CSCI 355: Algorithm Design and Analysis
6. Divide and Conquer II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*

Divide-and-conquer recurrences

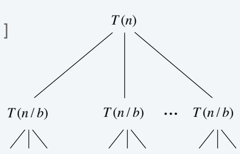
Goal. A recipe for solving common divide-and-conquer recurrences:

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

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

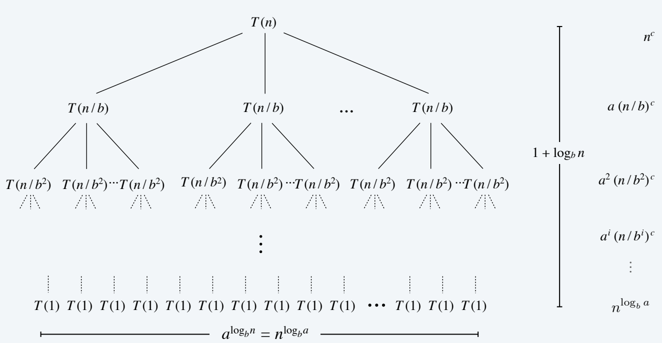
Terms.
- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.

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.



$r = a / b^c \qquad T(n) = n^c \sum_{i=0}^{\log_b n} r^i$

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.

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

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

← 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 $1 + r + r^2 + r^3 + \ldots + r^k = (r^{k+1} - 1)\,/\,(r - 1)$.

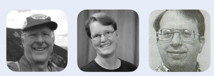
Divide-and-conquer recurrences: master theorem

Master theorem. Let $a \ge 1$, $b \ge 2$, 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 $T(n) = \Theta(n^c)$.



Proof 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

$$\lceil \lceil \lceil n/b \rceil / b \rceil / b \rceil \;<\; n/b^3 + (1/b^2 + 1/b + 1)$$
$$\le\; n/b^3 + 2$$

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 the 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 $T(n) = \Theta(n^c)$.


Bentley Haken Saxe

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

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

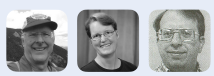
Divide-and-conquer recurrences: master theorem

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 $T(n) = \Theta(n^c)$.


Bentley Haken Saxe

okay to intermix floor and ceiling

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

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

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 $T(n) = \Theta(n^c)$.


Bentley Haken Saxe

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

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

Divide-and-conquer recurrences: master theorem

Gaps in the master theorem (inadmissible equations).

- Number of subproblems is not a constant.

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

- Number of subproblems is less than 1.

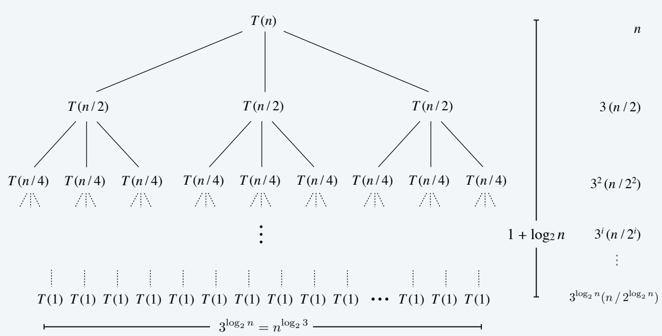
$$T(n) \;=\; \frac{1}{2}\,T(n/2) \;+\; n^2$$

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

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

Recurrence tree: cost dominated by cost of leaves

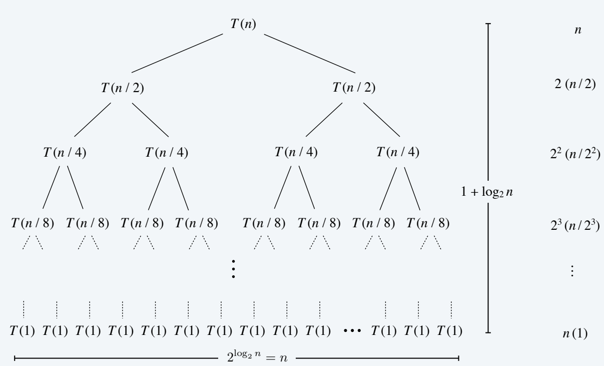
Ex 1. If $T(n)$ satisfies $T(n) = 3\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n^{\log_2 3})$.



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

Recurrence tree: cost evenly distributed among levels

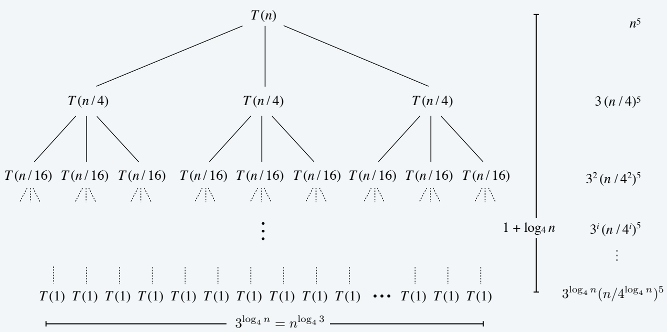
Ex 2. If $T(n)$ satisfies $T(n) = 2\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log n)$.



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

Recurrence tree: 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 $T(n) = \Theta(n^5)$.



$r = 3/4^5 < 1 \qquad T(n) \le (1 + r + r^2 + r^3 + \dots)\, n^5 \le \dfrac{1}{1-r}\, n^5$

CSCI 355: Algorithm Design and Analysis
6. Divide and Conquer II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*

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
```


Kolmogorov


Karatsuba

Conjecture. [Kolmogorov 1956] Grade-school algorithm is optimal.
Theorem. [Karatsuba 1960] Conjecture is false.

Integer multiplication: divide-and-conquer

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

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

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

Integer multiplication: divide-and-conquer

MULTIPLY(x, y, n)

IF ($n = 1$)
  RETURN $x\,y$.
ELSE
  $m \leftarrow \lceil n / 2 \rceil$.
  $a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.
  $c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.   ← $\Theta(n)$
  $e \leftarrow$ MULTIPLY(a, c, m).
  $f \leftarrow$ MULTIPLY(b, d, m).
  $g \leftarrow$ MULTIPLY(b, c, m).
  $h \leftarrow$ MULTIPLY(a, d, m).   ← $4\,T(\lceil n / 2 \rceil)$
  RETURN $2^{2m}\,e + 2^m\,(g + h) + f$.   ← $\Theta(n)$

Karatsuba's trick

To multiply two n-bit integers x and y:

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

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

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

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

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

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

$$x\,y \;=\; (2^m\, a + b)\,(2^m\, c + d) \;=\; 2^{2m}\, \underset{1}{ac} + 2^m\, \overbrace{(\underset{2}{bc} + \underset{3}{ad})}^{\text{middle term}} \;+\; \underset{4}{bd}$$

$$=\; 2^{2m}\, \underset{1}{ac} + 2^m\, (\underset{1}{ac} + \underset{4}{bd} \;-\; \underset{*}{(a - b)(c - d)}) + \underset{4}{bd}$$

Integer multiplication: Karatsuba's algorithm

```
KARATSUBA-MULTIPLY(x, y, n)

IF (n = 1)
   RETURN x y.
ELSE
   m ← ⌈ n / 2 ⌉.                                   ┐
   a ← ⌊ x / 2^m ⌋;  b ← x mod 2^m.                  ├ ← Θ(n)
   c ← ⌊ y / 2^m ⌋;  d ← y mod 2^m.                  ┘
   e ← KARATSUBA-MULTIPLY(a, c, m).                  ┐
   f ← KARATSUBA-MULTIPLY(b, d, m).                  ├ ← 3 T(⌈n / 2⌉)
   g ← KARATSUBA-MULTIPLY(|a – b|, |c – d|, m).      ┘
   Flip sign of g if needed.
   RETURN 2^(2m) e + 2^m (e + f – g) + f.          ← Θ(n)
```


Karatsuba's algorithm: 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:

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

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

In practice.

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

Integer arithmetic reductions

arithmetic problem	formula	bit complexity
integer multiplication	$a \times b$	$M(n)$
integer squaring	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

Integer multiplication in loglinear time

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

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

Analysis. [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)

History of integer multiplication

year	algorithm	bit operations
antiquity	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	$n \log n \, 2^{O(\log^* n)}$
2021	Harvey–van der Hoeven	$O(n \log n)$
	???	$O(n)$

number of bit operations to multiply two n-bit integers

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

GMP
«Arithmetic without limitations»

used in Maple, Mathematica, gcc, cryptography, …

CSCI 355: Algorithm Design and Analysis
6. Divide and Conquer II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*

Dot product

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

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

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

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

Remark. Grade school dot product algorithm is asymptotically optimal.

Matrix multiplication

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 algorithm. $\Theta(n^3)$ 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 asymptotically optimal?

Block matrix multiplication

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

C_{11} A_{11} A_{12} B_{11} B_{21}

Block matrix multiplication

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

$$C = A \times B$$

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

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. We can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

scalars

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

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

7 scalar multiplications

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

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

Strassen's trick

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

~~scalars~~ $\frac{1}{2}n$-by-$\frac{1}{2}n$ 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 ~~scalar~~ matrix multiplications
(of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices)

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

Block matrix multiplication: Strassen's algorithm

STRASSEN(n, A, B) — assume n is a power of 2

IF ($n = 1$) RETURN $A \times B$.
Partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
$P_1 \leftarrow$ 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.

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

Strassen's algorithm: analysis

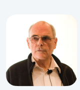
Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two n-by-n matrices.

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.


Strassen

Strassen's algorithm: analysis

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

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

- Research has reported an 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

Numeric linear algebra reductions

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	$A = LU$	$\Theta(MM(n))$
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

Fast matrix multiplication

Q. Can we multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Q. Can we multiply two 2-by-2 matrices with 6 scalar multiplications?
A. Impossible! [Hopcroft–Kerr, Winograd 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Race to n^2. [Pan 1978, Bini et al. 1979, Schönhage 1981, ...]

- Two 70-by-70 matrices with 143,640 scalar multiplications. $O(n^{2.7962})$
- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$

History of matrix multiplication

year	algorithm	arithmetic operations
1858	**grade school**	$O(n^3)$
1969	**Strassen**	$O(n^{2.81})$
1978	**Pan**	$O(n^{2.796})$
1979	**Bini–Capovani–Romani**	$O(n^{2.780})$
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	**Stothers**	$O(n^{2.3737})$
2011	**Williams**	$O(n^{2.3729})$
2014	**Le Gall**	$O(n^{2.3728639})$
2020	**Alman–Williams**	$O(n^{2.3728596})$
2022	**Duan–Wu–Zhou**	$O(n^{2.371866})$
2024	**Williams–Xu–Xu–Zhou**	$O(n^{2.371552})$
	???	$O(n^{2+\varepsilon})$

galactic algorithms

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