

## DIVIDE AND CONQUER II

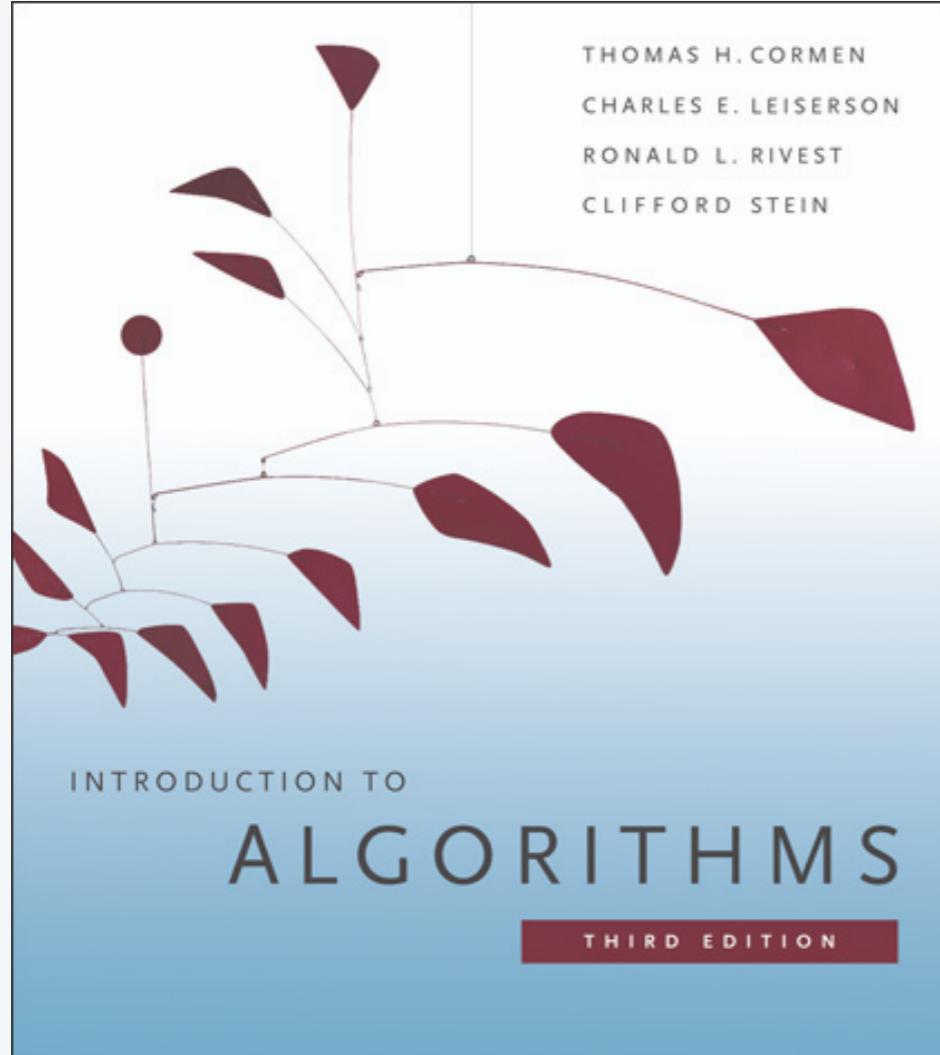
---

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



SECTIONS 4.3–4.6

## DIVIDE AND CONQUER II

---

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

# Master method

---

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

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

with  $T(0) = 0$  and  $T(1) = \Theta(1)$ .

## Terms.

- $a \geq 1$  is the (integer) number of subproblems.
- $b \geq 2$  is the (integer) factor by which the subproblem size decreases.
- $f(n)$  = work to divide and combine subproblems.

## Recursion tree.

- $k = \log_b n$  levels.
- $a^i$  = number of subproblems at level  $i$ .
- $n / b^i$  = size of subproblem at level  $i$ .

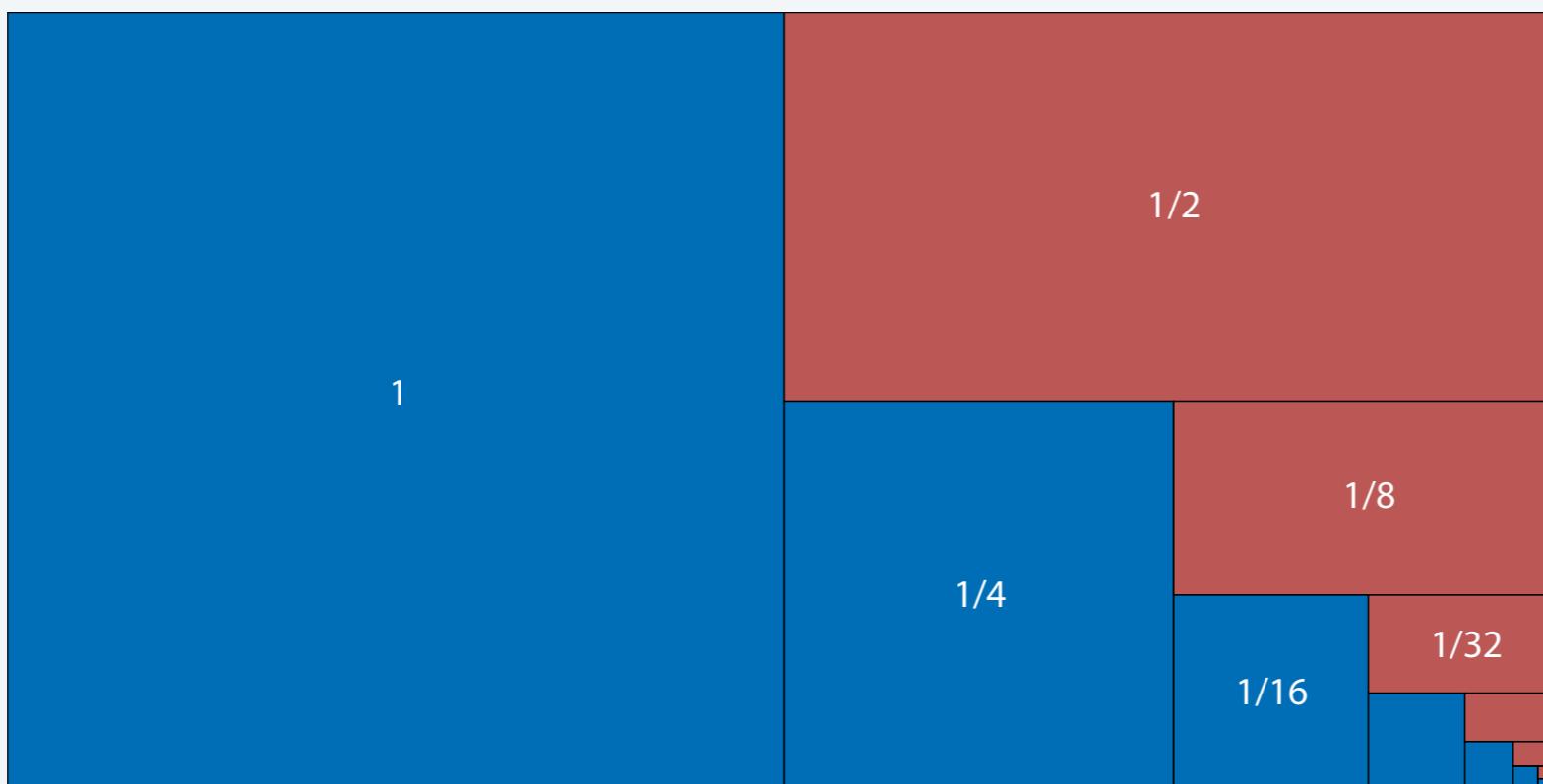
# Geometric series

---

**Fact 1.** For  $r \neq 1$ ,  $1 + r + r^2 + r^3 + \dots + r^{k-1} = \frac{1 - r^k}{1 - r}$

**Fact 2.** For  $r = 1$ ,  $1 + r + r^2 + r^3 + \dots + r^{k-1} = k$

**Fact 3.** For  $r < 1$ ,  $1 + r + r^2 + r^3 + \dots = \frac{1}{1 - r}$

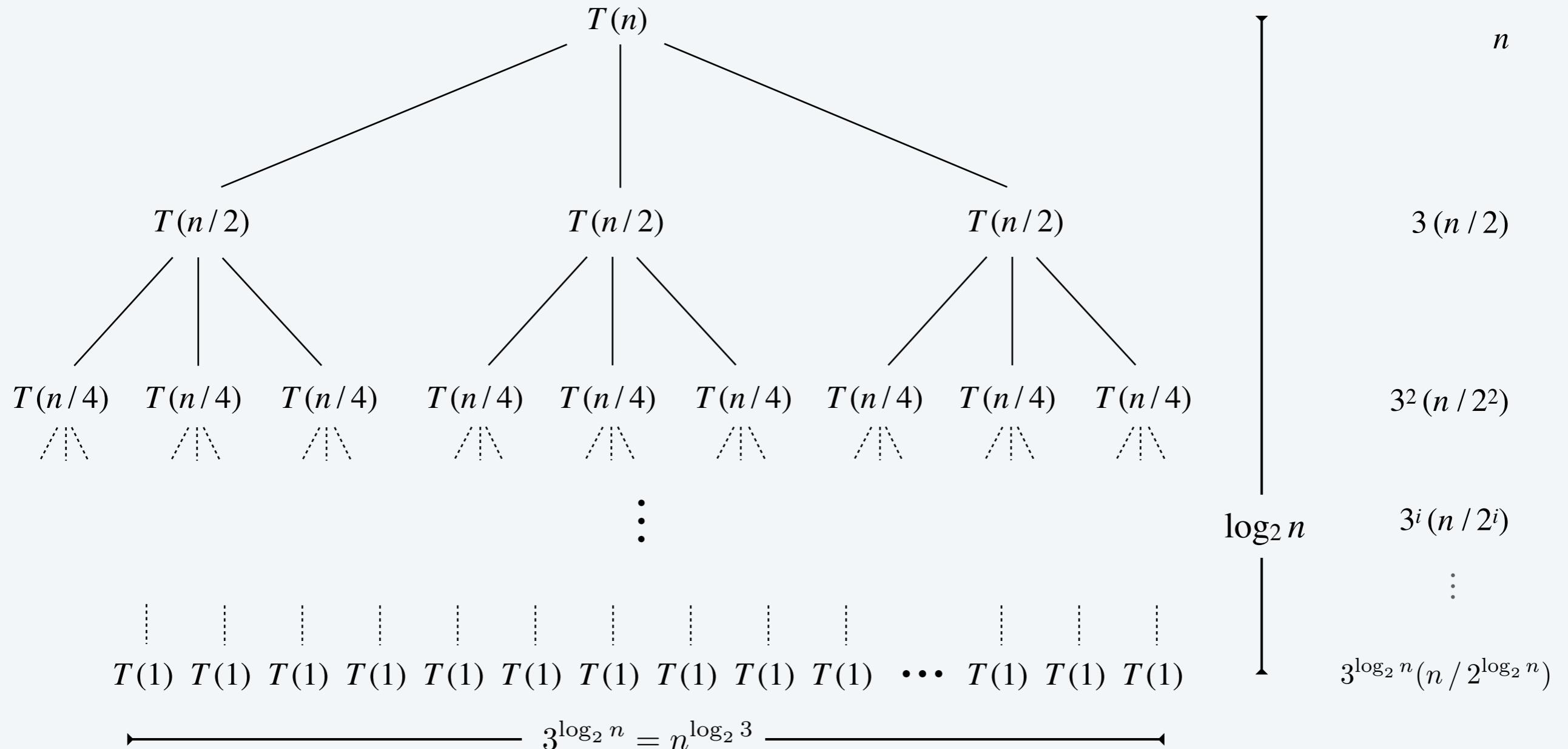


$$1 + 1/2 + 1/4 + 1/8 + \dots = 2$$

## 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  $T(n) = \Theta(n^{\log_2 3})$ .

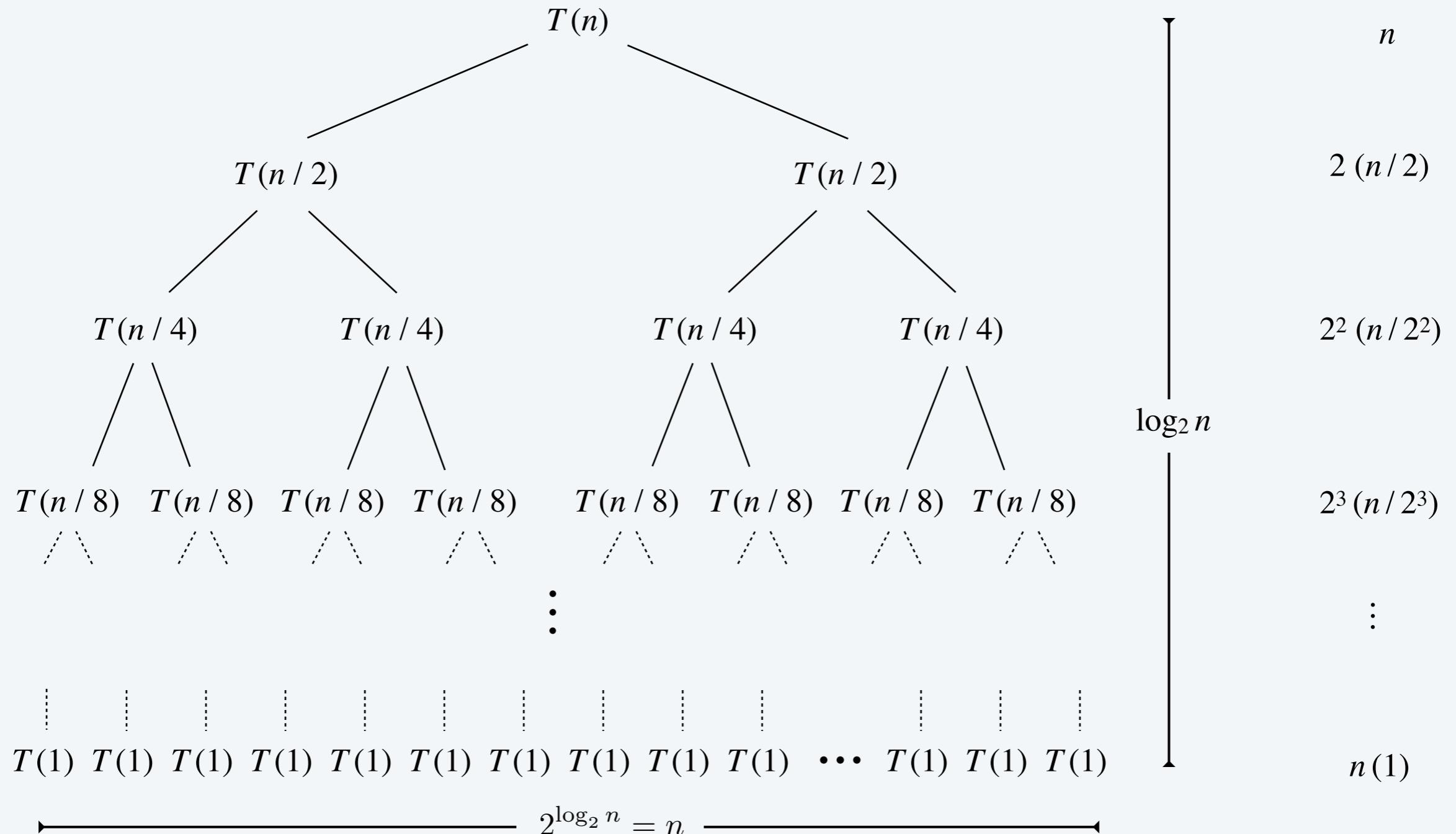


$$r = 3/2 > 1 \quad T(n) = (1 + r + r^2 + r^3 + \dots + r^{\log_2 n}) n = \frac{r^{1+\log_2 n} - 1}{r - 1} n = 3n^{\log_2 3} - 2n$$

## 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  $T(n) = \Theta(n \log n)$ .



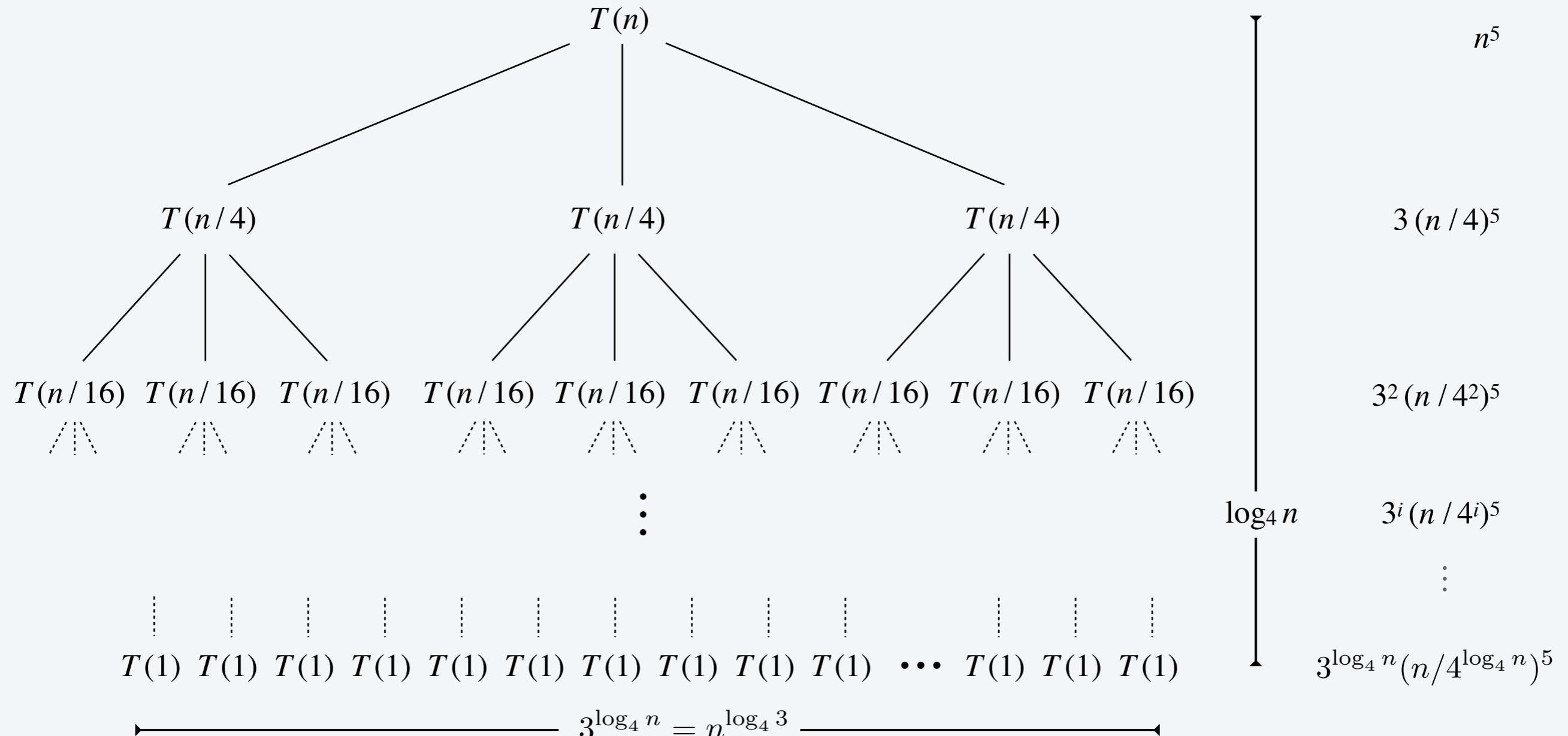
$$r = 1$$

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

## Case 3: total cost dominated by cost of root

---

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



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

## Master theorem

---

**Master theorem.** Suppose that  $T(n)$  is a function on the non-negative integers that satisfies the recurrence

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

with  $T(0) = 0$  and  $T(1) = \Theta(1)$ , where  $n/b$  means either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then,

**Case 1.** If  $f(n) = O(n^k)$  for some constant  $k < \log_b a$ , then  $T(n) = \Theta(n^{\log_b a})$ .

**Ex.**  $T(n) = 3 T(n / 2) + 5n$ .

- $a = 3$ ,  $b = 2$ ,  $f(n) = 5n$ ,  $k = 1$ ,  $\log_b a = 1.58\dots$
- $T(n) = \Theta(n^{\log_2 3})$ .

## Master theorem

---

**Master theorem.** Suppose that  $T(n)$  is a function on the non-negative integers that satisfies the recurrence

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

with  $T(0) = 0$  and  $T(1) = \Theta(1)$ , where  $n/b$  means either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then,

**Case 2.** If  $f(n) = \Theta(n^k \log^p n)$  for  $p \geq 0$  and  $k = \log_b a$ , then  $T(n) = \Theta(n^k \log^{p+1} n)$ .

**Ex.**  $T(n) = 2 T(n / 2) + 17 n \log_2 n$ .

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

## Master theorem

---

**Master theorem.** Suppose that  $T(n)$  is a function on the non-negative integers that satisfies the recurrence

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

with  $T(0) = 0$  and  $T(1) = \Theta(1)$ , where  $n/b$  means either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then,

**Case 3.** If  $f(n) = \Omega(n^k)$  for some constant  $k > \log_b a$ , and if  $a f(n/b) \leq c f(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ .

**Ex.**  $T(n) = 3 T(n/2) + n^2$ .

“regularity condition”  
holds if  $f(n) = \Theta(n^k)$

- $a = 3, b = 2, f(n) = n^2, k = 2, \log_b a = 1.58\dots$
- Regularity condition:  $3(n/2)^2 \leq c n^2$  for  $c = 3/4$ .
- $T(n) = \Theta(n^2)$ .

## Master theorem

---

**Master theorem.** Suppose that  $T(n)$  is a function on the non-negative integers that satisfies the recurrence

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

with  $T(0) = 0$  and  $T(1) = \Theta(1)$ , where  $n/b$  means either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then,

**Case 1.** If  $f(n) = O(n^k)$  for some constant  $k < \log_b a$ , then  $T(n) = \Theta(n^{\log_b a})$ .

**Case 2.** If  $f(n) = \Theta(n^k \log^p n)$  for  $p \geq 0$  and  $k = \log_b a$ , then  $T(n) = \Theta(n^k \log^{p+1} n)$ .

**Case 3.** If  $f(n) = \Omega(n^k)$  for some constant  $k > \log_b a$ , and if  $a f(n/b) \leq c f(n)$  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.

# Master theorem need not apply

---

## Gaps in master theorem.

- Number of subproblems must be a constant.

$$T(n) = \textcircled{n} T(n/2) + n^2$$

- Number of subproblems must be  $\geq 1$ .

$$T(n) = \frac{1}{\textcircled{2}} T(n/2) + n^2$$

- Non-polynomial separation between  $f(n)$  and  $n^{\log_b a}$ .

$$T(n) = 2 T(n/2) + \frac{n}{\log n}$$

- $f(n)$  is not positive.

$$T(n) = 2 T(n/2) \textcircled{-} n^2$$

- Regularity condition does not hold.

$$T(n) = T(n/2) + \textcircled{n(2 - \cos n)}$$

## Akra–Bazzi theorem

---

**Desiderata.** Generalizes master theorem to divide-and-conquer algorithms where subproblems have substantially different sizes.

**Theorem.** [Akra–Bazzi] Given constants  $a_i > 0$  and  $0 < b_i \leq 1$ , functions  $h_i(n) = O(n / \log^2 n)$  and  $g(n) = O(n^c)$ , if the function  $T(n)$  satisfies the recurrence:

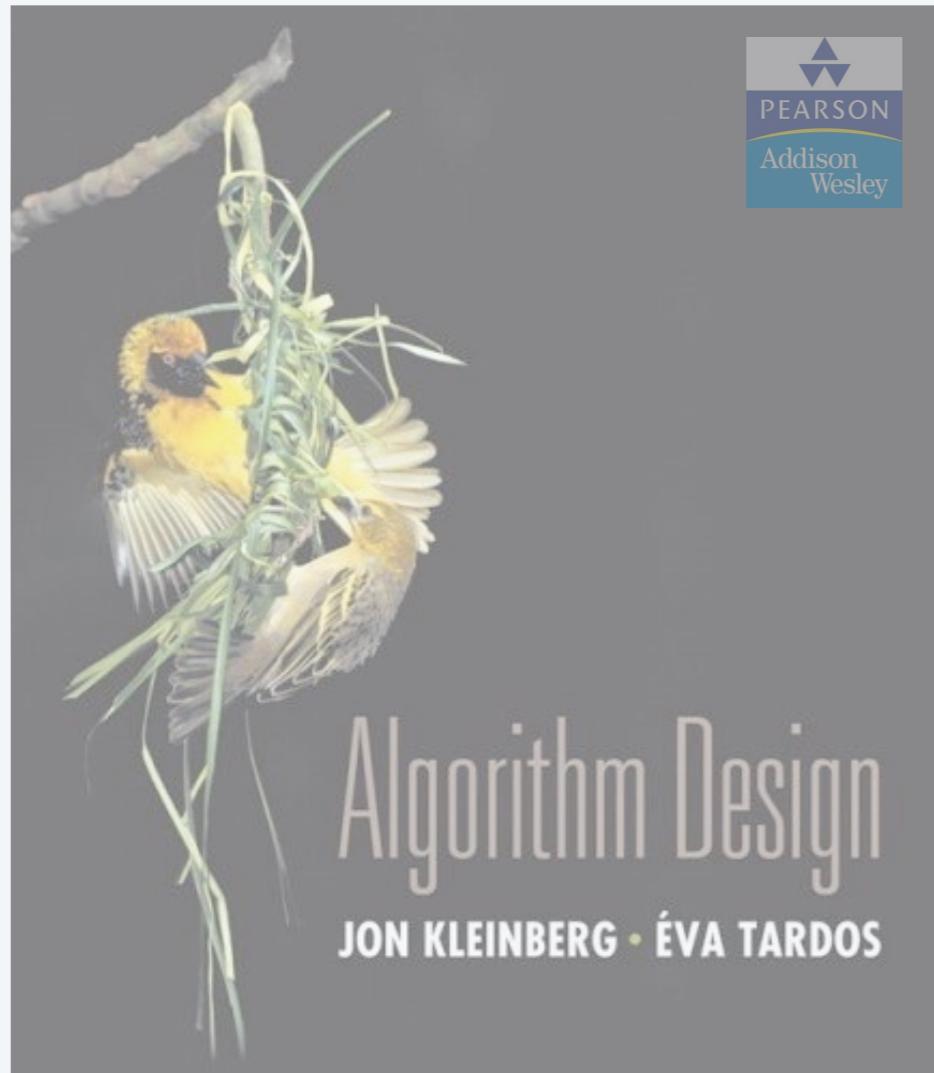
$$T(n) = \sum_{i=1}^k a_i T(b_i n + h_i(n)) + g(n)$$

↑  
a<sub>i</sub> subproblems  
of size b<sub>i</sub> n      ↑  
small perturbation to handle  
floors and ceilings

Then  $T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right)$  where  $p$  satisfies  $\sum_{i=1}^k a_i b_i^p = 1$ .

**Ex.**  $T(n) = 7/4 T(\lfloor n/2 \rfloor) + T(\lceil 3/4 n \rceil) + n^2$ , with  $T(0) = 0$  and  $T(1) = 1$ .

- $a_1 = 7/4, b_1 = 1/2, a_2 = 1, b_2 = 3/4 \Rightarrow p = 2$ .
- $h_1(n) = \lfloor 1/2 n \rfloor - 1/2 n, h_2(n) = \lceil 3/4 n \rceil - 3/4 n$ .
- $g(n) = n^2 \Rightarrow T(n) = \Theta(n^2 \log n)$ .



## DIVIDE AND CONQUER II

---

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

# Integer addition

---

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

Subtraction. Given two  $n$ -bit integers  $a$  and  $b$ , compute  $a - b$ .

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

1	1	1	1	1	1	0	1
	1	1	0	1	0	1	0
+	0	1	1	1	1	1	0
	1	0	1	0	1	0	0
	1	0	1	0	1	0	0

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

# Integer multiplication

**Multiplication.** Given two  $n$ -bit integers  $a$  and  $b$ , compute  $a \times b$ .

Grade-school algorithm.  $\Theta(n^2)$  bit operations.

**Conjecture.** [Kolmogorov 1952] Grade-school algorithm is optimal.

**Theorem.** [Karatsuba 1960] Conjecture is wrong.

# Divide-and-conquer multiplication

To multiply two  $n$ -bit integers  $x$  and  $y$ :

- Divide  $x$  and  $y$  into low- and high-order bits.
- Multiply four  $\frac{1}{2}n$ -bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$\begin{array}{ll} a = \lfloor x / 2^m \rfloor & b = x \bmod 2^m \\ c = \lfloor y / 2^m \rfloor & d = y \bmod 2^m \end{array}$$

| ← use bit shifting  
to compute 4 terms

$$(2^m a + b) (2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

1                  2                  3                  4

*Ex.*  $x = \underbrace{1 0 0 0}_{a} \underbrace{1 1 0 1}_{b}$      $y = \underbrace{1 1 1}_{c} \underbrace{0 0 0 0 1}_{d}$

# Divide-and-conquer multiplication

---

**MULTIPLY** ( $x, y, n$ )

**IF** ( $n = 1$ )

**RETURN**  $x \times y$ .

**ELSE**

$m \leftarrow \lceil n / 2 \rceil$ .

$a \leftarrow \lfloor x / 2^m \rfloor$ ;  $b \leftarrow x \bmod 2^m$ .

$c \leftarrow \lfloor y / 2^m \rfloor$ ;  $d \leftarrow y \bmod 2^m$ .

$e \leftarrow \text{MULTIPLY} (a, c, m)$ .

$f \leftarrow \text{MULTIPLY} (b, d, m)$ .

$g \leftarrow \text{MULTIPLY} (b, c, m)$ .

$h \leftarrow \text{MULTIPLY} (a, d, m)$ .

**RETURN**  $2^{2m} e + 2^m (g + h) + f$ .

## Divide-and-conquer multiplication analysis

---

**Proposition.** The divide-and-conquer multiplication algorithm requires  $\Theta(n^2)$  bit operations to multiply two  $n$ -bit integers.

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

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

# Karatsuba trick

---

To compute middle term  $bc + ad$ , use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

middle term  
↓

$$\begin{aligned} (2^m a + b)(2^m c + d) &= 2^{2m} ac + 2^m (bc + ad) + bd \\ &= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd \end{aligned}$$

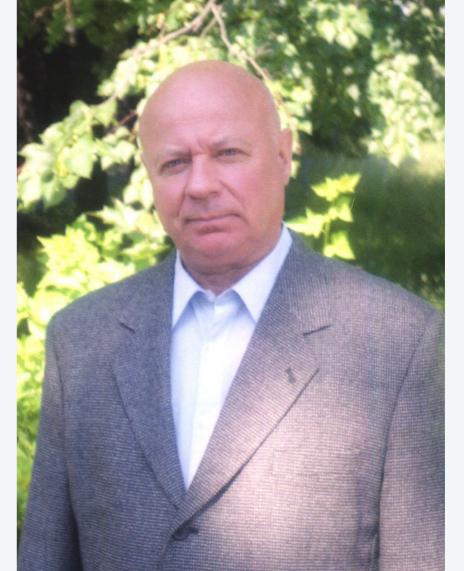
1

1

3

2

3



Bottom line. Only three multiplications of  $n/2$ -bit integers.

# Karatsuba multiplication

---

**KARATSUBA-MULTIPLY** ( $x, y, n$ )

**IF** ( $n = 1$ )

**RETURN**  $x \times y$ .

**ELSE**

$m \leftarrow \lceil n / 2 \rceil$ .

$a \leftarrow \lfloor x / 2^m \rfloor$ ;  $b \leftarrow x \bmod 2^m$ .

$c \leftarrow \lfloor y / 2^m \rfloor$ ;  $d \leftarrow y \bmod 2^m$ .

$e \leftarrow \text{KARATSUBA-MULTIPLY} (a, c, m)$ .

$f \leftarrow \text{KARATSUBA-MULTIPLY} (b, d, m)$ .

$g \leftarrow \text{KARATSUBA-MULTIPLY} (a - b, c - d, m)$ .

**RETURN**  $2^{2m} e + 2^m (e + f - g) + f$ .

## Karatsuba analysis

---

**Proposition.** Karatsuba's algorithm requires  $O(n^{1.585})$  bit operations to multiply two  $n$ -bit integers.

**Pf.** Apply Case 1 of the master theorem to the recurrence:

$$T(n) = 3T(n/2) + \Theta(n) \implies T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

**Practice.** Faster than grade-school algorithm for about 320–640 bits.

# Integer arithmetic reductions

---

Integer multiplication. Given two  $n$ -bit integers, compute their product.

problem	arithmetic	running time
integer multiplication	$a \times b$	$M(n)$
integer division	$a / b, a \bmod b$	$\Theta(M(n))$
integer square	$a^2$	$\Theta(M(n))$
integer square root	$\lfloor \sqrt{a} \rfloor$	$\Theta(M(n))$

integer arithmetic problems with the same complexity  $M(n)$  as integer multiplication

# History of asymptotic complexity of integer multiplication

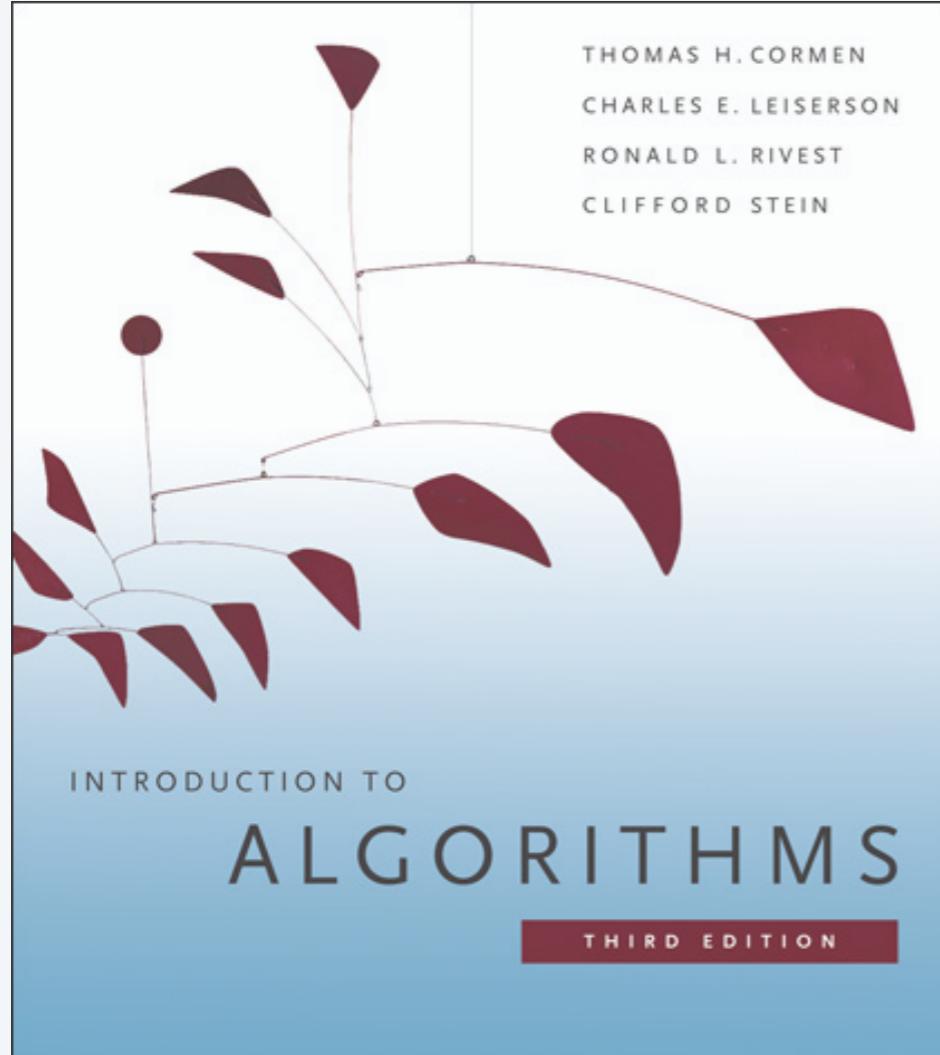
year	algorithm	order of growth
?	<b>brute force</b>	$\Theta(n^2)$
1962	<b>Karatsuba-Ofman</b>	$\Theta(n^{1.585})$
1963	<b>Toom-3, Toom-4</b>	$\Theta(n^{1.465}), \Theta(n^{1.404})$
1966	<b>Toom-Cook</b>	$\Theta(n^{1+\varepsilon})$
1971	<b>Schönhage-Strassen</b>	$\Theta(n \log n \log \log n)$
2007	<b>Fürer</b>	$n \log n 2^{O(\log^* n)}$
?	?	$\Theta(n)$

number of bit operations to multiply two  $n$ -bit integers

used in Maple, Mathematica, gcc, cryptography, ...

Remark. GNU Multiple Precision Library uses one of five different algorithms depending on size of operands.





## SECTION 4.2

# DIVIDE AND CONQUER II

---

- ▶ *master theorem*
- ▶ *integer multiplication*
- ▶ *matrix multiplication*
- ▶ *convolution and FFT*

## Dot product

---

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

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

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

$$a = [ .70 \quad .20 \quad .10 ]$$

$$b = [ .30 \quad .40 \quad .30 ]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

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

# Matrix multiplication

---

**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

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

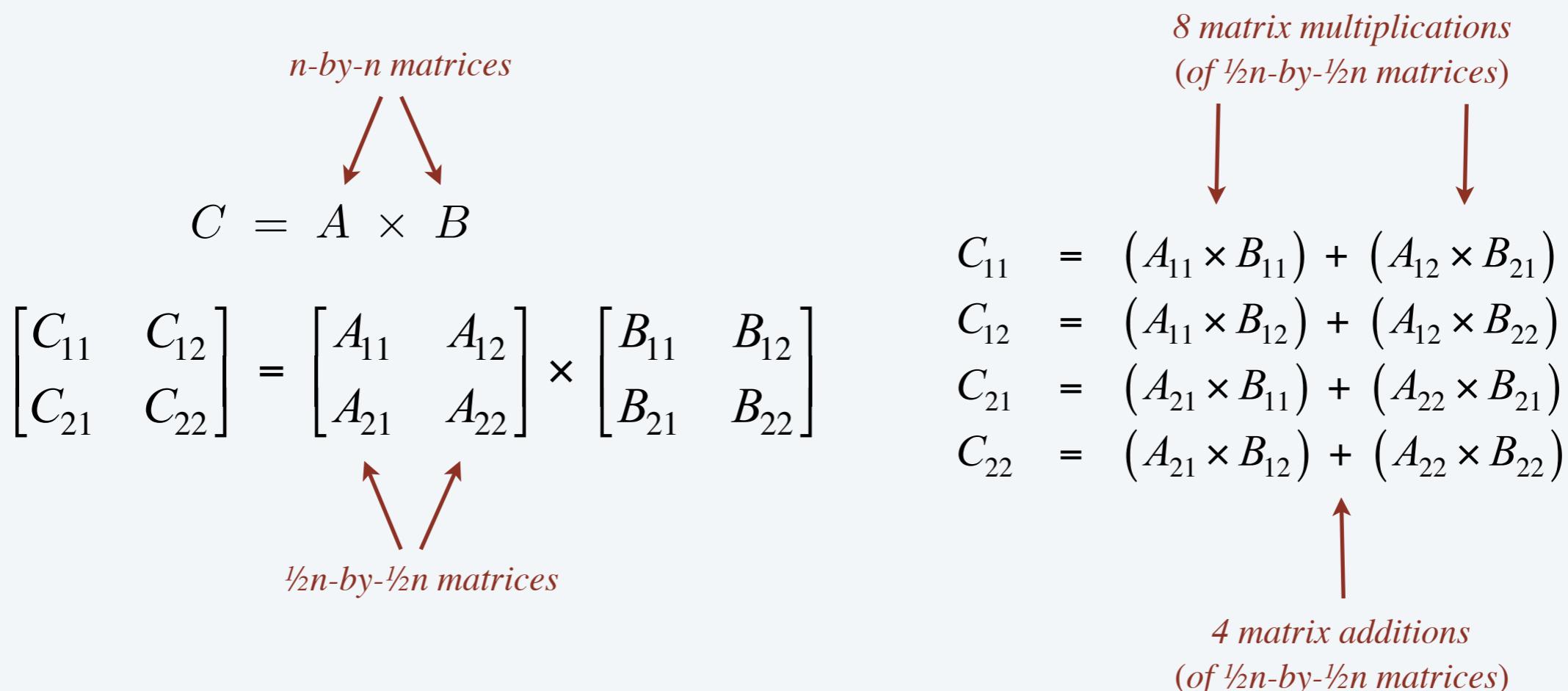
$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526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

# Block 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.



**Running time.** Apply case 1 of the master theorem.

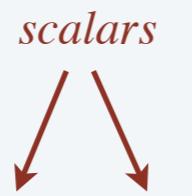
$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

## Strassen's trick

**Key idea.** Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

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

*scalars*



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

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

$$C_{21} = P_3 + P_4$$

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

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

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

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

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

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

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

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



*7 scalar multiplications*

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

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

$$= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark$$

# Strassen's trick

*n-by-n*      *½n-by-½n matrix*  
**Key idea.** Can multiply two ~~2-by-2~~ matrices via 7 ~~scalar~~ multiplications  
 (plus 11 additions and 7 subtractions).

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

*½n-by-½n matrices*

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

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

$$C_{21} = P_3 + P_4$$

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

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

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

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

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

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

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

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



*7 matrix multiplications  
 (of ½n-by-½n matrices)*

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

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

$$= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark$$

# Strassen's algorithm

assume n is a power of 2

**STRASSEN**( $n, A, B$ )

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

Partition  $A$  and  $B$  into  $\frac{1}{2}n$ -by- $\frac{1}{2}n$  blocks.

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

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

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

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

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

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

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

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

$C_{12} = P_1 + P_2$ .

$C_{21} = P_3 + P_4$ .

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

RETURN  $C$ .

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

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

# Analysis of Strassen's algorithm

---

**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:

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

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

**A.** Could pad matrices with zeros.

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

## Strassen's algorithm: practice

---

### Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm when  $n$  is “small.”

Common misperception. “*Strassen’s algorithm 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.

# Linear algebra reductions

---

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

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

numerical linear algebra problems with the same complexity  $MM(n)$  as matrix multiplication

## Fast matrix multiplication: theory

---

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–Kerr 1971]

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

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

A. Unknown.

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

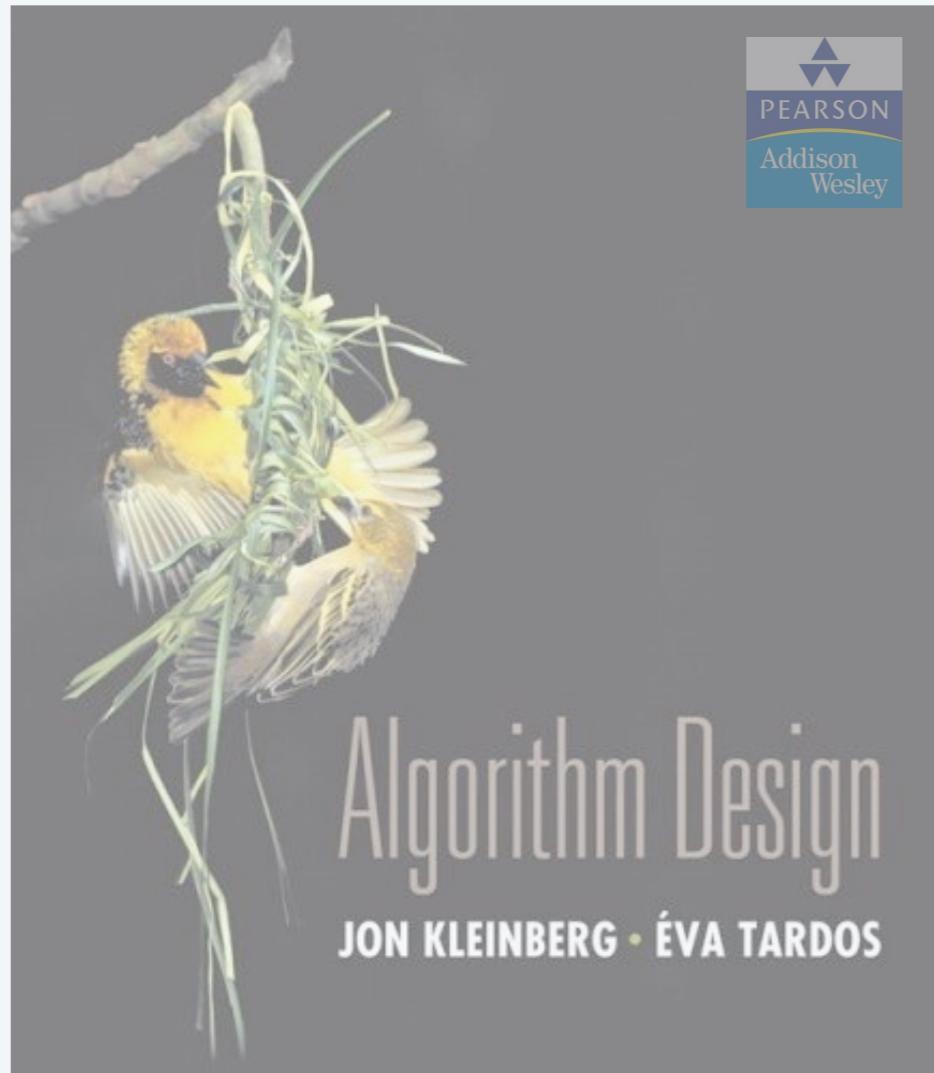
Begun, the decimal wars have. [Pan, Bini et al., Schönhage, ...]

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

number of floating-point operations to multiply two n-by-n matrices



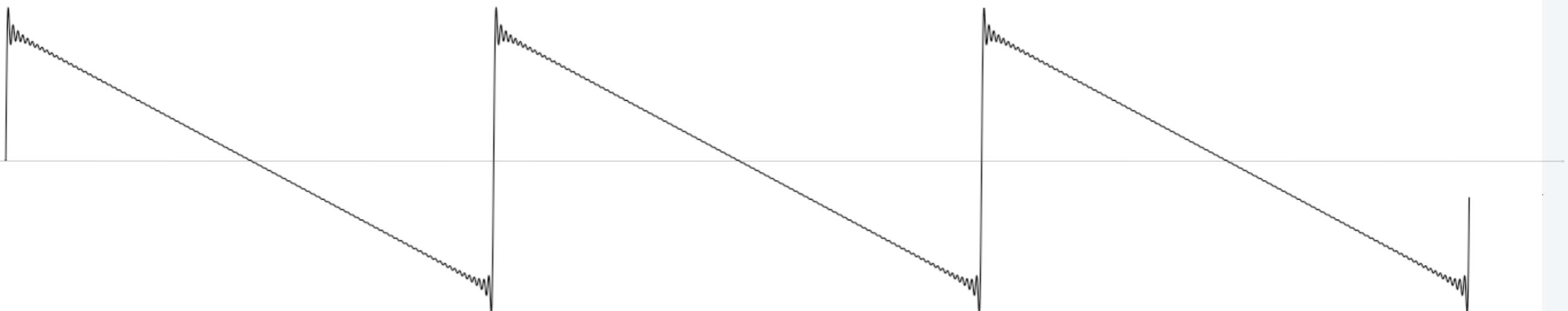
## DIVIDE AND CONQUER II

---

- ▶ *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.



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

## 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

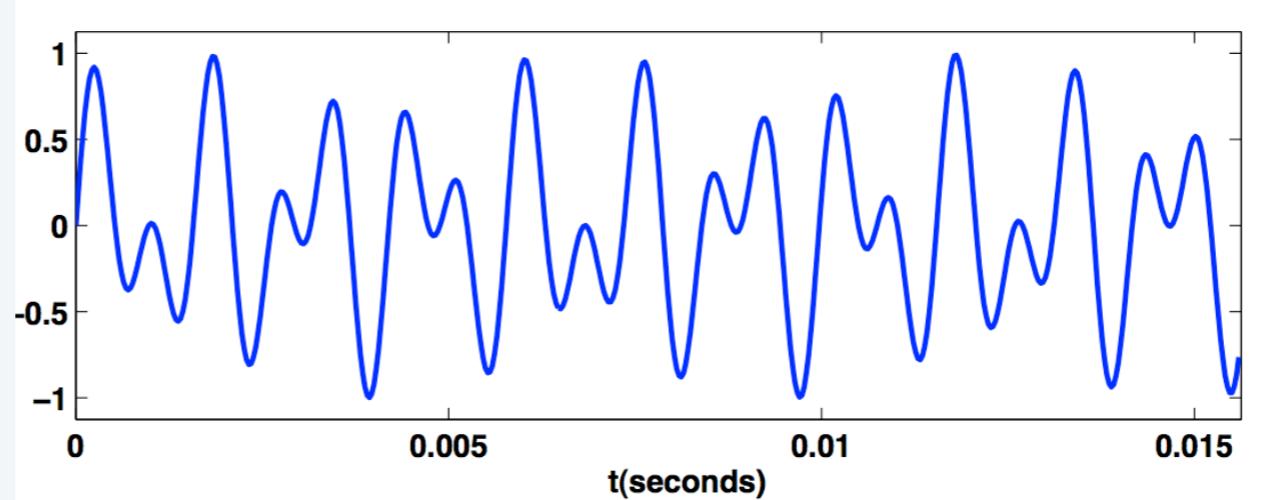
---

Signal. [touch tone button 1]

$$y(t) = \frac{1}{2} \sin(2\pi \cdot 697 t) + \frac{1}{2} \sin(2\pi \cdot 1209 t)$$

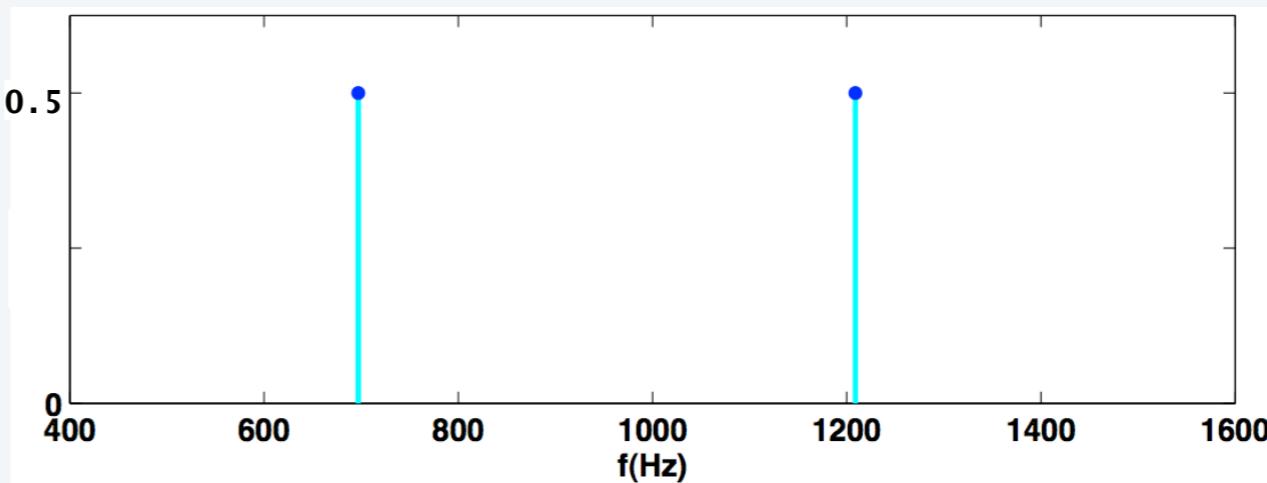
Time domain.

sound  
pressure



Frequency domain.

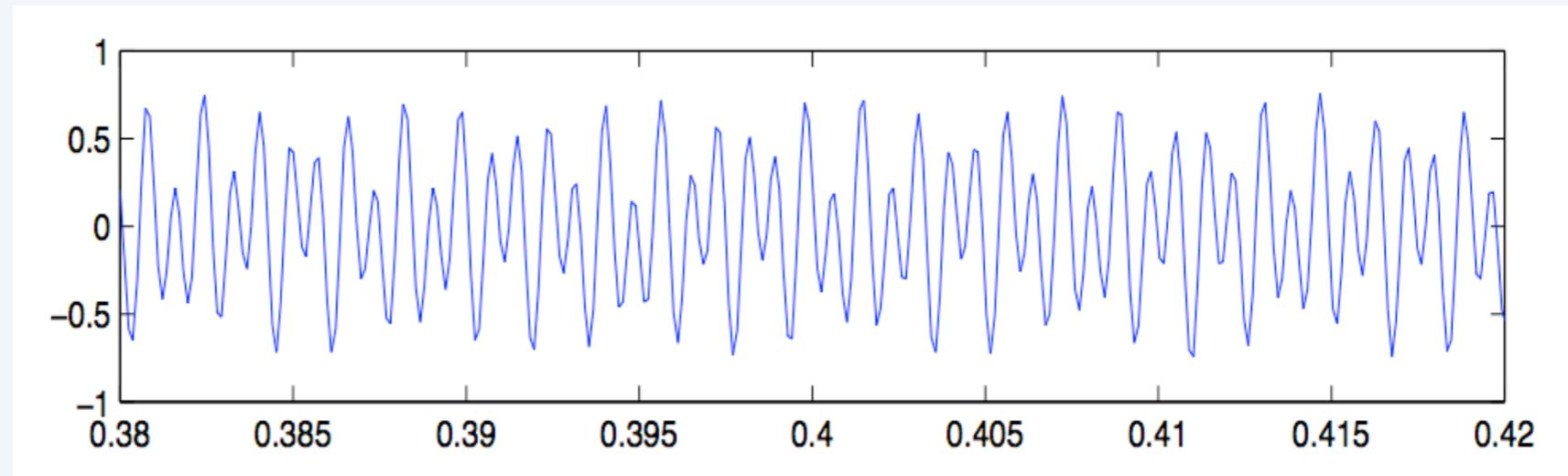
amplitude



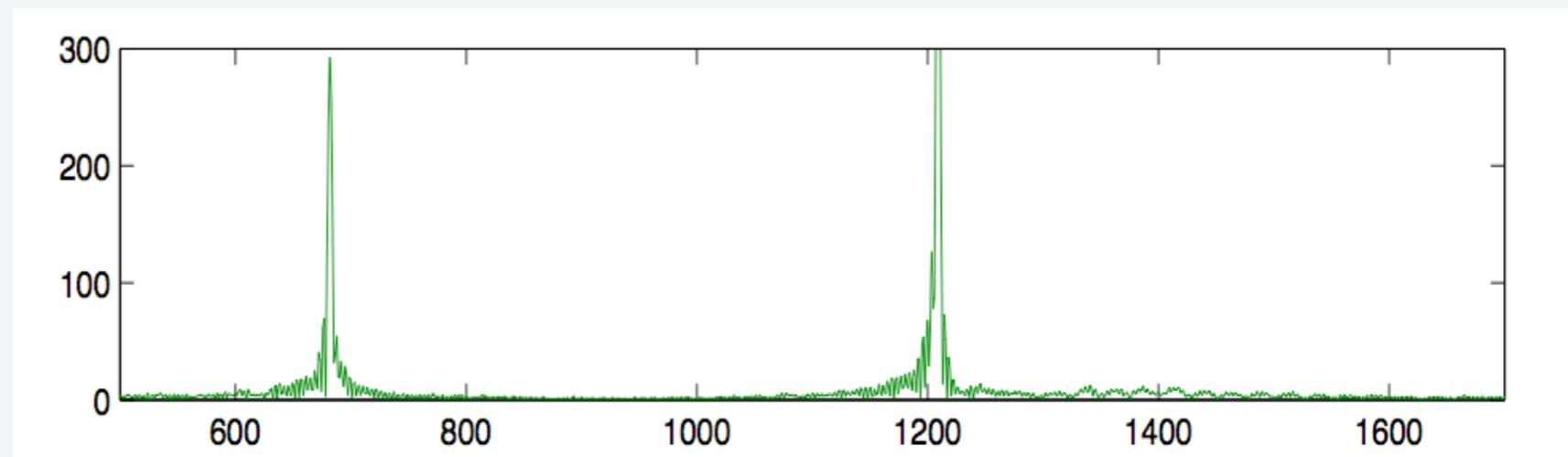
# Time domain vs. frequency domain

---

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



# Fast Fourier transform

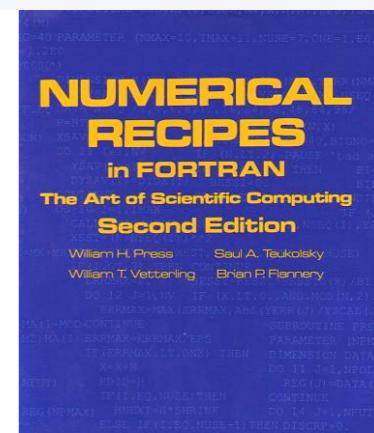
---

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

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

*“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

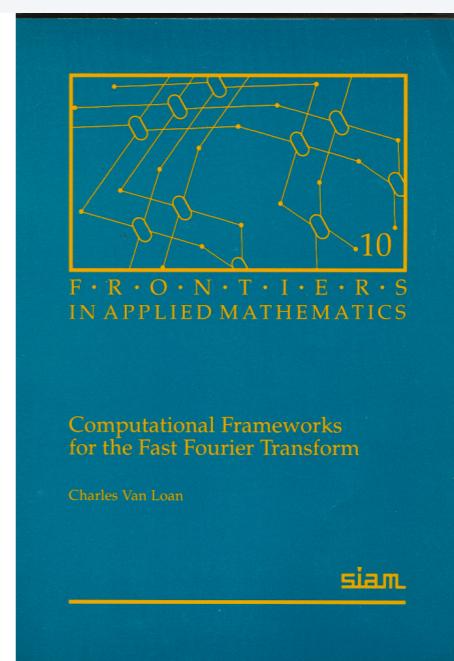
---

## 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]

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$$

Add.  $O(n)$  arithmetic operations.

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

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

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

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

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

# A modest Ph.D. dissertation title

---

DEMONSTRATIO NOVA  
THEOREMATIC  
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  $x^m + Ax^{m-1} + Bx^{m-2} + \dots + M = 0$ , ita vt  $m$  sit numerus integer positivus. 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  $\alpha, \beta, \gamma$  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:

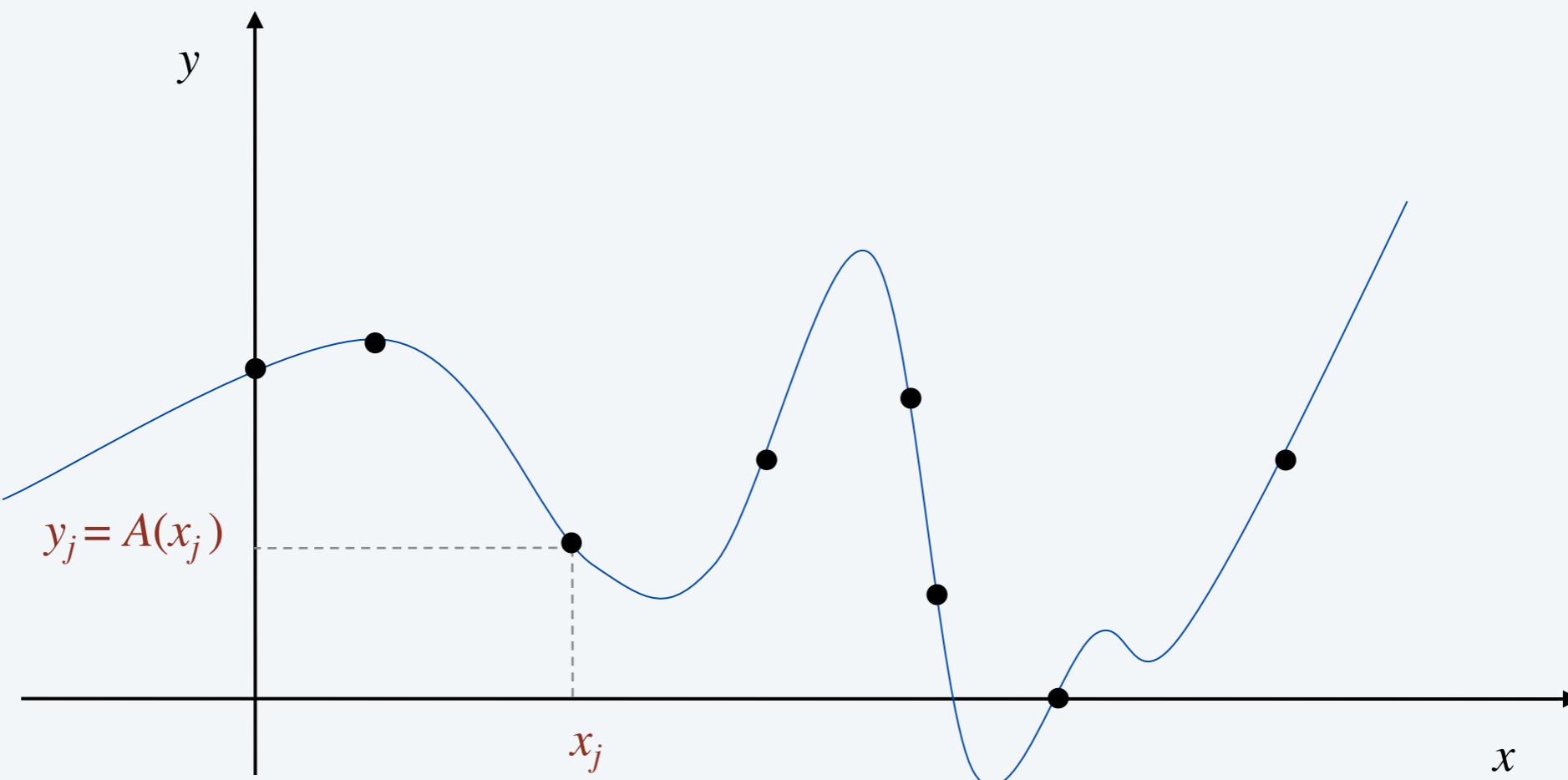
**“New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree.”**

# 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]

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

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

Add.  $O(n)$  arithmetic operations.

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

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

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

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

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

# Converting between two representations

---

Tradeoff. Fast evaluation **or** fast multiplication. We want both!

representation	multiply	evaluate
<b>coefficient</b>	$O(n^2)$	$O(n)$
<b>point-value</b>	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations  $\Rightarrow$  all ops fast.



## Converting between two representations: brute force

---

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

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

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

## Converting between two representations: brute force

**Point-value  $\Rightarrow$  coefficient.** Given  $n$  distinct points  $x_0, \dots, x_{n-1}$  and values  $y_0, \dots, y_{n-1}$ , find unique polynomial  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , that has given values at given points.

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



Vandermonde matrix is invertible iff  $x_i$  distinct

**Running time.**  $O(n^3)$  for Gaussian elimination.



or  $O(n^{2.3727})$  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 up polynomial into even and odd powers.

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

## Coefficient to point-value representation: intuition

---

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

**Divide.** Break up polynomial into even and odd powers.

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

**Intuition.** Choose two points to be  $\pm 1$ .

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



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

## Coefficient to point-value representation: intuition

---

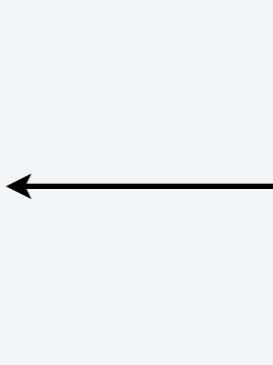
**Coefficient  $\Rightarrow$  point-value.** Given a polynomial  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , evaluate it at  $n$  distinct points  $x_0, \dots, x_{n-1}$ . ← we get to choose which ones!

**Divide.** Break up polynomial into even and odd powers.

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

**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  $\leq n$  at 4 points by evaluating two polynomials of degree  $\leq \frac{1}{2}n$  at 2 point.

# Discrete Fourier transform

---

Coefficient  $\Rightarrow$  point-value. Given a polynomial  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , evaluate it at  $n$  distinct points  $x_0, \dots, x_{n-1}$ .

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

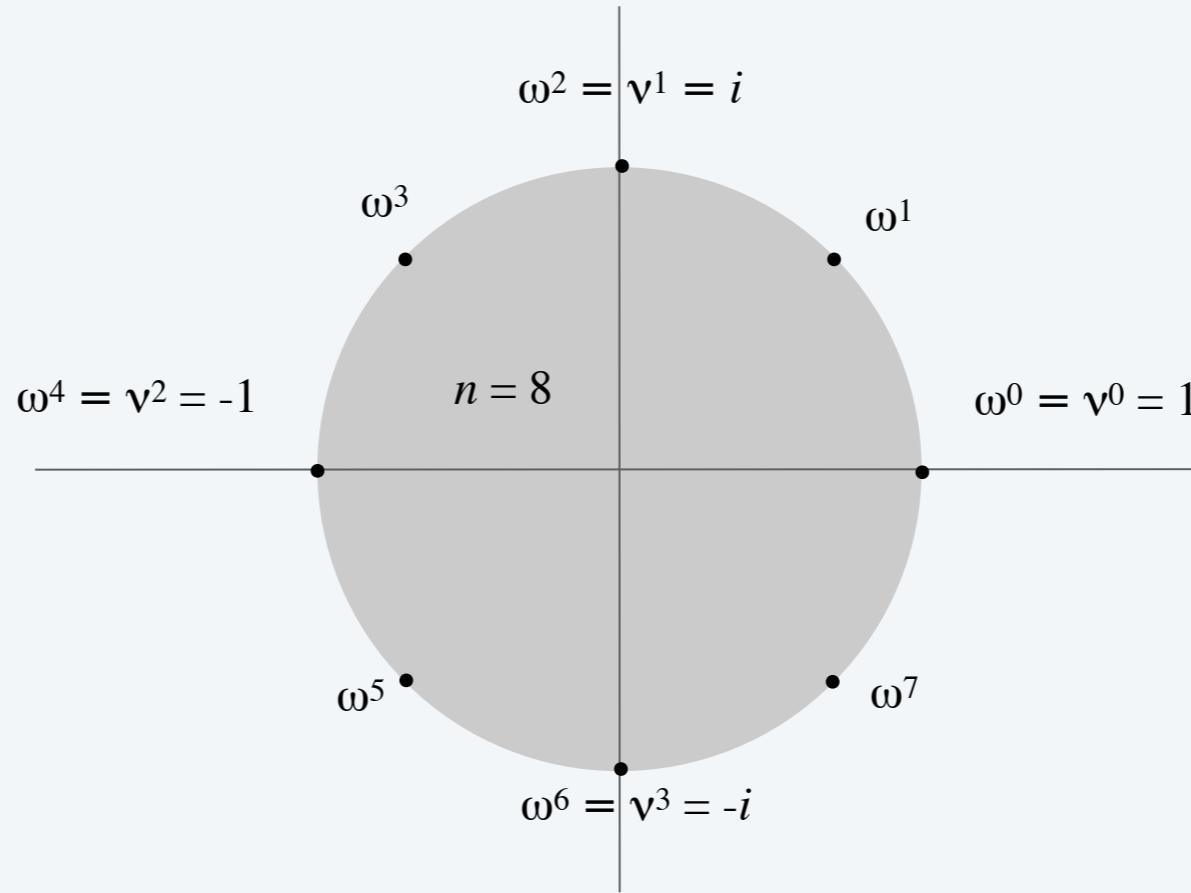
---

**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:  $\omega^0, \omega^1, \dots, \omega^{n-1}$  where  $\omega = e^{2\pi i / n}$ .

**Pf.**  $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$ .

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



# Fast Fourier transform

---

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

**Divide.** Break up polynomial into even and odd powers.

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

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

**Combine.**

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

$$\begin{array}{ccc} v^k = (\omega^k)^2 & & \\ \swarrow & & \\ v^k = (\omega^{k+\frac{1}{2}n})^2 & & \omega^{k+\frac{1}{2}n} = -\omega^k \end{array}$$

# FFT: implementation

---

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

**IF** ( $n = 1$ ) **RETURN**  $a_0$ .

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

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

**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**  $(y_0, y_1, y_2, \dots, y_{n-1})$ .

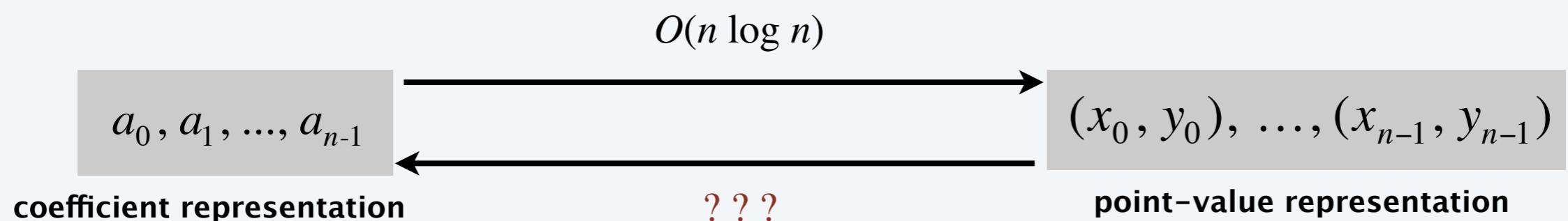
## FFT: summary

---

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

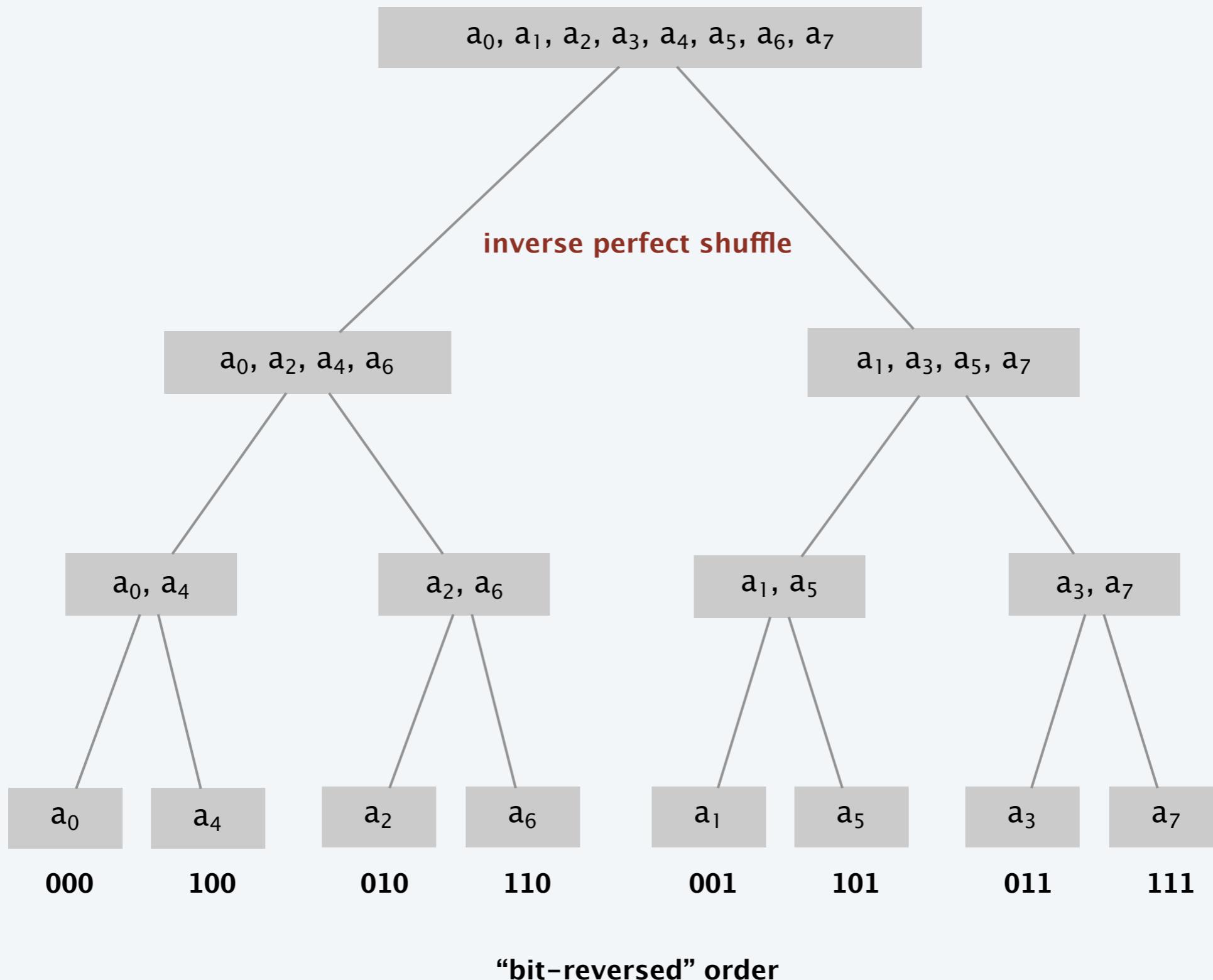
Pf.  $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$

assumes  $n$  is a power of 2



# FFT: recursion tree

---



# Inverse discrete Fourier transform

---

**Point-value  $\Rightarrow$  coefficient.** Given  $n$  distinct points  $x_0, \dots, x_{n-1}$  and values  $y_0, \dots, y_{n-1}$ , find unique polynomial  $a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ , that has given values at given points.

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

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  $n^{\text{th}}$  root of unity (and divide the result by  $n$ ).

## Inverse FFT: proof of correctness

---

**Claim.**  $F_n$  and  $G_n$  are inverses.

Pf.

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

summation lemma (below)

**Summation lemma.** Let  $\omega$  be a principal  $n^{\text{th}}$  root of unity. Then

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

Pf.

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

# Inverse FFT: implementation

Note. Need to divide result by  $n$ .

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

**IF** ( $n = 1$ ) **RETURN**  $y_0$ .

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

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

**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**  $(a_0, a_1, a_2, \dots, a_{n-1})$ .

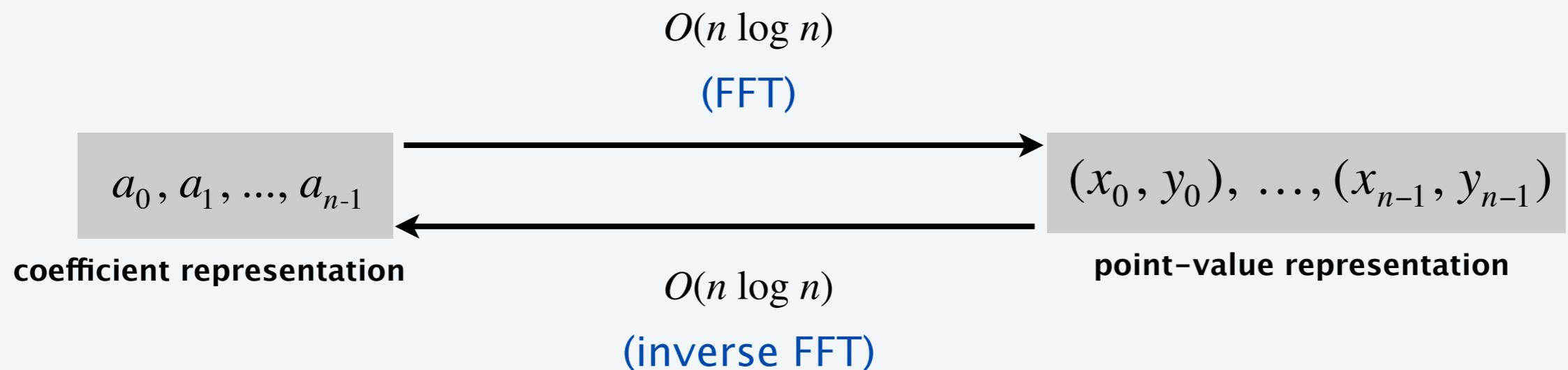
switch roles of  $a_i$  and  $y_i$

## Inverse FFT: summary

---

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

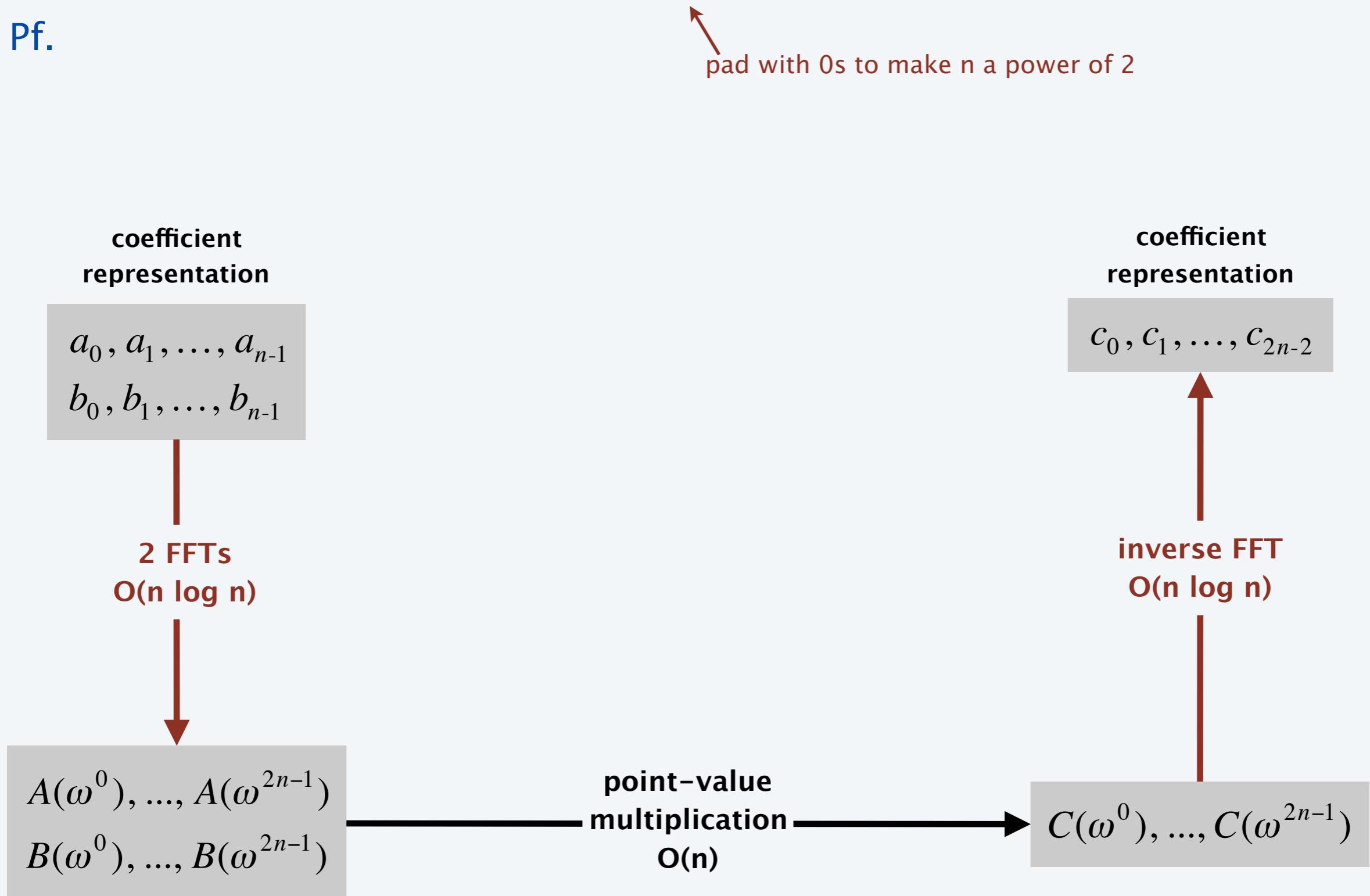
↑  
assumes  $n$  is a power of 2



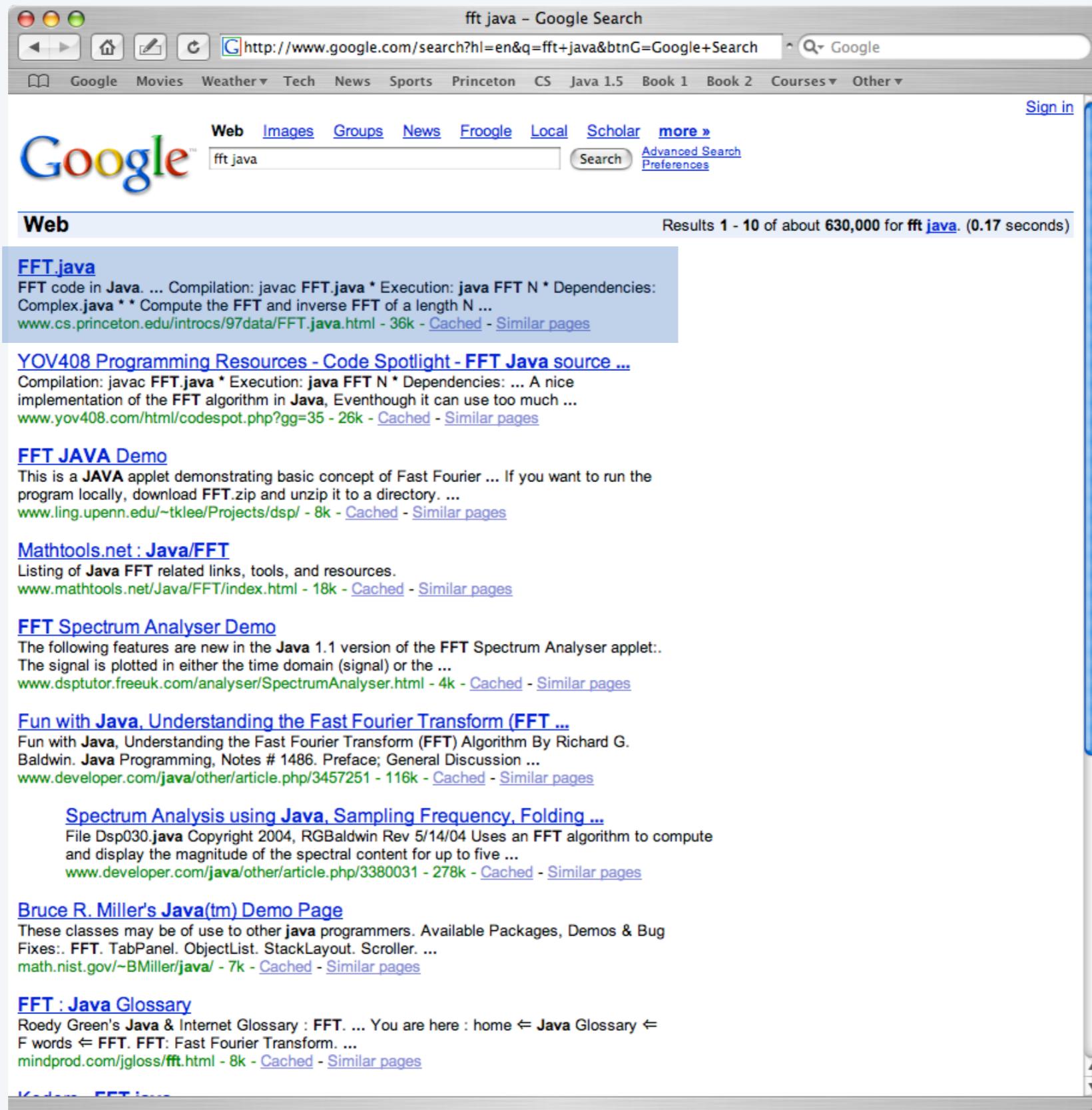
# Polynomial multiplication

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

Pf.



# FFT in practice ?



# FFT in practice

---

Fastest Fourier transform in the West. [Frigo–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  $a = a_{n-1} \dots a_1 a_0$  and  $b = b_{n-1} \dots b_1 b_0$ , 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.

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

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

**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  $n$ .