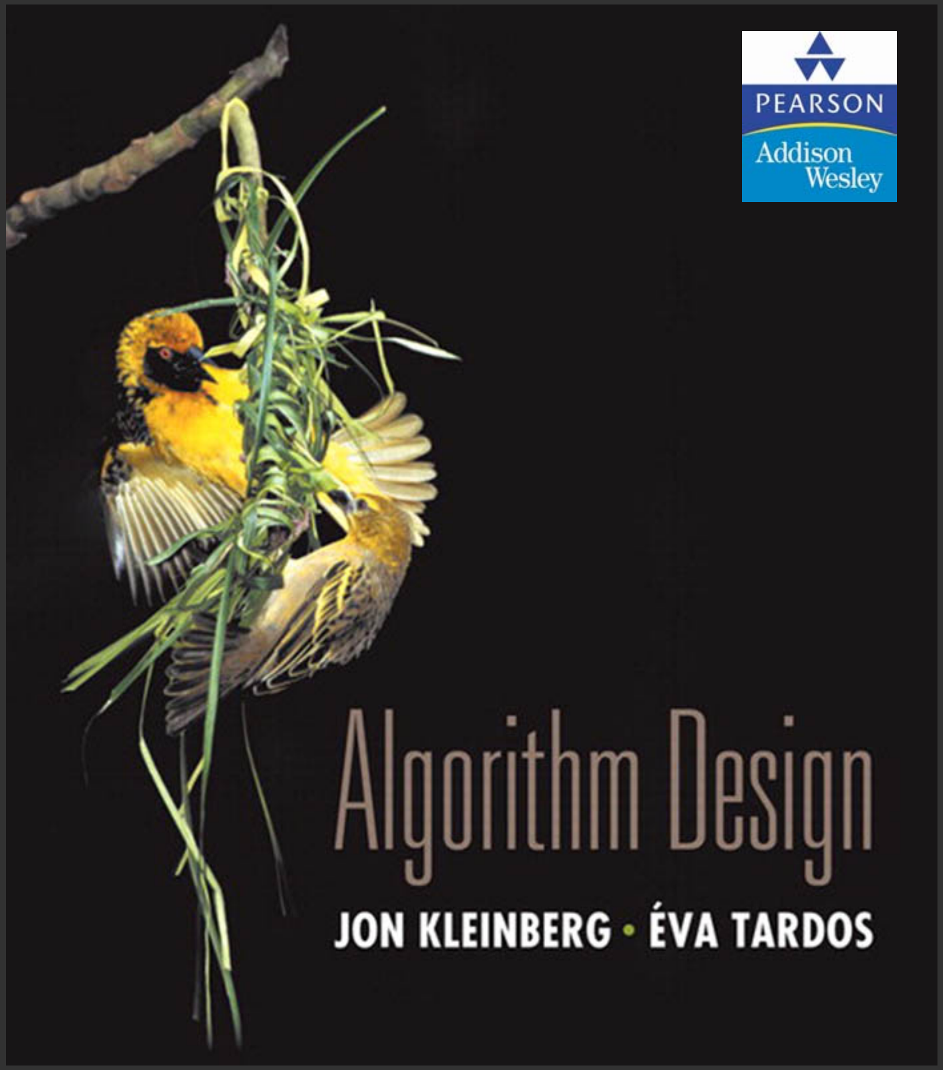


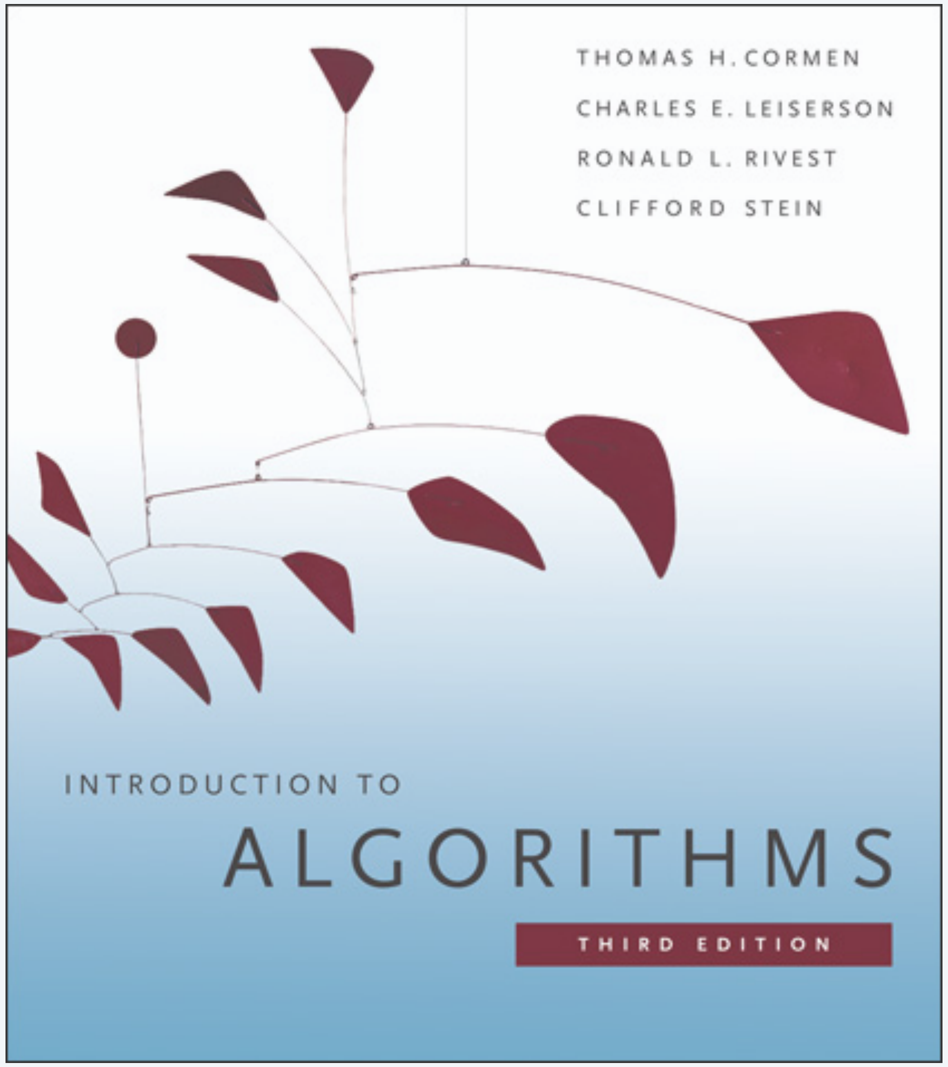
DIVIDE AND CONQUER II

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

Lecture slides by Kevin Wayne

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

Last updated on 5/3/18 11:00 PM



SECTIONS 4.4–4.6

DIVIDE AND CONQUER II

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

Divide-and-conquer recurrences

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

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

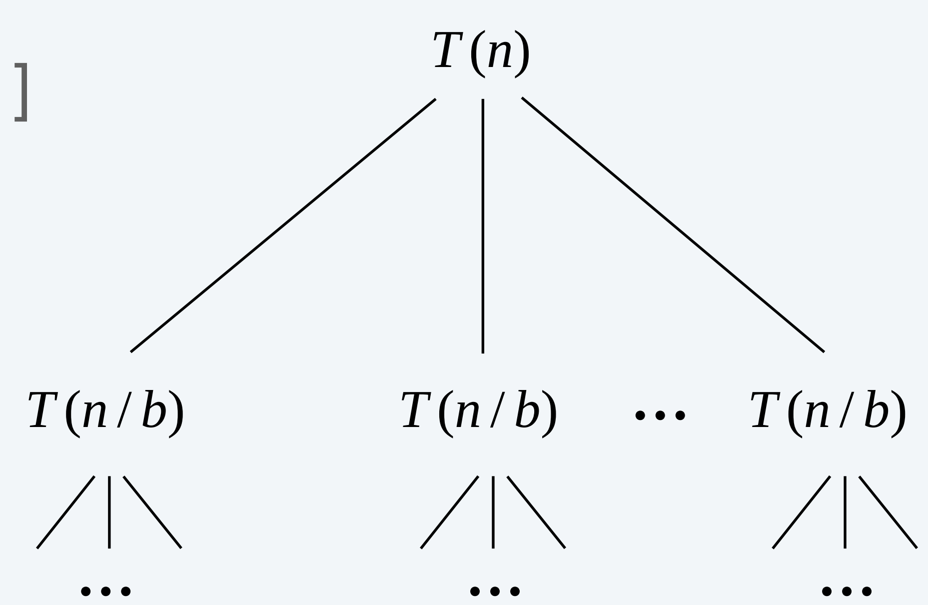
with $T(0) = 0$ and $T(1) = \Theta(1)$.

Terms.

- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.

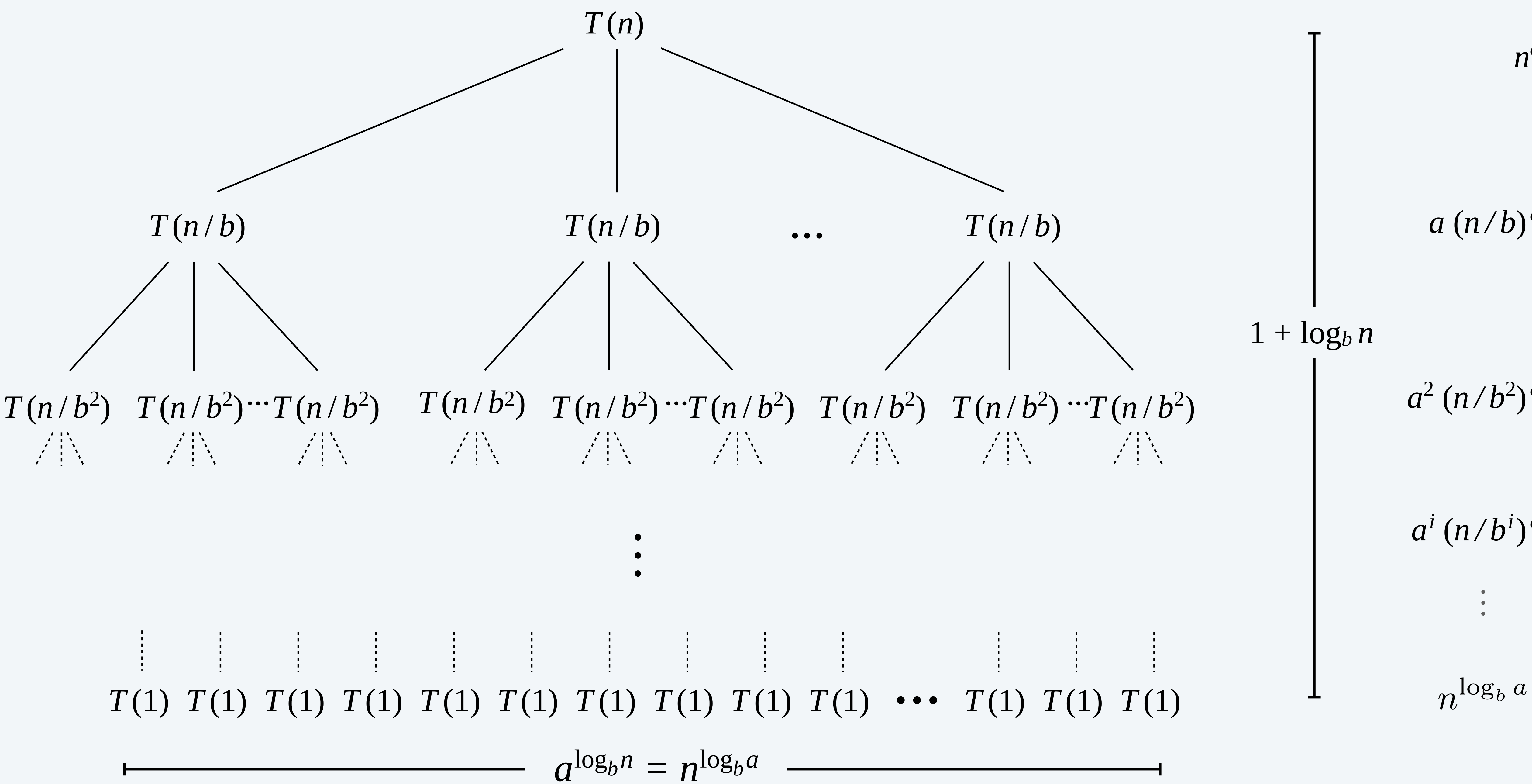
Recursion tree. [assuming n is a power of b]

- a = branching factor.
- a^i = number of subproblems at level i.
- $1 + \log_b n$ levels.
- n / b^i = size of subproblem at level i.



Divide-and-conquer recurrences: recursion tree

Suppose $T(n)$ satisfies $T(n) = a\,T(n/b) + n^c$ with $T(1) = 1$, for n a power of b.



$r = a / b^c$

$$T(n) \;=\; n^c \sum_{i=0}^{\log_b n} r^i$$

Divide-and-conquer recurrences: recursion tree analysis

Suppose $T(n)$ satisfies $T(n) = a\,T(n / b) + n^c$ with $T(1) = 1$, for n a power of b.

Let $r = a / b^c$. Note that $r < 1$ iff $c > \log_b a$.

$$T(n) \;=\; n^c \sum_{i=0}^{\log_b n} r^i \;=\; \begin{cases} \Theta(n^c) & \text{if } r < 1 \quad c > \log_b a \\ \Theta(n^c \log n) & \text{if } r = 1 \quad c = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } r > 1 \quad c < \log_b a \end{cases}$$

cost dominated by cost of root

cost evenly distributed in tree

cost dominated by cost of leaves

Geometric series.

- If $0 < r < 1$, then $1 + r + r^2 + r^3 + \ldots + r^k \le 1 / (1 - r)$.
- If $r = 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = k + 1$.
- If $r > 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = (r^{k+1} - 1) / (r - 1)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

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 $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Pf sketch.

- Prove when b is an integer and n is an exact power of b.
- Extend domain of recurrences to reals (or rationals).
- Deal with floors and ceilings. ← at most 2 extra levels in recursion tree

$$\begin{aligned} \lceil \lceil \lceil n/b \rceil /b \rceil /b \rceil \rceil \;&<\; n/b^3 + (1/b^2 + 1/b + 1) \\ &\leq\; n/b^3 + 2 \end{aligned}$$

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \ge 1$, $b \ge 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

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 $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Extensions.

- Can replace Θ with O everywhere.
- Can replace Θ with Ω everywhere.
- Can replace initial conditions with $T(n) = \Theta(1)$ for all $n \le n_0$ and require recurrence to hold only for all $n > n_0$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

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 $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Ex 1. $T(n) = 3\,T(\lfloor n / 2 \rfloor) + 5\,n$.

- $a = 3$, $b = 2$, $c = 1$, $\log_b a < 1.58$.
- $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.58})$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

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 $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.


ok to intermix floor and ceiling

Ex 2. $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 17\,n$.

- $a = 2$, $b = 2$, $c = 1$, $\log_b a = 1$.
- $T(n) = \Theta(n \log n)$.

Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c > 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

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 $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
Case 3. If $c > \log_b a$, then $T(n) = \Theta(n^c)$.

Ex 3. $T(n) = 48\, T(\lfloor n / 4 \rfloor) + n^3$.

- $a = 48$, $b = 4$, $c = 3$, $\log_b a \; > \; 2.79$.
- $T(n) = \Theta(n^3)$.

Master theorem need not apply

Gaps in master theorem.

- Number of subproblems is not a constant.

$$T(n) \; = n T(n/2) \; + \; n^2$$

- Number of subproblems is less than 1.

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

- Work to divide and combine subproblems is not $\Theta(n^c)$.

$$T(n) \; = \; 2\,T(n/2) \; + n \log n$$

Consider the following recurrence. Which case of the master theorem?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

A. Case 1: $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.

B. Case 2: $T(n) = \Theta(n \log n)$.

C. Case 3: $T(n) = \Theta(n)$.

D. Master theorem not applicable.

Consider the following recurrence. Which case of the master theorem?

$$T(n) \;=\; \begin{cases} 0 & \text{if } n \le 1 \\[2ex] T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

A. Case 1: $T(n) = \Theta(n)$.

B. Case 2: $T(n) = \Theta(n \log n)$.

C. Case 3: $T(n) = \Theta(n)$.

D. Master theorem not applicable.

Akra–Bazzi theorem

Theorem. [Akra–Bazzi 1998] Given constants $a_i > 0$ and $0 < b_i < 1$, functions $|h_i(n)| = O(n / \log^2 n)$ and $g(n) = O(n^c)$. If $T(n)$ satisfies the recurrence:

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

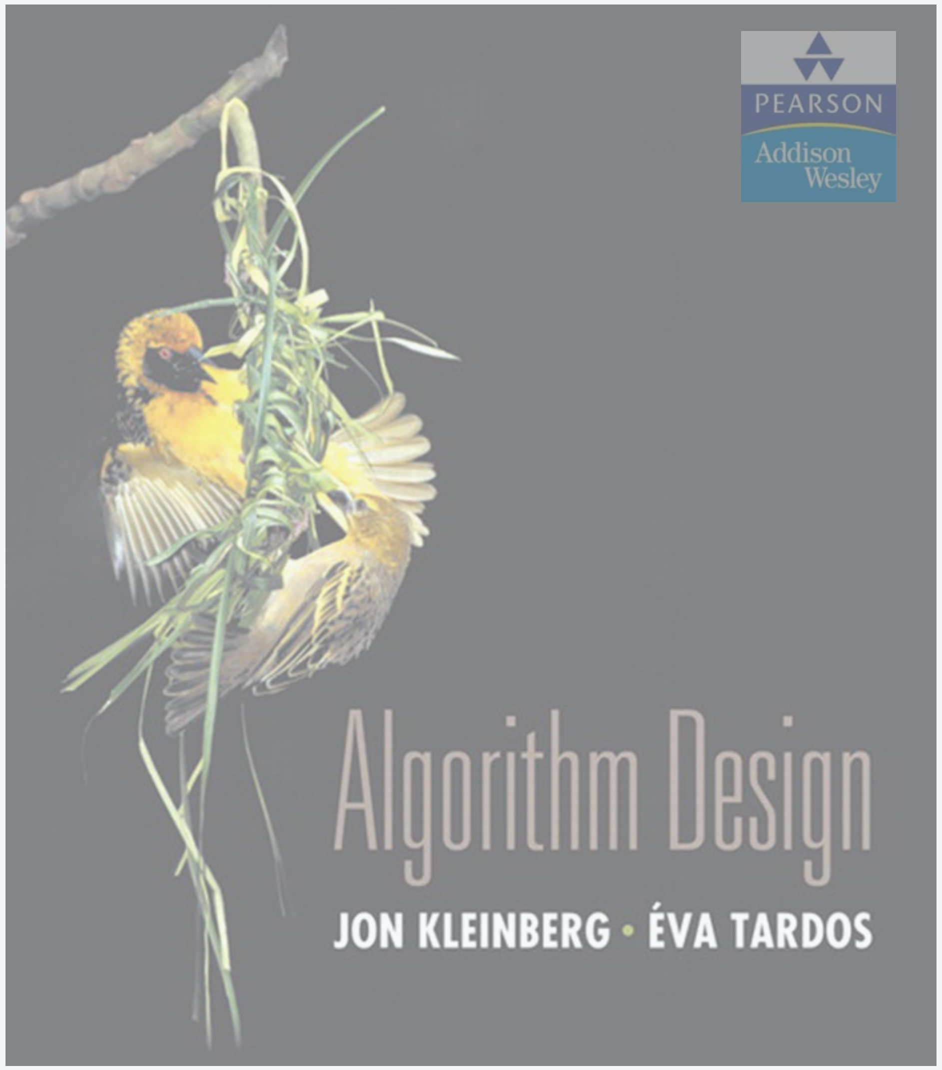
a_i subproblems of size $b_i\, n$

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. $T(n) = T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5\, n$, with $T(0) = 0$ and $T(1) = 0$.

- $a_1 = 1, \; b_1 = 1/5, \; a_2 = 1, \; b_2 = 7/10 \;\Rightarrow\; p = 0.83978\ldots < 1$.
- $h_1(n) = \lfloor n/5 \rfloor - n/5, \; h_2(n) = 3/10\, n - 3\lfloor n/10 \rfloor$.
- $g(n) = 11/5\, n \;\Rightarrow\; T(n) = \Theta(n)$.



SECTION 5.5

DIVIDE AND CONQUER II

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

Integer addition and subtraction

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. ← "bit complexity" (instead of word RAM)

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

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

Integer multiplication

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

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

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

Conjecture. [Kolmogorov 1956] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is false.

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 ½n-bit integers, recursively.
- Add and shift to obtain result.

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

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

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

use bit shifting to compute 4 terms

$$x\,y \;=\; (2^m\, a + b)\,(2^m\, c + d) \;=\; 2^{2m}\, \underbrace{ac}_{1} \;+\; 2^m\,(\underbrace{bc}_{2} + \underbrace{ad}_{3}) \;+\; \underbrace{bd}_{4}$$

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

Divide-and-conquer multiplication

$\text{MULTIPLY}(x, y, n)$

$\text{IF } (n = 1)$

$\quad \text{RETURN } x \times y.$

ELSE

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

$\quad a \leftarrow \lfloor x / 2^m \rfloor;\quad b \leftarrow x \bmod 2^m.$ ← $\Theta(n)$

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

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

$\quad f \leftarrow \text{MULTIPLY}(b, d, m).$ ← $4\,T(\lceil n / 2 \rceil)$

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

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

$\quad \text{RETURN } 2^{2m}\, e + 2^m\, (g + h) + f.$ ← $\Theta(n)$

Divide-and-conquer II: quiz 3

How many bit operations to multiply two n-bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

A. $T(n) = \Theta(n^{1/2})$.

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

C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.

D. $T(n) = \Theta(n^2)$.

Karatsuba trick

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
- To compute middle term $bc + ad$, use identity:

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

- Multiply only three ½n-bit integers, recursively.

$$x \;=\; \underbrace{1\;0\;0\;0}_{a}\;\underbrace{1\;1\;0\;1}_{b}$$

$$y \;=\; \underbrace{1\;1\;1\;0}_{c}\;\underbrace{0\;0\;0\;1}_{d}$$

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

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

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

middle term

$$x\,y \;=\; (2^m\, a \;+\; b)\;(2^m\, c \;+\; d) \;=\; 2^{2m}\, ac \;+\; 2^m\, \underbrace{(bc \;+\; ad\,)}_{\text{middle term}} \;+\; bd$$

$$=\; 2^{2m}\, ac \;+\; 2^m\, (ac \;+\; bd \;-\; (a - b)(c - d)) \;+\; bd$$

1 1 3 2 3

Karatsuba multiplication

KARATSUBA-MULTIPLY(x, y, n)

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

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

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

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

  $\Theta(n)$

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

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

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

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

  Flip sign of g if needed.

  RETURN $2^{2m}\, e + 2^m\, (e + f - g) + f$. $\Theta(n)$

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:

$$T(n) \;=\; \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) \;+\; \Theta(n) & \text{if } n > 1 \end{cases}$$

$$\Longrightarrow \quad T(n) \;=\; \Theta(n^{\log_2 3}) \;=\; O(n^{1.585})$$

Practice.

- Use base 32 or 64 (instead of base 2).
- Faster than grade-school algorithm for about 320–640 bits.

Integer arithmetic reductions

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

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

$$ab = \frac{(a+b)^2 - a^2 - b^2}{2}$$

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

History of asymptotic complexity of integer multiplication

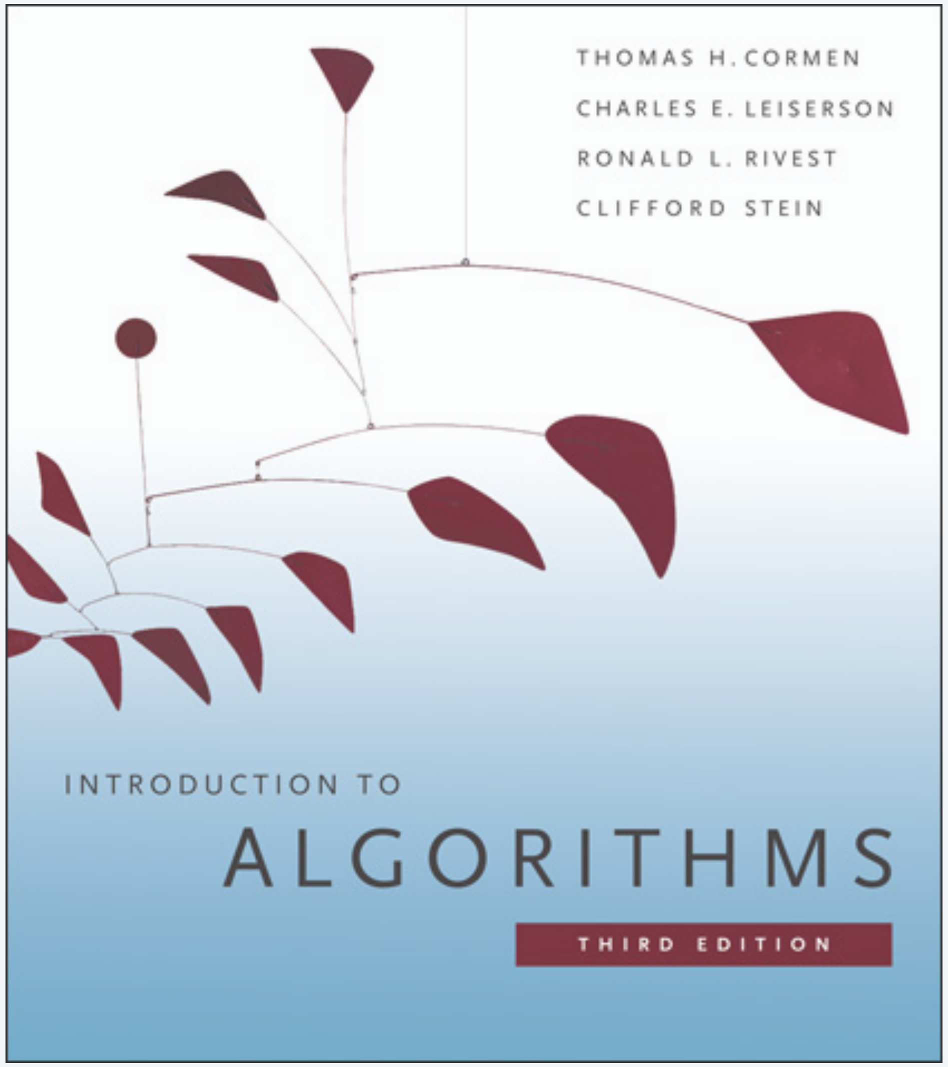
year	algorithm	bit operations
12xx	**grade school**	$O(n^2)$
1962	**Karatsuba–Ofman**	$O(n^{1.585})$
1963	**Toom-3, Toom-4**	$O(n^{1.465}),\ \ O(n^{1.404})$
1966	**Toom–Cook**	$O(n^{1+\varepsilon})$
1971	**Schönhage–Strassen**	$O(n \log n \cdot \log \log n)$
2007	**Fürer**	$n \log n\, 2^{O(\log^* n)}$
2018	**Harvey–van der Hoeven**	$O(n \log n \cdot 2^{2 \lg^* n})$
	???	$O(n)$

number of bit operations to multiply two n-bit integers

Remark. GNU Multiple Precision library uses one of first five algorithms depending on n.

GMP
«Arithmetic without limitations»

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



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 $\underbrace{C = AB}_{c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}}$.

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

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

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

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

Block matrix multiplication

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

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

Block matrix multiplication: warmup

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

- Divide: partition A and B into ½n-by-½n blocks.
- Conquer: multiply 8 pairs of ½n-by-½n matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

n-by-n matrices

$$\underbrace{C}_{n\text{-by-}n} = \underbrace{A}_{n\text{-by-}n} \times B$$

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

½n-by-½n matrices

8 matrix multiplications (of ½n-by-½n matrices)

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

4 matrix additions (of ½n-by-½n matrices)

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).

scalars

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 scalar multiplications

Pf.
$$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \ \checkmark \end{aligned}$$

Strassen's trick

Key idea. Can multiply two $\overset{n\text{-by-}n}{\cancel{2\text{-by-}2}}$ matrices via 7 $\overset{\frac{1}{2}n\text{-by-}\frac{1}{2}n \text{ matrix}}{\cancel{\text{scalar}}}$ multiplications (plus 11 additions and 7 subtractions).

½n-by-½n matrices

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

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

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

Pf. $$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad ✔ \end{aligned}$$

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 ½n-by-½n blocks.

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

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

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

$7\,T(n / 2) + \Theta(n^2)$

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

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

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

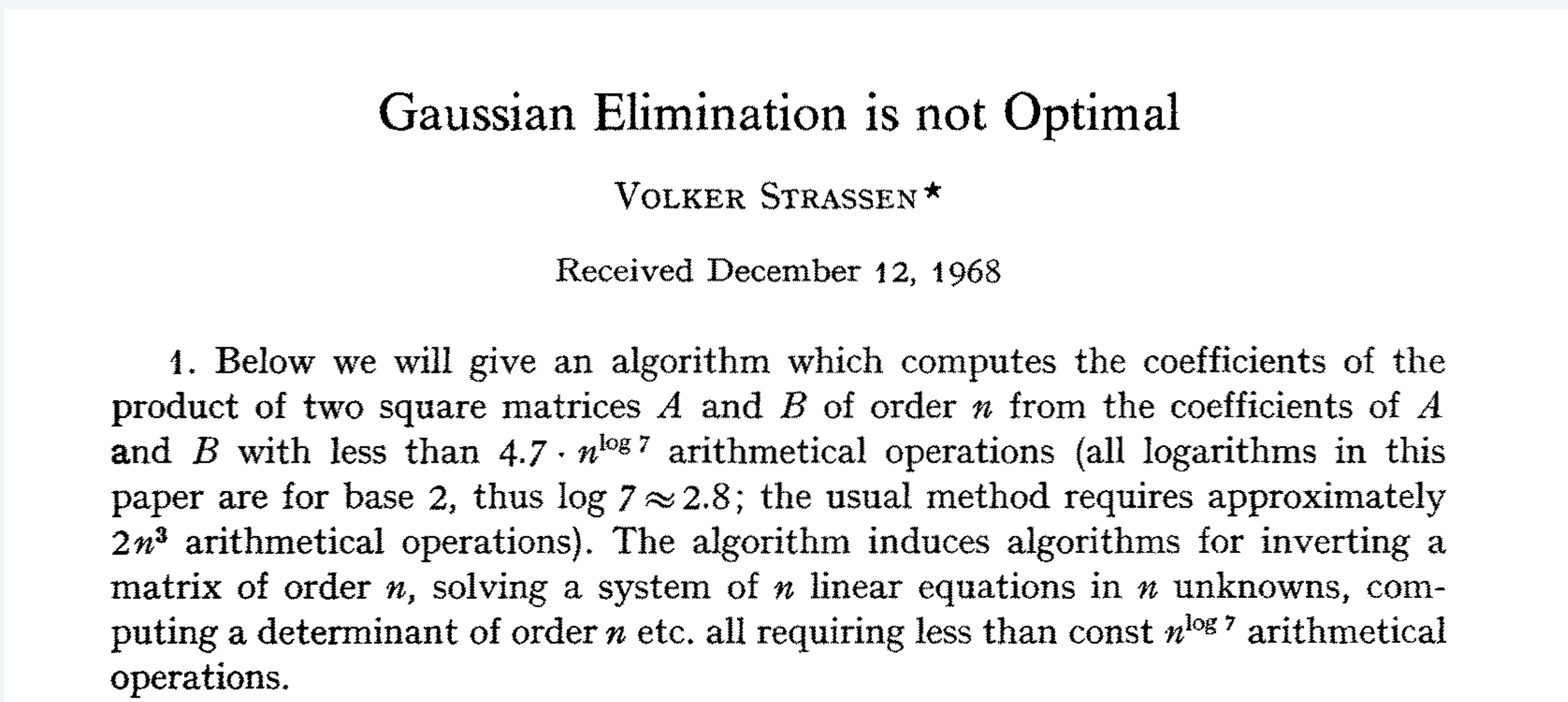
$\Theta(n^2)$

RETURN C.

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

Analysis of Strassen's algorithm

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



Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

Analysis of Strassen's algorithm

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

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:

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

- When n is not a power of 2, pad matrices with zeros to be n'-by-n', where $n \le n' < 2n$ and n' is a power of 2.

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

Strassen's algorithm: practice

Implementation issues.

- Sparsity.
- Caching.
- n not a power of 2.
- Numerical stability.
- Non-square matrices.
- Storage for intermediate submatrices.
- Crossover to classical algorithm when n is "small."
- Parallelism for multi-core and many-core architectures.

Common misperception. *"Strassen's algorithm is only a theoretical curiosity."*

- Apple reports 8x speedup when $n \approx 2{,}048$.
- Range of instances where it's useful is a subject of controversy.

Strassen's Algorithm Reloaded

Jianyu Huang*, Tyler M. Smith*†, Greg M. Henry‡, Robert A. van de Geijn*†
*Department of Computer Science and †Institute for Computational Engineering and Sciences,
The University of Texas at Austin, Austin, TX 78712
Email: jianyu,tms,rvdg@cs.utexas.edu
‡Intel Corporation, Hillsboro, OR 97124
Email: greg.henry@intel.com

Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n-by-n matrices?

A. $\Theta(n^{\log_3 21})$

B. $\Theta(n^{\log_2 21})$

C. $\Theta(n^{\log_9 21})$

D. $\Theta(n^2)$

Is it possible to multiply two 3-by-3 matrices using only 21 scalar multiplications?

A. Yes.

B. No.

C. Unknown.

Fast matrix multiplication: theory

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

A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

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

A. Impossible. [Hopcroft–Kerr, Winograd 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Begun, the decimal wars have. [Pan 1978, Bini et al., Schönhage, …]

- Two 70-by-70 matrices with 143,640 scalar multiplications. $O(n^{2.7962})$
- 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 arithmetic complexity of matrix multiplication

year	algorithm	arithmetic operations
1858	**"grade school"**	$O(n^3)$
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**	$O(n^{2.3755})$
2010	**Strother**	$O(n^{2.3737})$
2011	**Williams**	$O(n^{2.372873})$
2014	**Le Gall**	$O(n^{2.372864})$
	???	$O(n^{2+\varepsilon})$

galactic algorithms

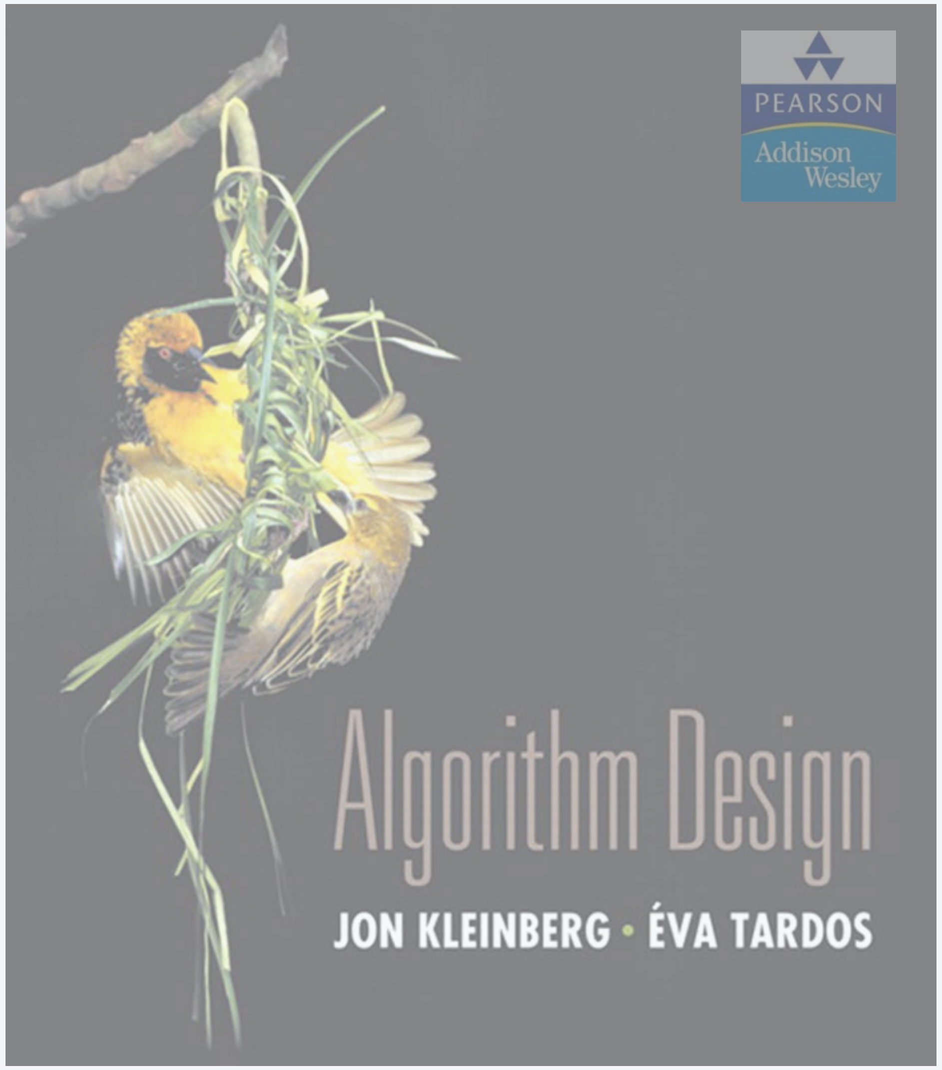
number of arithmetic operations to multiply two n-by-n matrices

Numeric linear algebra reductions

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

linear algebra problem	expression	arithmetic complexity
matrix multiplication	$A \times B$	$MM(n)$
matrix squaring	A^2	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$\lvert A \rvert$	$\Theta(MM(n))$
rank	$rank(A)$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L\,U$	$\Theta(MM(n))$
least squares	$\min \lVert Ax - b \rVert_2$	$\Theta(MM(n))$

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



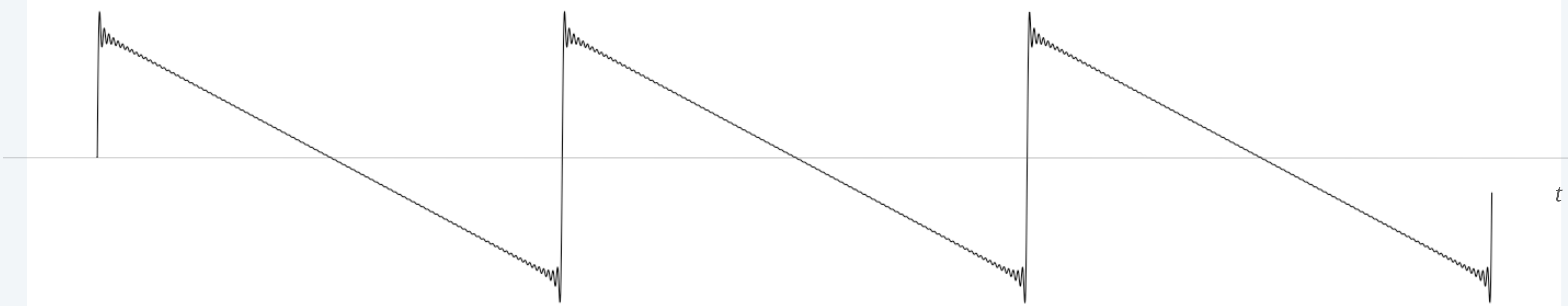
SECTION 5.6

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

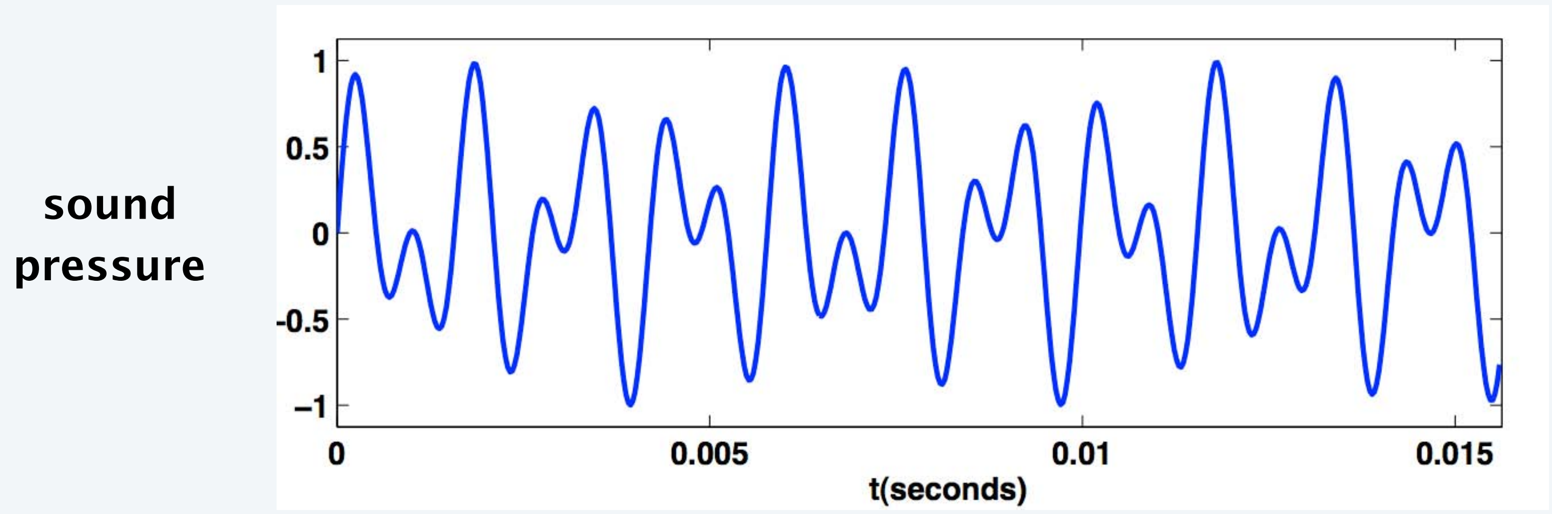
Euler's identity. $e^{ix} = \cos x + i \sin x$.

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

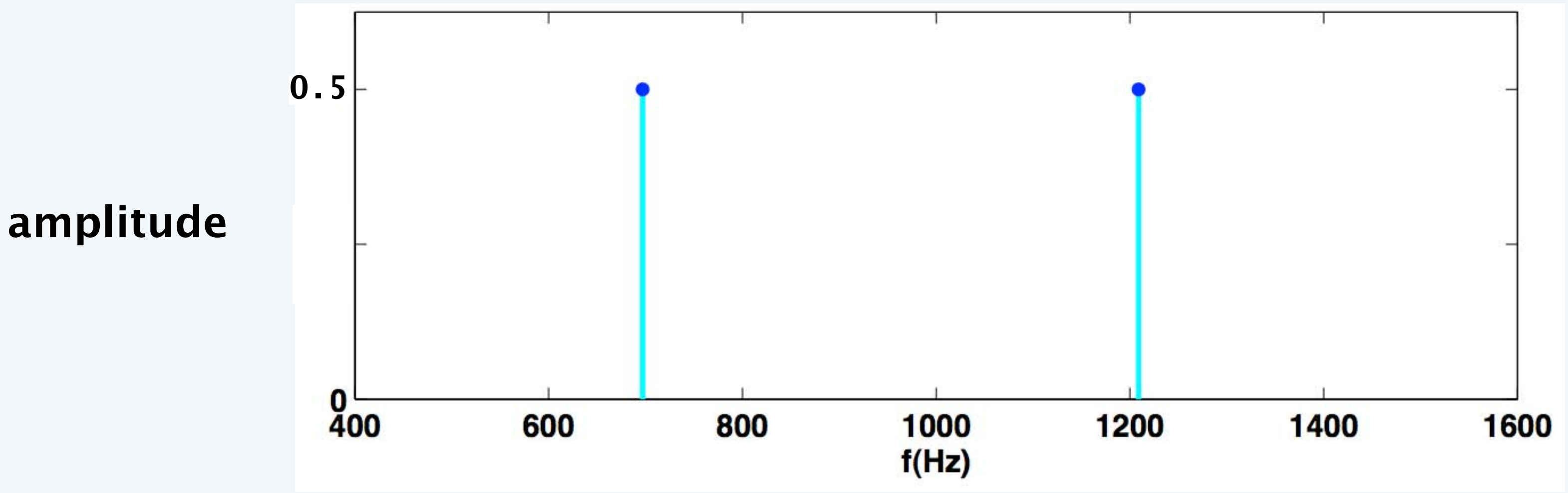
Time domain vs. frequency domain

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

Time domain.



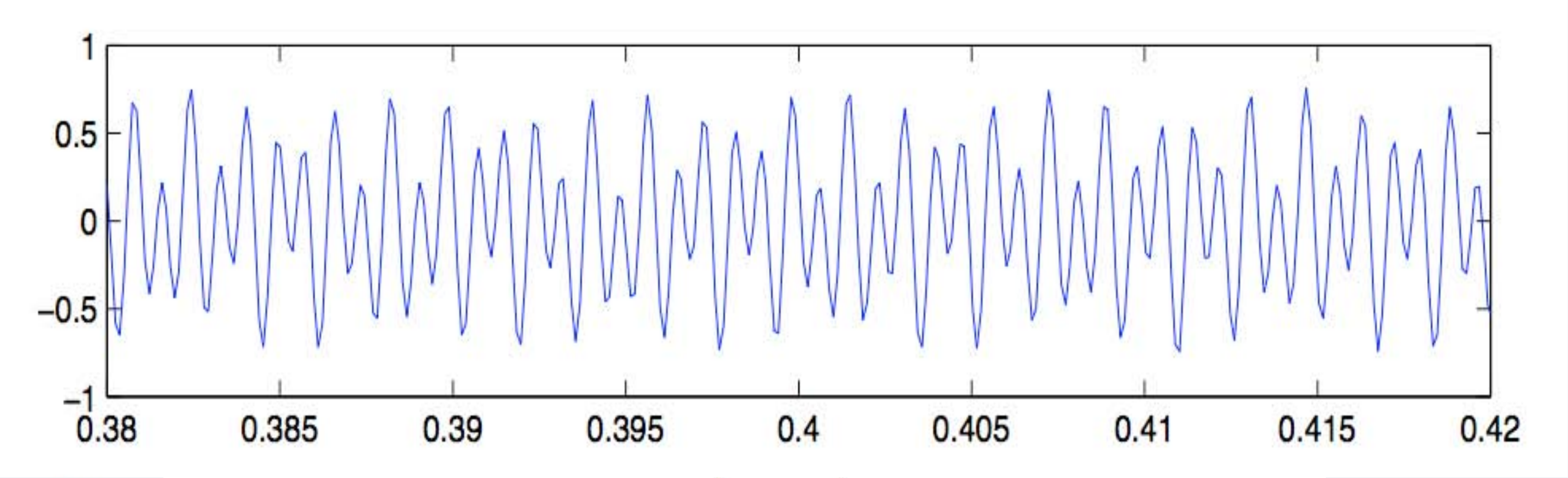
Frequency domain.



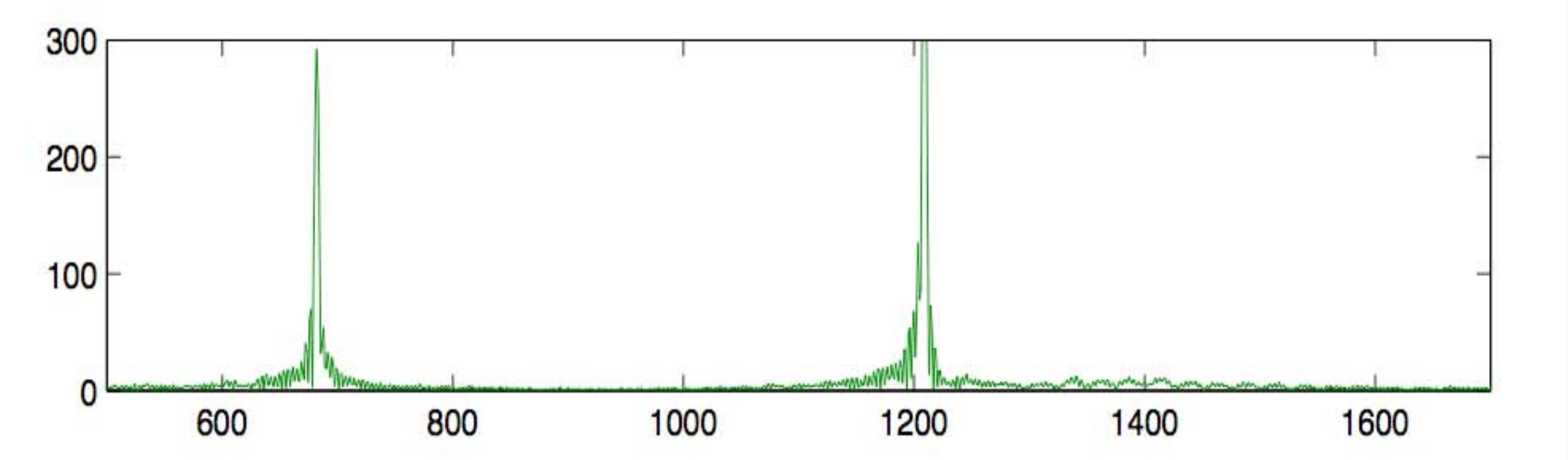
Reference: Cleve Moler, Numerical Computing with MATLAB

Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

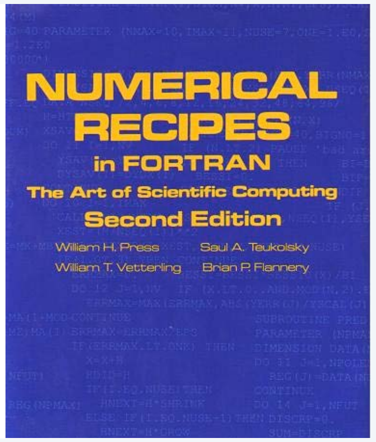
Fast Fourier transform

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*

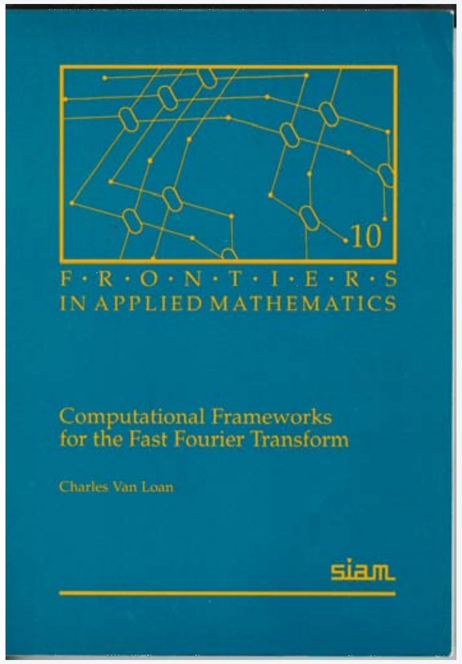


Fast Fourier transform: applications

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*



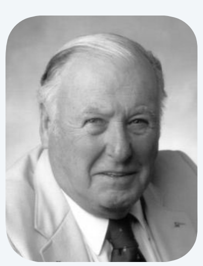
Fast Fourier transform: brief history

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). Detect nuclear tests in Soviet Union and track 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.

Importance not fully realized until emergence of digital computers.

Polynomials: coefficient representation

Univariate polynomial. [coefficient representation]

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

Addition. $O(n)$ arithmetic operations.

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

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

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

```
double val = 0.0;
for (int j = n-1; j >= 0; j--)
   val = a[j] + (x * val);
```

Multiplication (linear convolution). $O(n^2)$ using brute force.

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

What was the subject of Gauss' Ph.D thesis?

A. Gaussian elimination.

B. Fast Fourier transform.

C. Prime number theorem.

D. Cauchy integral theorem.

E. Fundamental theorem of algebra.

F. Angle-preserving maps.

G. Method of least squares.

H. Non-Euclidean geometry.

I. Constructing a regular heptadecagon with straightedge and compass.

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.

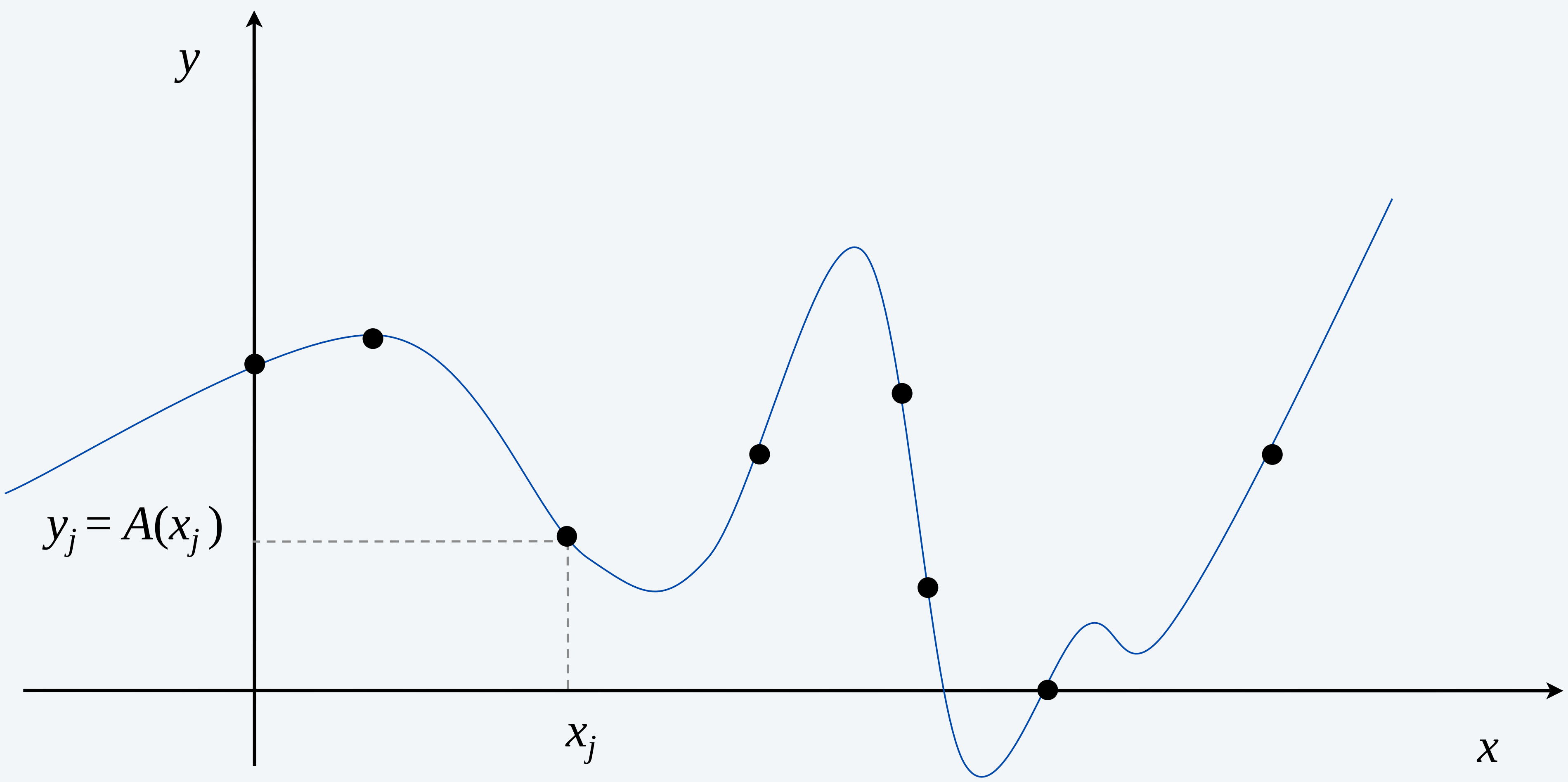
Quaelibet aequatio algebraica determinata reduci potest ad formam $x^m+Ax^{m-1}+Bx^{m-2}+$ etc. $+M=0$, ita vt m sit numerus integer positiuus. Si partem primam huius aequationis per X denotamus, aequationique $X=0$ per plures valores inaequales ipsius x satisfieri supponimus, puta ponendo $x=\alpha$, $x=\beta$, $x=\gamma$ 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 univariate polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree $n-1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



Polynomials: point-value representation

Univariate polynomial. [point-value representation]

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

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

Addition. $O(n)$ arithmetic operations.

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

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

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

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

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

not used

Converting between two representations

Tradeoff. Either 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

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

point-value representation

Converting between two representations

Application. Polynomial multiplication (coefficient representation).

coefficient representation

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

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

FFT

$O(n \log n)$

point value representation

$(x_0, y_0),\ \ldots,\ (x_{2n-1}, y_{2n-1})$

$(x_0, z_0),\ \ldots,\ (x_{2n-1}, z_{2n-1})$

point-value multiplication

$O(n)$

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

point value representation

inverse FFT

$O(n \log n)$

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

coefficient representation

Converting between two representations: brute force

Coefficient $\Rightarrow$ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 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. $O(n^2)$ via 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 $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 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. $O(n^3)$ via Gaussian elimination.

or $O(n^{2.38})$ via fast matrix multiplication

Which divide-and-conquer approach to use to multiply polynomials?

$$A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$$

A. Divide polynomial into low- and high-degree terms.

$$A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$
$$A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$$

B. Divide polynomial into even- and odd-degree terms.

$$A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$
$$A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$

C. Either A or B.

D. Neither A nor B.

Divide-and-conquer

Decimation in time. Divide into even- and odd- degree terms.

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

Cooley–Tukey radix 2 FFT

Decimation in frequency. Divide into low- and high-degree terms.

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

Sande–Tukey radix 2 FFT

Coefficient to point-value representation: intuition

Coefficient $\Rightarrow$ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 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-degree terms.

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

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–1 at 2 points by evaluating two polynomials of degree ½n – 1 at only 1 point.**

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, 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-degree terms.

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

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

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

⟵ **Can evaluate polynomial of degree n−1 at 4 points by evaluating two polynomials of degree ½n − 1 at only 2 points.**

Discrete Fourier transform

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

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

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

DFT

Fourier matrix F_n

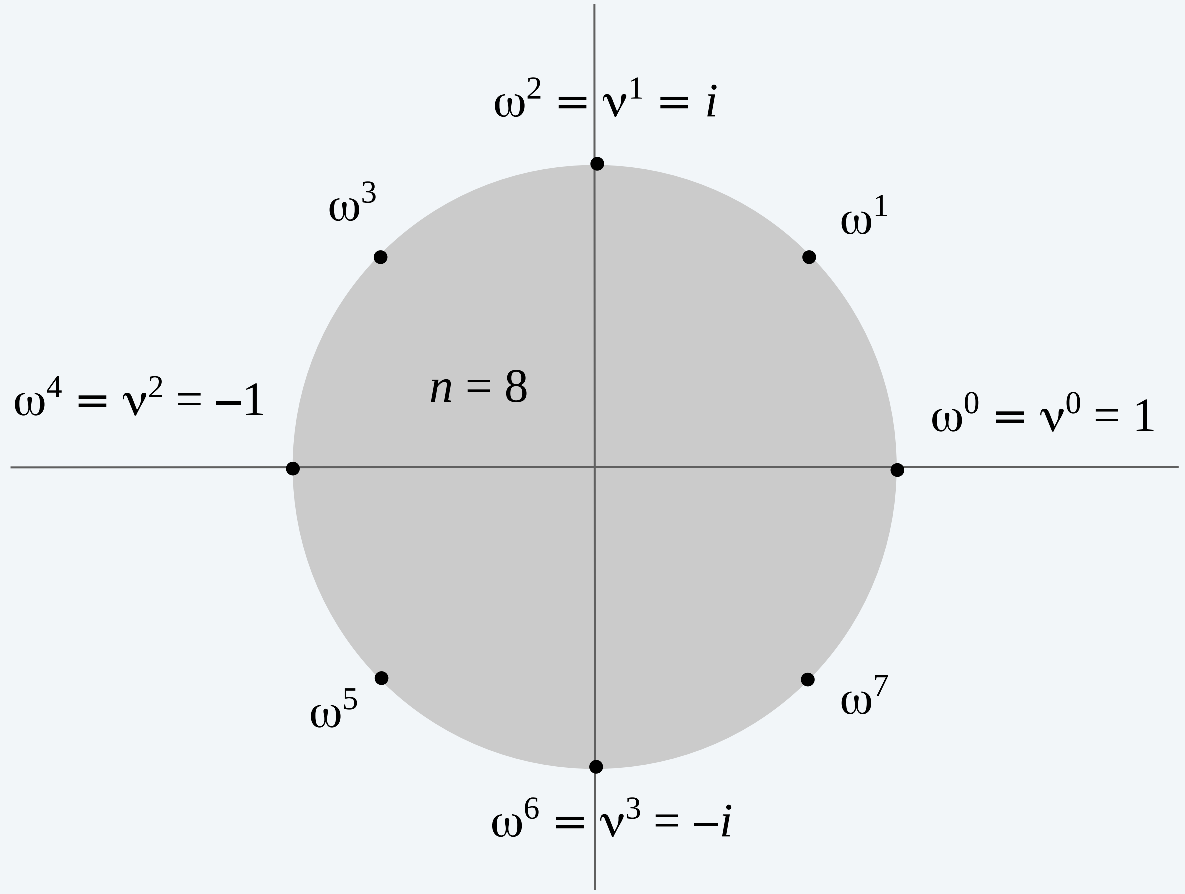
Roots of unity

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-degree terms.

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

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.

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

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

$A(-\omega^k)$

FFT: implementation

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: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

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

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($n / 2, a_1, a_3, a_5, \ldots, a_{n-1}$).

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

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

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

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

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

$\leftarrow \Theta(n)$

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 $O(n \log n)$ arithmetic operations and $O(n)$ extra space.

assumes n is a power of 2

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

coefficient representation — $a_0, a_1, \ldots, a_{n-1}$

$O(n \log n)$

FFT →

← **???**

point-value representation — $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$

When computing the FFT of (a_0, a_1, a_2, …, a_7), which are the first two coefficients involved in an arithmetic operation?

A. a_0 and a_1.

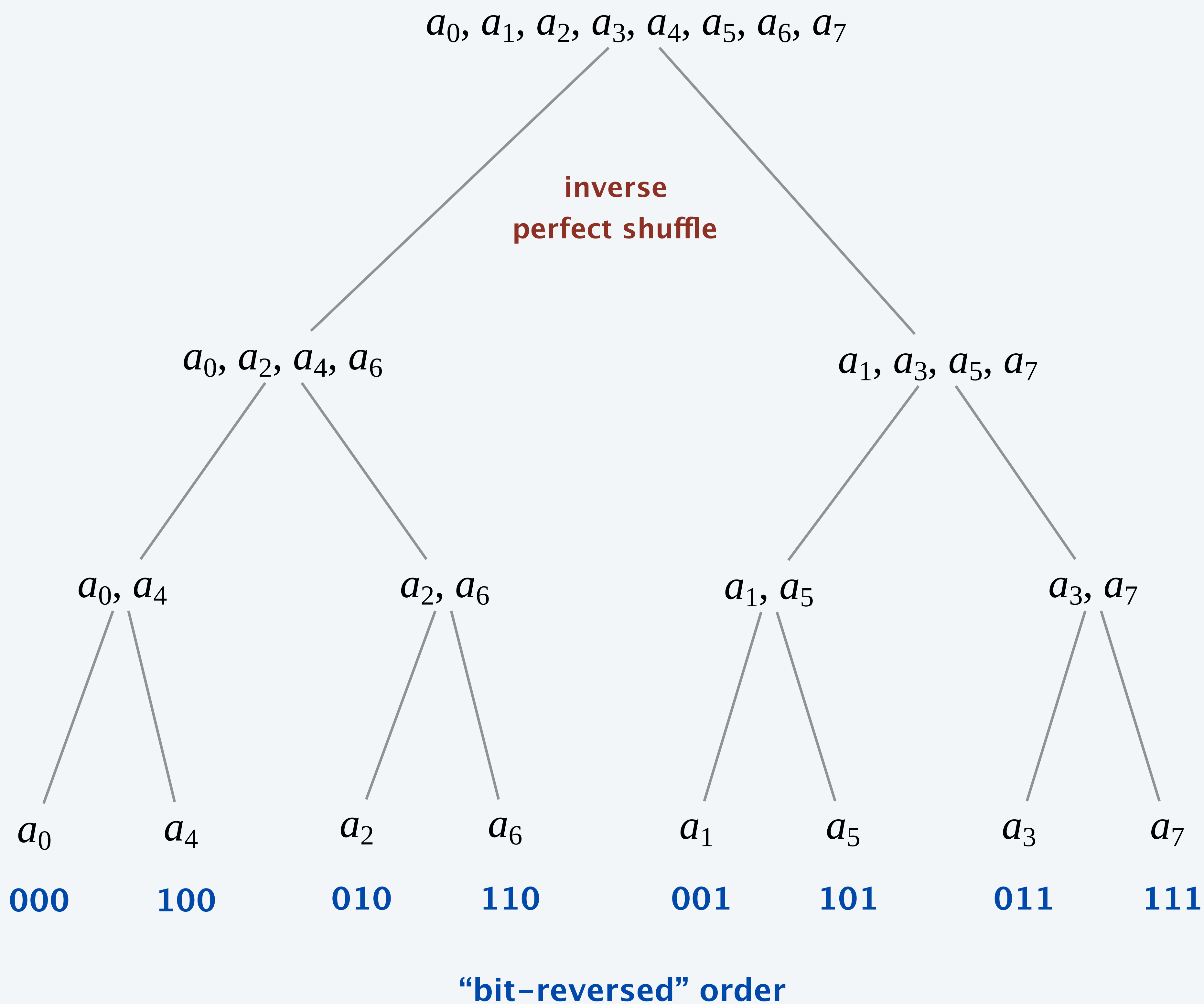
B. a_0 and a_2.

C. a_0 and a_4.

D. a_0 and a_7.

E. None of the above.

FFT: recursion tree



FFT: Fourier matrix decomposition

Alternative viewpoint. FFT is a recursive decomposition of Fourier matrix.

$$F_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Fourier matrix

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \qquad D_n = \begin{bmatrix} \omega^0 & 0 & 0 & \dots & 0 \\ 0 & \omega^1 & 0 & \dots & 0 \\ 0 & 0 & \omega^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \omega^{n-1} \end{bmatrix}$$

DFT

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

Inverse discrete Fourier transform

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

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

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 the inverse FFT, apply the 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.

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

summation lemma (below)

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

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

Pf.

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

Inverse FFT: implementation

Note. Need to divide result by n.

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

$\textsc{If}\ (n = 1)\ \textsc{Return}\ y_0.$

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

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

$\textit{FOR}\ k = 0\ \textsc{to}\ n / 2 - 1.$

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

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

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

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

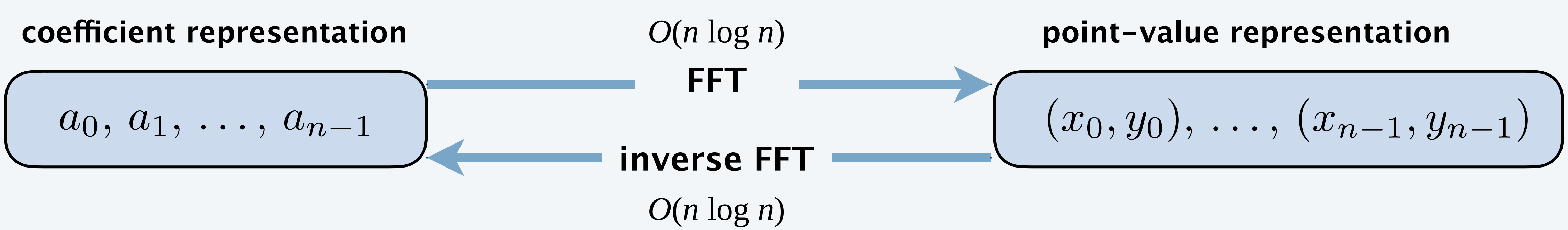
switch roles of a_i and y_i

Inverse FFT: summary

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

assumes n is a power of 2

Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.



Polynomial multiplication

Theorem. Given two polynomials $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ and $B(x) = b_0 + b_1 x + \ldots + b_{n-1} b^{n-1}$ of degree $n-1$, can multiply them in $O(n \log n)$ arithmetic operations.

pad with 0s to make n a power of 2

Pf.

coefficient representation

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

two FFTs
$O(n \log n)$

point-value representation

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

point-value multiplication
$O(n)$

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

point-value representation

inverse FFT
$O(n \log n)$

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

coefficient representation

FFT in practice ?

Google

fft java

All Videos Maps Images News More Settings Tools

About 422,000 results (0.27 seconds)

FFT.java
https://introcs.cs.princeton.edu/97data/FFT.java.html
Nov 29, 2017 - 0) { throw new IllegalArgumentException("n is not a power of 2"); } // **fft** of even terms Complex[] even = new Complex[n/2]; for (int k = 0; k < n/2; k++) { even[k] = x [2*k]; } Complex[] q = **fft**(even); // **fft** of odd terms Complex[] odd = even; // reuse the array for (int k = 0; k < n/2; k++) { odd[k] = x[2*k + 1]; } Complex[] r ...

FFT.java - Algorithms, 4th Edition
https://algs4.cs.princeton.edu/99scientific/FFT.java.html
Oct 20, 2017 - @param x the complex array * @return the **FFT** of the complex array {@code x} * @throws IllegalArgumentException if the length of {@code x} is not a power of 2 */ public static Complex[] **fft**(Complex[] x) { int n = x.length; // base case if (n == 1) { return new Complex[] { x[0] }; } // radix 2 Cooley-Tukey **FFT** if (n ...

Reliable and fast FFT in Java - Stack Overflow
https://stackoverflow.com/questions/3287518/reliable-and-fast-fft-in-java
Nov 7, 2011 - FFTW is the 'fastest fourier transform in the west', and has some **Java** wrappers: http://www.fftw.org/download.html. Hope that helps!

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 an in-place, nonrecursive version of Cooley–Tukey.
- Instead of executing a fixed algorithm, it evaluates the 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.
- Parallelism.

FFTW

http://www.fftw.org

Top 10 algorithms of the 20th century

FROM THE EDITORS

THE JOY OF ALGORITHMS

Francis Sullivan, Associate Editor-in-Chief

THE THEME OF THIS FIRST-OF-THE-CENTURY ISSUE OF *COMPUTING IN SCIENCE & ENGINEERING* IS ALGORITHMS. IN FACT, WE WERE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—TO CALL THE 10 EXAMPLES WE'VE SELECTED "THE TOP 10 ALGORITHMS OF THE CENTURY."

Daniel Rockmore describes the FFT as an algorithm "the whole family can use." The FFT is perhaps the most ubiquitous algorithm in use today to analyze and manipulate digital or discrete data. The FFT takes the operation count for discrete Fourier transform from $O(N^2)$ to $O(N \log N)$.

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: $O(n \log n)$ floating-point operations.

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

Theory. [Schönhage–Strassen 1971]

- $O(n \log^2 n)$ bit operations. ← FFT over complex numbers; need $O(\log n)$ bits of precision
- $O(n \log n \cdot \log \log n)$ bit operations. ← FFT over ring of integers (modulo a Fermat number)

Practice. [GNU Multiple Precision Arithmetic Library]
Switches to FFT-based algorithm when n is large (≥ 5–10K).

3-Sum (revisited)

3-Sum. Given three sets X, Y, and Z of n integers each, determine whether there is a triple $i \in X$, $j \in Y$, $k \in Z$ such that $i + j = k$.

Assumption. All integers are between 0 and m.

Goal. $O(m \log m + n \log n)$ time.

$m = 19, \; n = 3$

$X = \{ 4, 7, 10 \}$

$Y = \{ 5, 8, 15 \}$

$Z = \{ 4, 13, 19 \}$

a yes instance
(4 + 15 = 19)