How to Multiply

integers, matrices, and polynomials

COS 423
Spring 2007

slides by Kevin Wayne

Complex Multiplication

Complex multiplication. $(a + bi)(c + di) = x + yi.$

Grade-school. $x = ac - bd, \; y = bc + ad.$

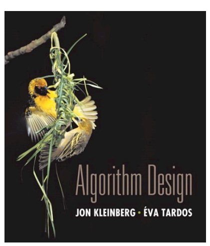
↖ 4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?
A. Yes. [Gauss] $x = ac - bd, \; y = (a + b)(c + d) - ac - bd.$

↖ 3 multiplications, 5 additions

Remark. Improvement if no hardware multiply.

Integer Multiplication



Section 5.5

Integer Addition

Addition. Given two n-bit integers a and b, compute $a + b$.
Grade-school. $\Theta(n)$ bit operations.

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 algorithm is optimal.

Integer Multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.
Grade-school. $\Theta(n^2)$ 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
```

Q. Is grade-school multiplication algorithm optimal?

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers a and b:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$a = 2^{n/2} \cdot a_1 + a_0$$
$$b = 2^{n/2} \cdot b_1 + b_0$$
$$ab = \left(2^{n/2} \cdot a_1 + a_0\right)\left(2^{n/2} \cdot b_1 + b_0\right) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0$$

Ex. $a = \underbrace{1000}_{a_1}\underbrace{1101}_{a_0}$ $\quad b = \underbrace{1110}_{b_1}\underbrace{0001}_{b_0}$

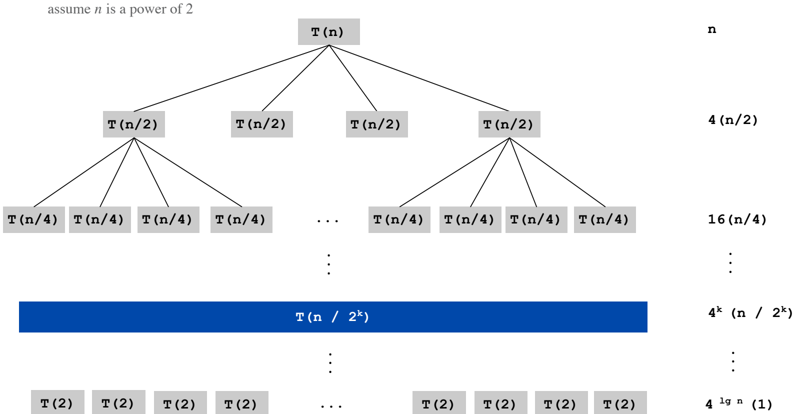
$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

assume n is a power of 2

$$T(n) = \sum_{k=0}^{\lg n} n\, 2^k = n\left(\frac{2^{1+\lg n} - 1}{2-1}\right) = 2n^2 - n$$



Karatsuba Multiplication

To multiply two n-bit integers a and b:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$a = 2^{n/2} \cdot a_1 + a_0$$
$$b = 2^{n/2} \cdot b_1 + b_0$$
$$ab = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0$$
$$= 2^n \cdot \underset{①}{a_1 b_1} + 2^{n/2} \cdot \left(\underset{②}{(a_1 + a_0)(b_1 + b_0)} - \underset{①}{a_1 b_1} - \underset{③}{a_0 b_0}\right) + \underset{③}{a_0 b_0}$$

Karatsuba Multiplication

To multiply two n-bit integers a and b:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$
\begin{aligned}
a &= 2^{n/2} \cdot a_1 + a_0 \\
b &= 2^{n/2} \cdot b_1 + b_0 \\
ab &= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0 \\
&= 2^n \cdot \underset{1}{a_1 b_1} + 2^{n/2} \cdot \left(\underset{2}{(a_1 + a_0)(b_1 + b_0)} - \underset{1}{a_1 b_1} - \underset{3}{a_0 b_0} \right) + \underset{3}{a_0 b_0}
\end{aligned}
$$

Theorem. [Karatsuba-Ofman 1962] Can multiply two n-bit integers in $O(n^{1.585})$ bit operations.

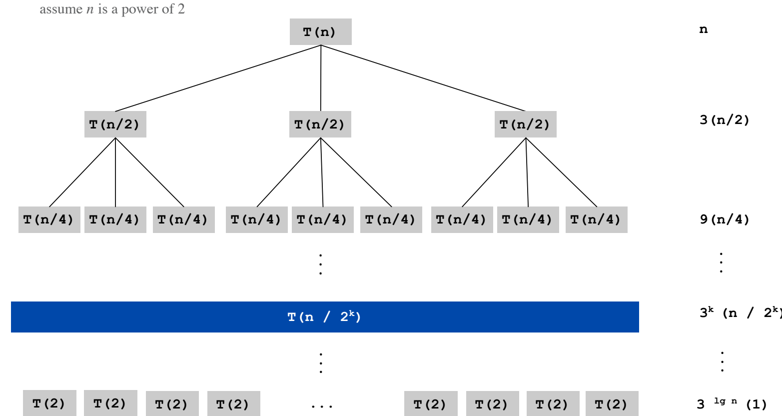
$$
T(n) \le \underbrace{T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + T\left(1 + \lceil n/2 \rceil\right)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \Rightarrow T(n) = O(n^{\lg 3}) = O(n^{1.585})
$$

Karatsuba: Recursion Tree

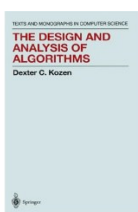
$$
T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}
$$

assume n is a power of 2

$$
T(n) = \sum_{k=0}^{\lg n} n \left(\tfrac{3}{2}\right)^k = n \left(\frac{\left(\frac{3}{2}\right)^{1 + \lg n} - 1}{\frac{3}{2} - 1} \right) = 3n^{\lg 3} - 2n
$$



Integer Division



Section 30.3

Integer Division

Integer division. Given two integer s and t of at most n bits each, compute the quotient and remainder: $q = \lfloor s / t \rfloor$, $r = s \bmod t$.

Ex.

- $s = 1{,}000,\ t = 110 \Rightarrow q = 9, r = 10$.
- $s = 4{,}905{,}648{,}605{,}986{,}590{,}685,\ t = 100 \Rightarrow r = 85$.

Long division. $\Theta(n^2)$.

Q. Is grade-school long division algorithm optimal?

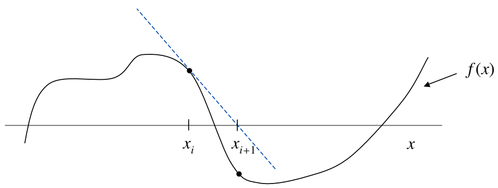
Newton's Method

Goal. Given a function $f(x)$, find a value x^* such that $f(x^*) = 0$.

sufficiently smooth

Newton's method.

- Start with initial guess x_0.
- Compute a sequence of approximations: $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$.



Convergence. No guarantees in general.

Integer Division: Newton's Method

Goal. Given two integer s and t compute $q = \lfloor s / t \rfloor$.

Our approach: Newton's method.

- Approximately compute $x = 1 / t$ using exact arithmetic.

$$f(x) = t - \frac{1}{x}$$
$$x_{i+1} = 2x_i - t x_i^2$$

- After not too many iterations, quotient q is either $\lfloor s\, x_k \rfloor$ or $\lceil s\, x_k \rceil$.

Integer Division: Newton's Method Example

Ex. `t = 7`

- x_0 = 0.1
- x_1 = 0.13
- x_2 = 0.1417
- x_3 = 0.142847770
- x_4 = 0.14285714224218970
- x_5 = 0.14285714285714285449568544449737
- x_6 = 0.14285714285714285714285714285714280809023113867839307631644158170

number of digits of accuracy doubles after each iteration

$$f(x) = t - \frac{1}{x}$$
$$x_{i+1} = 2x_i - t x_i^2$$

Ex. `s/t = 123456/7`

- $s\, x_5$ = 17636.571428571428244619342235867310720000
- Correct answer is either `17636` or `17637`.

Integer Division: Newton's Method

(q, r) = NewtonDivision$(s,\ t)$

Choose x to be unique fractional power of 2 in interval $(1 / (2t),\ 1 / t\,]$

repeat $\lg n$ times
    $x \leftarrow 2x - t\,x^2$

set $q = \lfloor s\,x \rfloor$ **or** $q = \lceil s\,x \rceil$
set $r = s - q\,t$

Analysis

L1. Iterates converge monotonically.

$$\frac{1}{2t} < x_0 \le x_1 \le x_2 \le \cdots \le \frac{1}{t}.$$

Pf. [by induction on i]

- Base case: by construction, $\frac{1}{2t} < x_0 \le \frac{1}{t}$
- Inductive hypothesis: $\frac{1}{2t} < x_0 \le x_1 \le \cdots \le x_i \le \frac{1}{t}$.

$$\begin{aligned} x_{i+1} &= 2x_i - t\,x_i^2 \\ &= x_i(2 - t\,x_i) \\ &\ge x_i(2 - t\,(1/t)) \\ &= x_i \end{aligned}$$

inductive hypothesis

(monotonic)

$$\begin{aligned} x_{i+1} &= 2x_i - t\,x_i^2 \\ &= (2x_i - t\,x_i^2 - 1/t) + 1/t \\ &= -t(x_i - 1/t)^2 + 1/t \\ &\le 1/t \end{aligned}$$

(bounded)

Analysis

L2. Iterates converge quadratically to $1/t$: $1 - tx_i < \frac{1}{2^{2^i}}$.

x_i is approximates $1/t$ to 2^i significant bits of accuracy

Pf. [by induction on i]

- Base case: by construction, $\frac{1}{2t} < x_0 \;\Rightarrow\; 1 - t\,x_0 < \frac{1}{2}$
- Inductive hypothesis: $1 - t\,x_i < \frac{1}{2^{2^i}}$

$$\begin{aligned} 1 - t\,x_{i+1} &= 1 - t\,(2x_i - t\,x_i^2) \\ &= (1 - t\,x_i)^2 \\ &< \left(\frac{1}{2^{2^i}}\right)^2 \\ &= \frac{1}{2^{2^{i+1}}} \end{aligned}$$

inductive hypothesis

Analysis

L3. Algorithm returns correct answer.

Pf.

- By L2, after $k = \lceil \lg \lg (s/t) \rceil$ steps, we have: $1 - tx_k < \frac{1}{2^{2^k}} \le \frac{t}{s}$.

L2 *choice of k*

- Thus, $0 \le \frac{s}{t} - sx_k < 1$

$x_k \le 1/t$ by L1 *rearranging expression above*

- This implies, $q = \lfloor s/t \rfloor$ is either $\lfloor s\,x_k \rfloor$ or $\lceil s\,x_k \rceil$.
- Note: $k \le \lg n$.

Analysis

Theorem. Algorithm computes quotient and remainder in $O(M(n))$ time, where $M(n)$ is the time to multiply two n-bit integers.

Pf.

- The number of iterations is $k = \lg n$.
- By L2, the algorithm returns the correct answer.
- Each iterate involves $O(1)$ multiplications and additions.

$$\begin{aligned} f(x) &= t - \frac{1}{x} \\ x_{i+1} &= 2x_i - tx_i^2 \end{aligned}$$

- Note: algorithm still works if we only keep track of 2^i significant digits in iteration i.
- Overall running time: $M(1) + M(2) + M(4) + \ldots + M(2^k) = O(M(n))$.

Analysis

Theorem. Algorithm computes quotient and remainder in $O(M(n))$ time, where $M(n)$ is the time to multiply two n-bit integers.

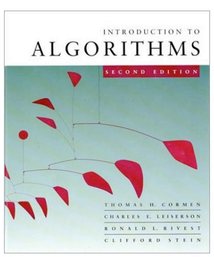
Corollary. Can do integer division in $O(n^{1.585})$ bit operations.

Integer Arithmetic

Theorem. The following have the same asymptotic bit complexity.

- Multiplication.
- Squaring.
- Quotient.
- Remainder.

Matrix Multiplication



Chapter 28.2

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 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 optimal?

Block Matrix Multiplication

C_{11}, A_{11}, A_{12}, B_{11}, B_{11}

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

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

Matrix Multiplication: Warmup

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

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

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

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

Fast Matrix Multiplication

Key idea. multiply 2-by-2 blocks with only 7 multiplications.

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

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

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

- 7 multiplications.
- $18 = 8 + 10$ additions and subtractions.

Fast Matrix Multiplication

To multiply two n-by-n matrices A and B: [Strassen 1969]
- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Compute: 14 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.
- Assume n is a power of 2.
- $T(n)$ = # arithmetic operations.

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

Fast Matrix Multiplication: Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misperception. *"Strassen is only a theoretical curiosity."*
- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2{,}500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" $Ax = b$, determinant, eigenvalues, SVD, ….

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 and Kerr 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Q. Two 3-by-3 matrices with 21 scalar multiplications?
A. Also impossible. $\Theta(n^{\log_3 21}) = O(n^{2.77})$

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. $O(n^{2.521801})$

Fast Matrix Multiplication: Theory

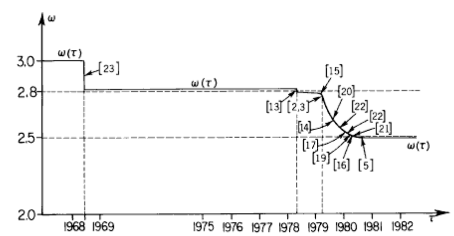


FIG. 1. $\omega(\tau)$ *is the best exponent announced by time* τ.

Best known. $O(n^{2.376})$ [Coppersmith-Winograd 1987]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.