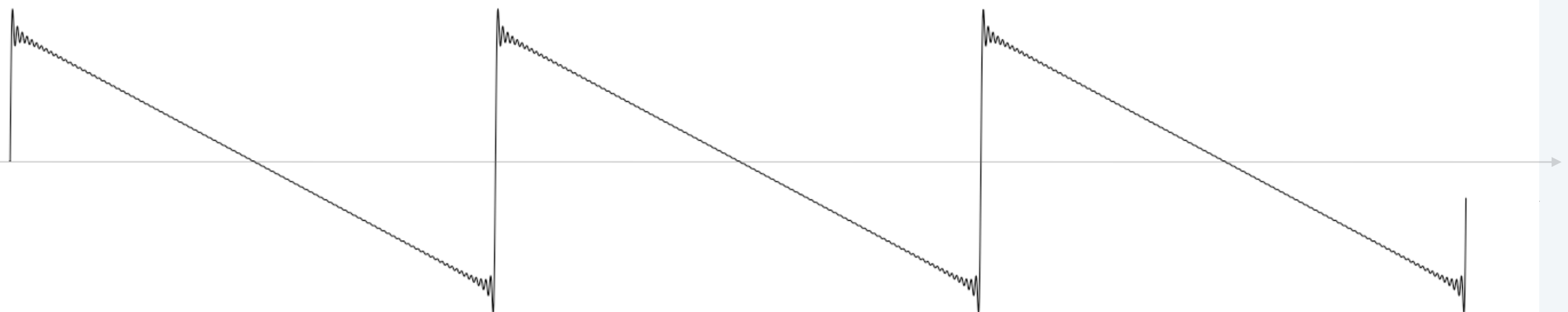number of floating-point operations to multiply two n-by-n matrices
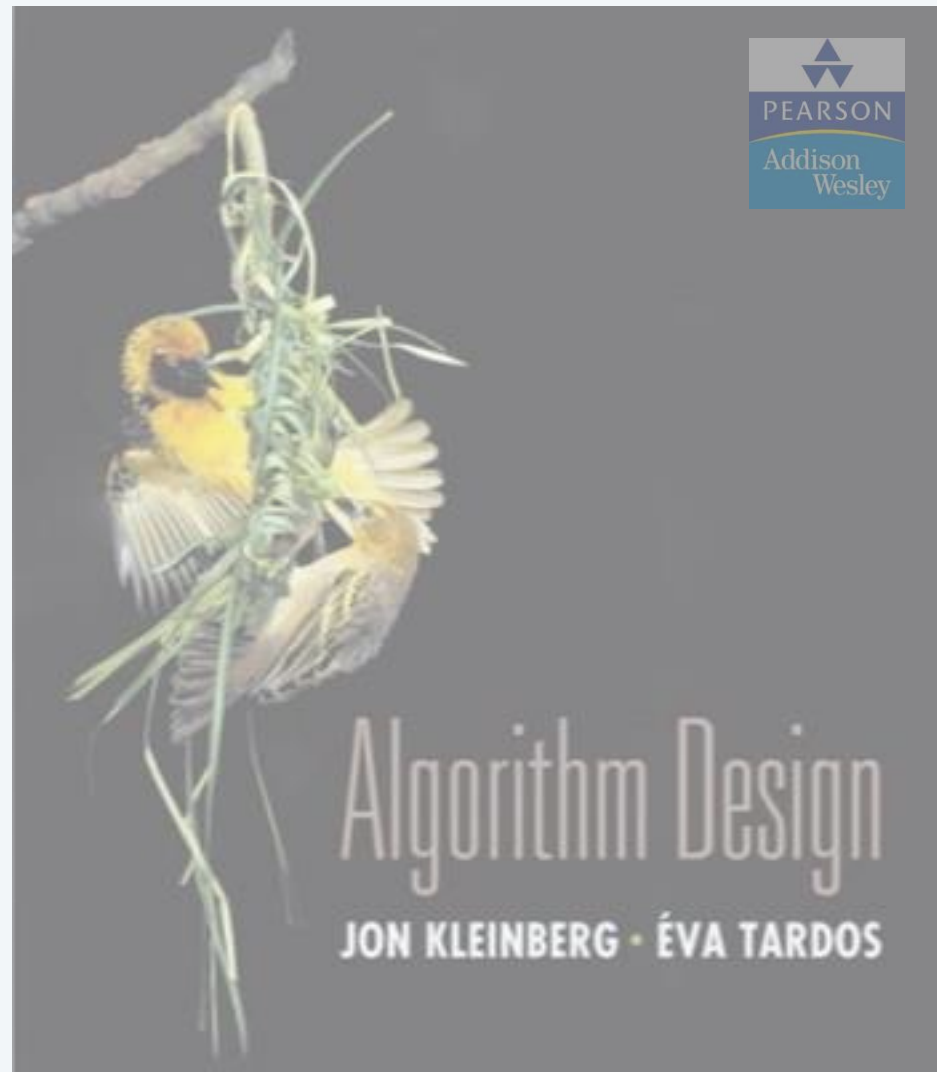
DIVIDE AND CONQUER II

‣ *master theorem*

‣ *integer multiplication*

‣ *matrix multiplication*

‣ **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. $e^{ix} = \cos x + i \sin x$.

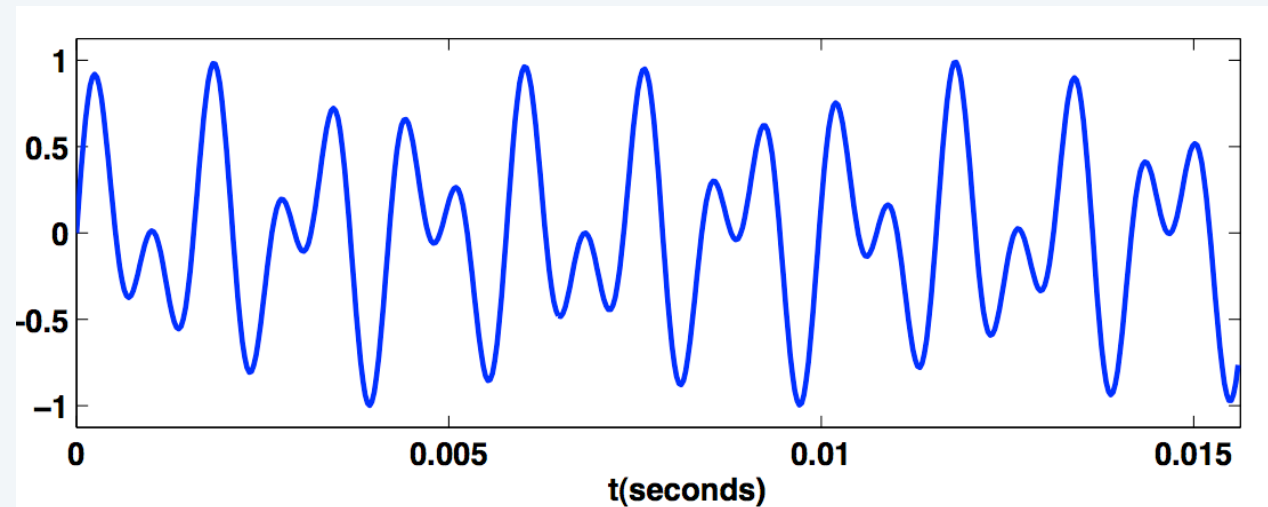**Sinusoids.** Sum of sine and cosines = sum of complex exponentials.

Time domain vs. frequency domain

Signal. [touch tone button 1] $\quad y(t) = \frac{1}{2}\sin(2\pi \cdot 697\,t) + \frac{1}{2}\sin(2\pi \cdot 1209\,t)$
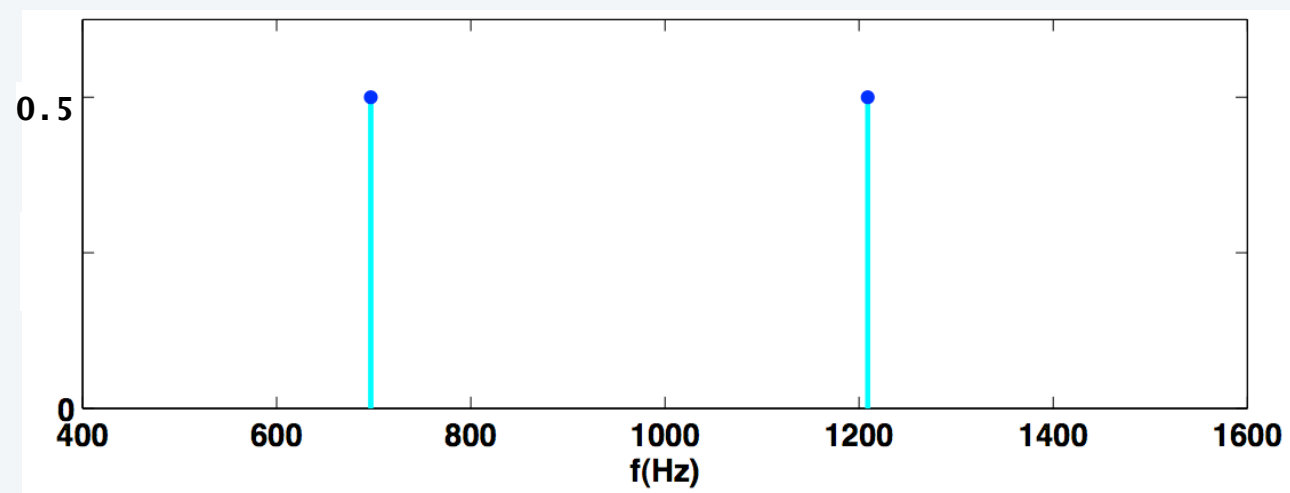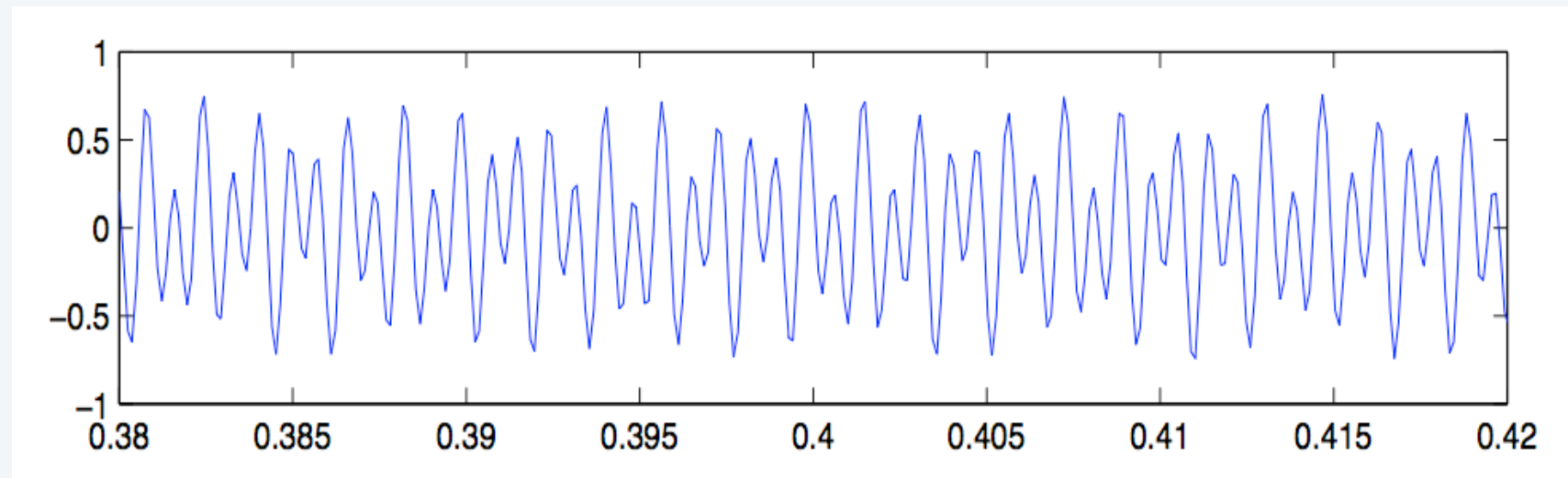
Time domain.

Frequency domain.

Reference: Cleve Moler, Numerical Computing with MATLAB

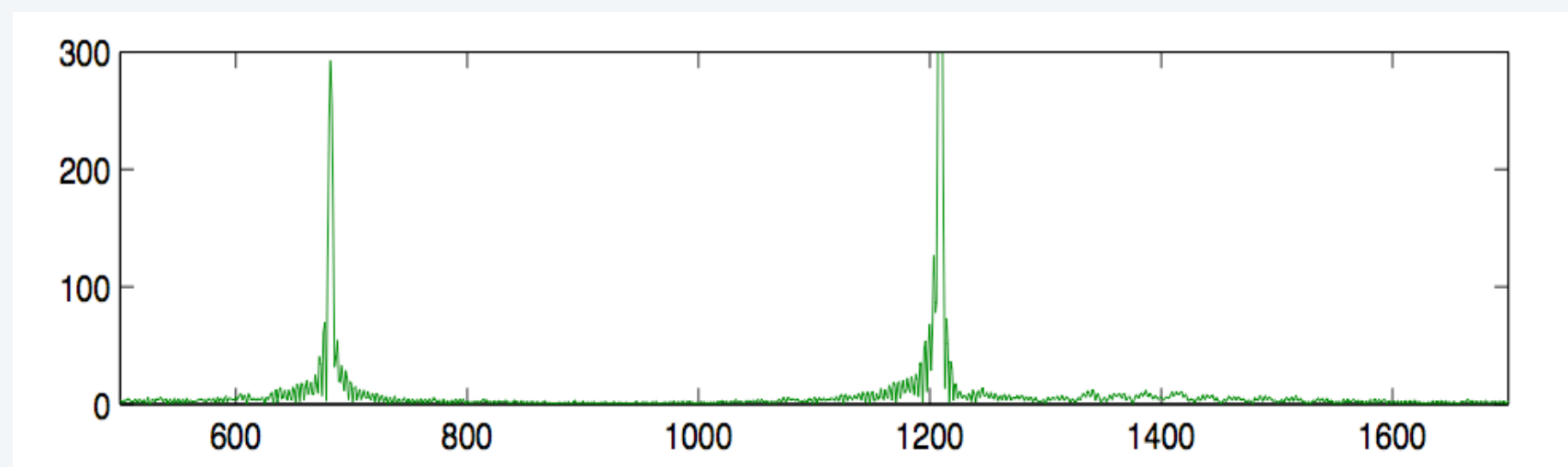# Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]

Magnitude of discrete Fourier transform.

Reference: Cleve Moler, Numerical Computing with MATLAB
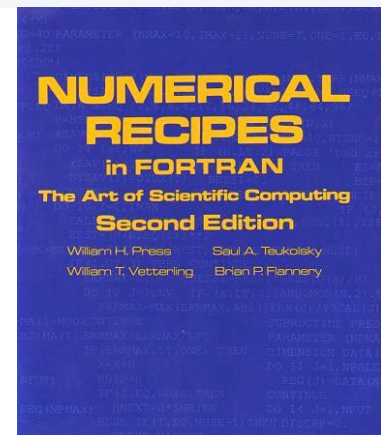
FFT. Fast way to convert between time-domain and frequency-domain.

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*

NUMERICAL
RECIPES
in FORTRAN
The Art of Scientific Computing
Second Edition
William H. Press   Saul A. Teukolsky
William T. Vetterling   Brian P. Flannery

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

10

F·R·O·N·T·I·E·R·S
IN APPLIED MATHEMATICS
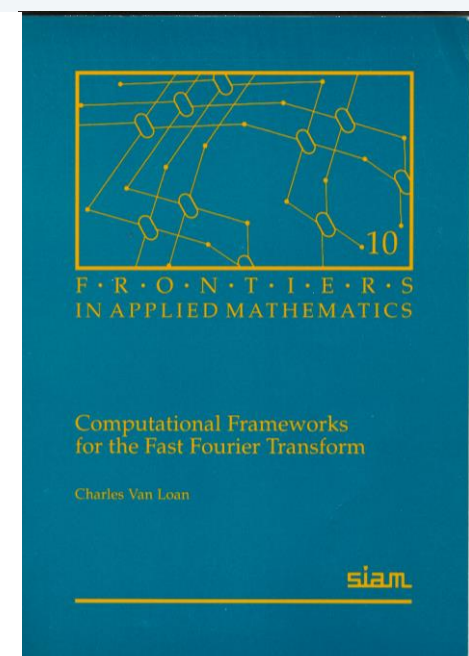
Computational Frameworks
for the Fast Fourier Transform

Charles Van Loan

siam

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.

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.

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

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."

48

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
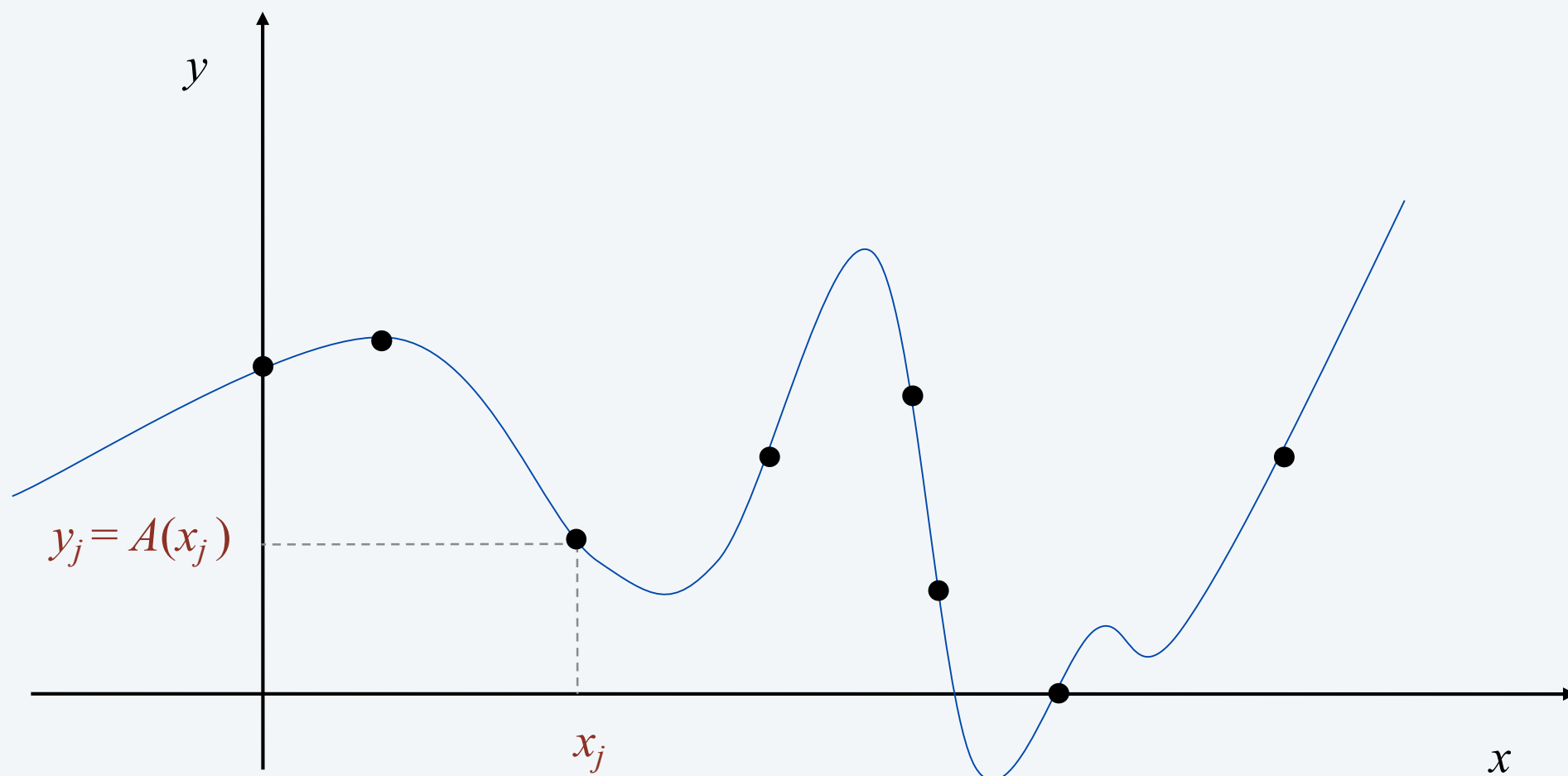
Polynomial. [point-value representation]

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

Add. $O(n)$ arithmetic operations.

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

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

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

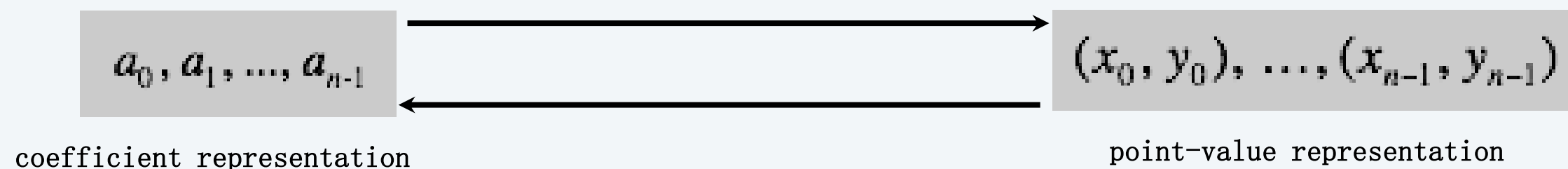**Evaluate.** $O(n^2)$ using Lagrange's formula.

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

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.

$$a_0, a_1, \dots, a_{n-1} \qquad \longleftrightarrow \qquad (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

coefficient representation

point-value representation

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

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

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

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

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

- $A(x) \quad = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$

- $A_{low}(x) \quad = a_0 + a_1x + a_2x^2 + a_3x^3.$

- $A_{high}(x) = a_4 + a_5x + a_6x^2 + a_7x^3.$

- $A(x) = A_{low}(x) + x^4 A_{high}(x).$

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

- $A(x) \quad = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$

- $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$

- $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$

- $A(x) = A_{even}(x^2) + x A_{odd}(x^2).$

Coefficient ® point-value. Given a polynomial $a_0 + a_1 x + ... + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, ..., x_{n-1}$. $\longleftarrow$ we get to choose which ones!

Divide. Break up polynomial into even and odd powers.

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

Intuition. Choose two points to be ± 1.

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

$\longleftarrow$ Can evaluate polynomial of degree $\leqslant$ n at 2 points by evaluating two polynomials of degree $\leqslant$ ½n at 1 point.

Coefficient ® point-value. Given a polynomial $a_0 + a_1x + ... + a_{n-1}\, x^{n-1}$,
evaluate it at n distinct points $x_0, ..., x_{n-1}$. $\longleftarrow$ we get to choose which ones!

Divide. Break up polynomial into even and odd powers.

- $A(x) \quad = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$
- $A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$
- $A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$
- $A(x) \quad = A_{even}(x^2) + x\, A_{odd}(x^2).$
- $A(-x) = A_{even}(x^2) - x\, A_{odd}(x^2).$

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

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

$\longleftarrow$ Can evaluate polynomial of degree $\leqslant$ n
at 4 points by evaluating two polynomials
of degree $\leqslant$ ½n at 2 point.

Discrete Fourier transform

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

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

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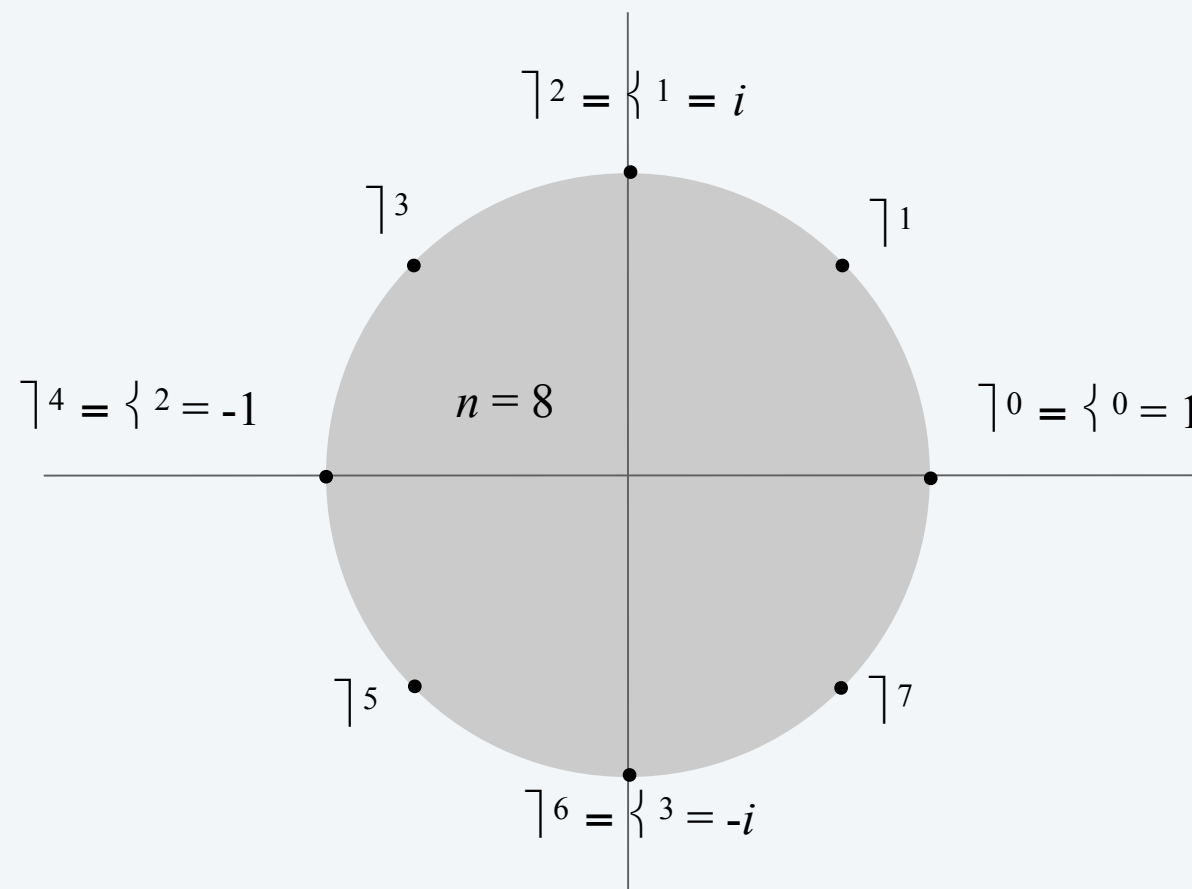$$

DFT

Fourier matrix F_n

57

Def. An nth root of unity is a complex number x such that $x^n = 1$.

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

$\omega^2 = \zeta^1 = i$

ω^3

ω^1

$\omega^4 = \zeta^2 = -1$

$n = 8$

$\omega^0 = \zeta^0 = 1$

ω^5

ω^7

$\omega^6 = \zeta^3 = -i$

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1}\,x^{n-1}$ at its nth 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 $\tfrac{1}{2}n^{th}$ 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 $(n, a_0, a_1, a_2, ..., a_{n-1})$

IF $(n = 1)$ RETURN a_0.

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

$(d_0, d_1, ..., d_{n/2-1}) \leftarrow$ FFT $(n/2, a_1, a_3, a_5, ..., a_{n-1})$.

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

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

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

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

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

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

assumes n is a power of 2

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

$O(n \log n)$

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

coefficient representation $\qquad\qquad$? ? ? $\qquad\qquad$ point-value representation

FFT: recursion tree

$$a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$$

inverse perfect shuffle

$$a_0, a_2, a_4, a_6 \qquad a_1, a_3, a_5, a_7$$

$$a_0, a_4 \qquad a_2, a_6 \qquad a_1, a_5 \qquad a_3, a_7$$

$$a_0 \qquad a_4 \qquad a_2 \qquad a_6 \qquad a_1 \qquad a_5 \qquad a_3 \qquad a_7$$

000　　　**100**　　　　**010**　　　**110**　　　　**001**　　　**101**　　　　**011**　　　**111**

"bit-reversed" order
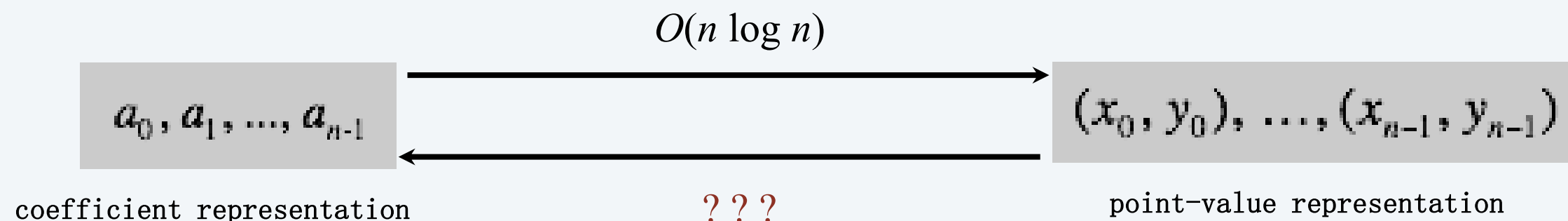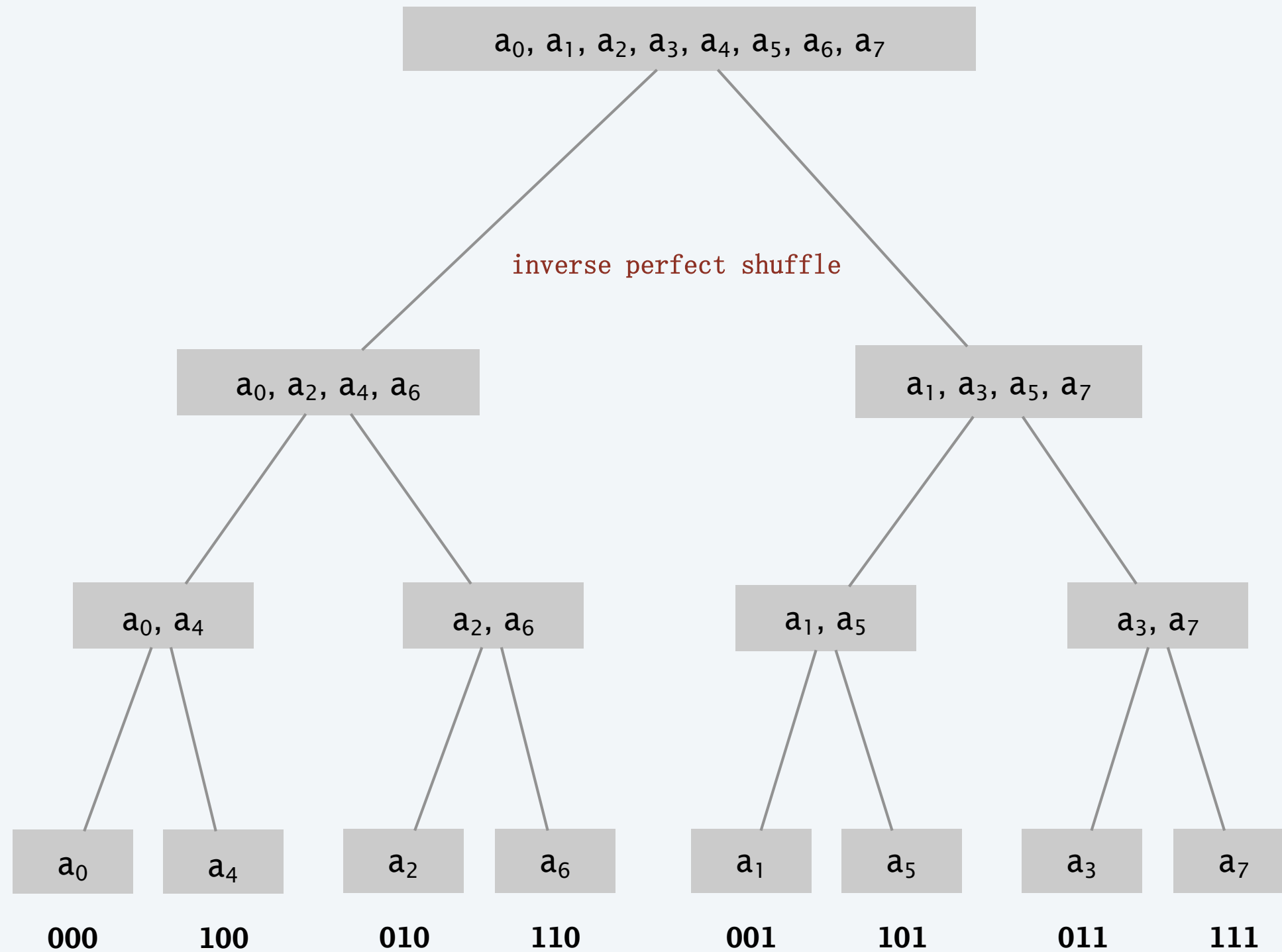
Inverse discrete Fourier transform

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.

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

$\uparrow$ $\uparrow$

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

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

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

```
Fₙ / √n is a unitary matrix
```

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

Claim. F_n and G_n are inverses.

Pf.

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

summation lemma (below)

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

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

Pf.

- If k is a multiple of n then $\omega^k = 1$ $\Rightarrow$ series sums to n.
- Each nth 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: $1 + \omega^k + \omega^{k(2)} + \ldots + \omega^{k(n-1)} = 0$ ® series sums to 0. ▪

Inverse FFT: implementation

Note. Need to divide result by n.

INVERSE-FFT $(n, y_0, y_1, y_2, \ldots, y_{n-1})$

IF $(n = 1)$ RETURN y_0.

$(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow$ INVERSE-FFT $(n/2, y_0, y_2, y_4, \ldots, y_{n-2})$.

$(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow$ INVERSE-FFT $(n/2, y_1, y_3, y_5, \ldots, y_{n-1})$.

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

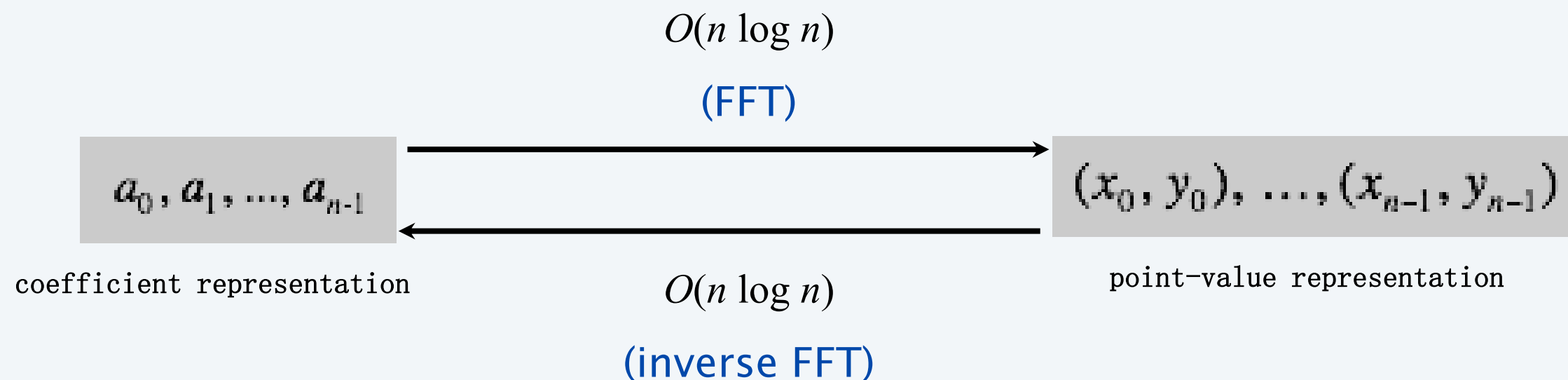$\omega^k \leftarrow e^{-2\pi i k/n}$.

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

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

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

switch roles of a_i and y_i

Theorem. The inverse FFT algorithm interpolates a degree $n-1$ polynomial given values at each of the nth roots of unity in $O(n \log n)$ steps.

assumes n is a power of 2

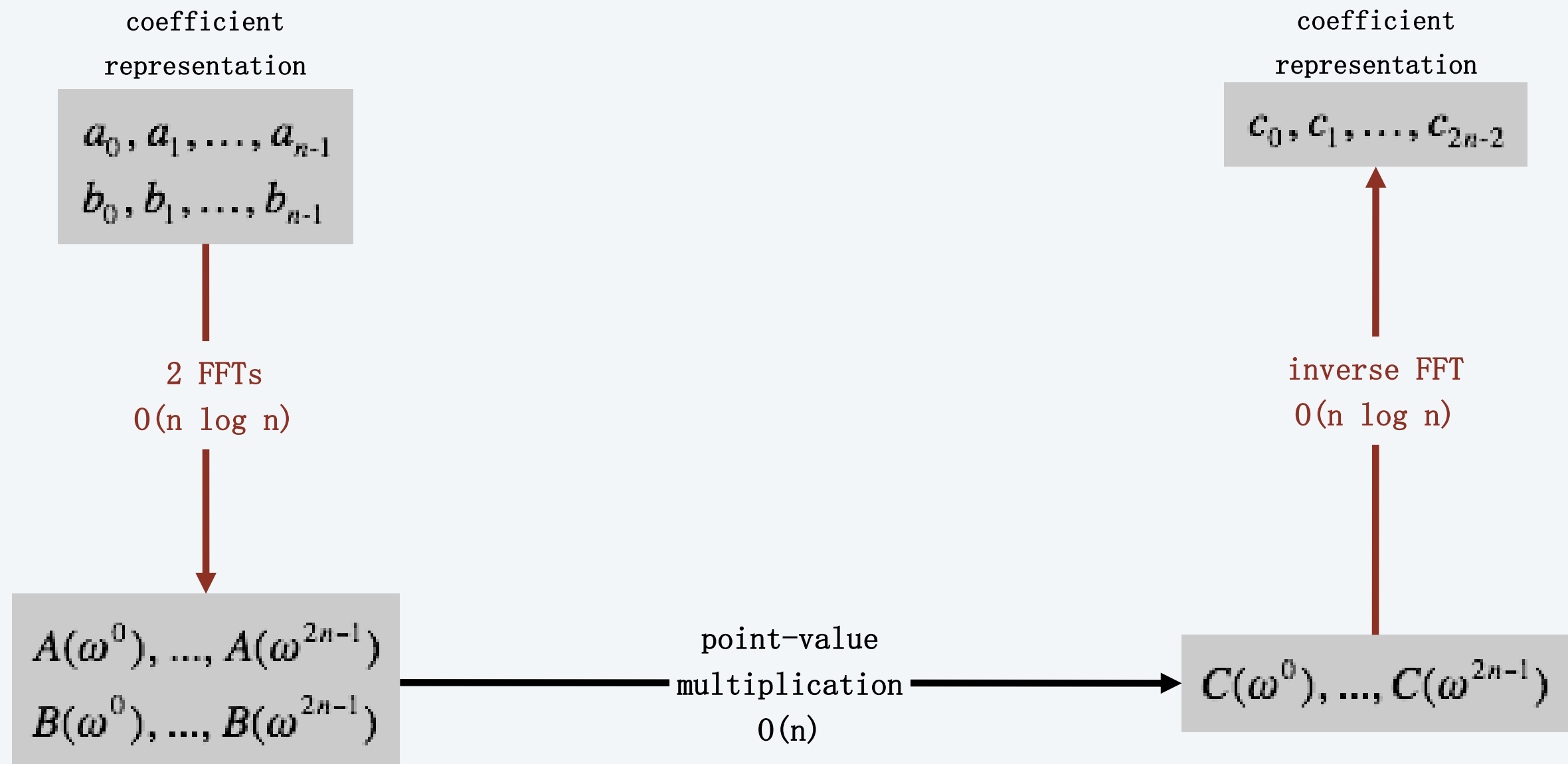$O(n \log n)$

(FFT)

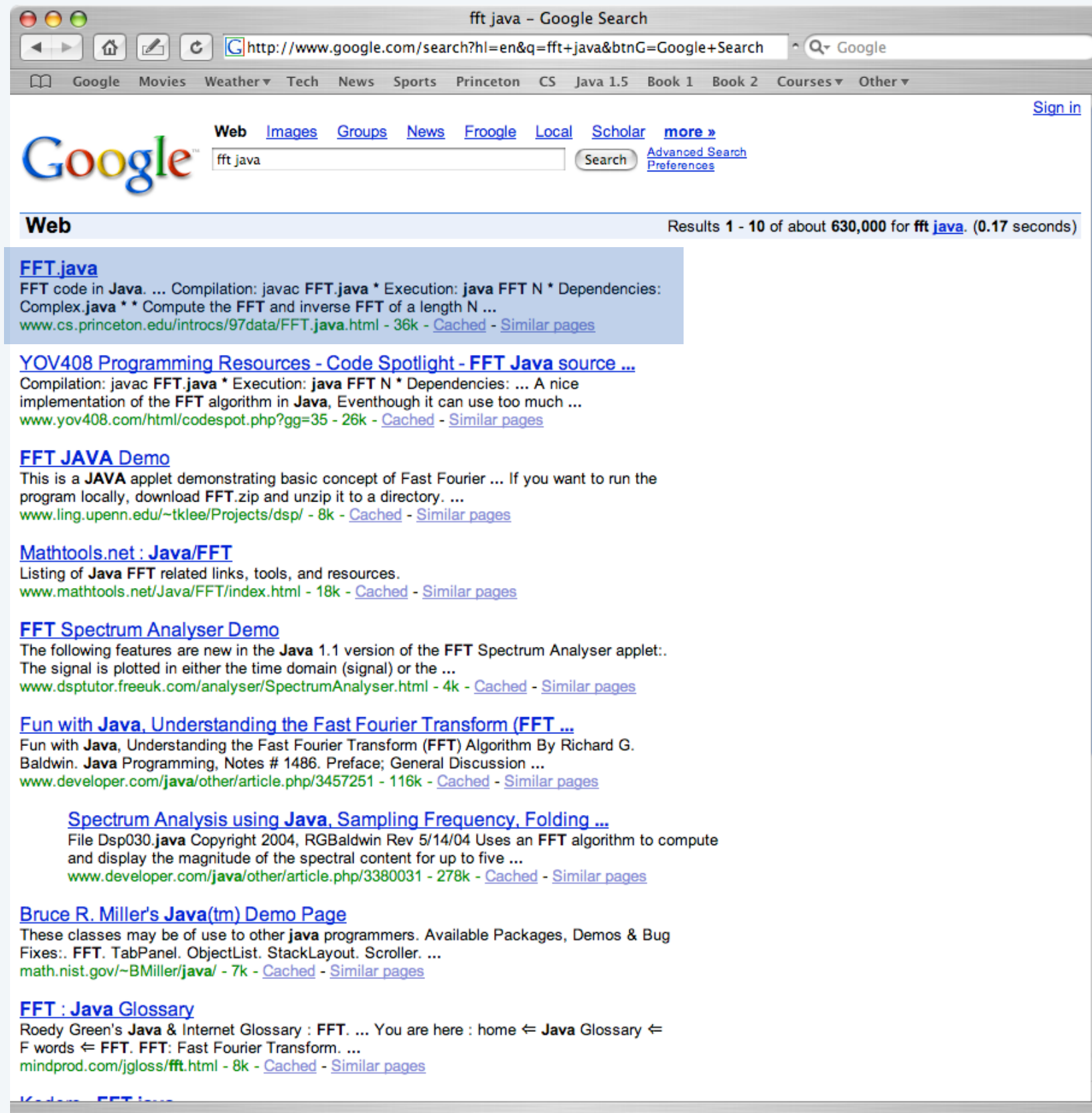$a_0, a_1, ..., a_{n-1}$ $\longrightarrow$ $(x_0, y_0), ..., (x_{n-1}, y_{n-1})$
$\longleftarrow$

coefficient representation   point-value representation

$O(n \log n)$

(inverse FFT)

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

coefficient
representation

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

coefficient
representation

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

2 FFTs
O(n log n)

inverse FFT
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)

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

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. Given two n-bit integers $a = a_{n-1} \dots a_1 a_0$ and $b = b_{n-1} \dots b_1 b_0$, compute their product $a \oplus b$.

Convolution algorithm.

- Form two polynomials.
- Note: $a = A(2)$, $b = B(2)$.
- Compute $C(x) = A(x) \oplus B(x)$.
- Evaluate $C(2) = a \oplus b$.
- 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.