Algorithm Design and Analysis

CSE 565

LECTURE 15

Divide and Conquer

- Integer Multiplication
  - Karatsuba's algorithm
- Matrix Multiplication
  - Strassen's algorithm

Sofya Raskhodnikova

Review: use asymptotic notation

- The statement "my alogirhtm takes time $2n$" is not accurate.
  - Do you mean $2n$ seconds? $2n$ minutes?
  - The exact running time depends on the processor!
- Use asymptotic notation: "my alogirhtm takes time $\Theta(n)$". It means that the running time is proportional to n.
- The same applies to space complexity.

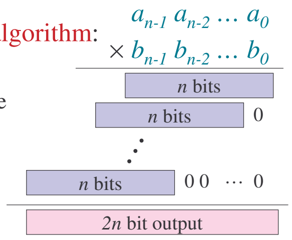
Last time: order statistics

- Deterministic O(n) time algorithm
- There is a randomized algorithm that runs in expected linear time.
- The randomized algorithm is far more practical.

Exercise: *Why not divide into groups of 3?*

Multiplying large integers

- Given n-bit integers a, b (in binary), compute $c=ab$
- **Naive** (grade-school) algorithm:
  - Write a,b in binary
  - Compute n intermediate products
  - Do n additions
  - Total work: $\Theta(n^2)$

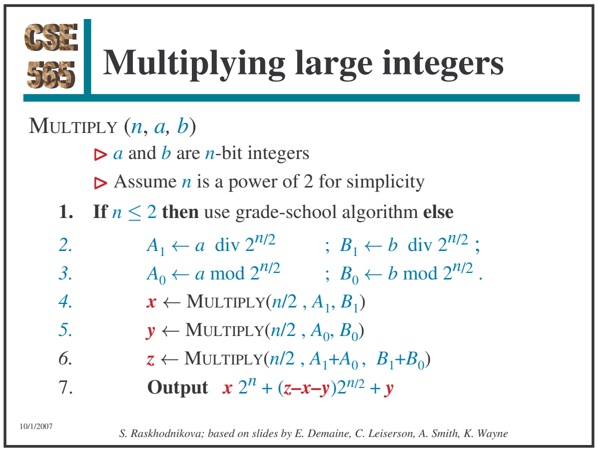


Multiplying large integers

- **Divide and Conquer** (Attempt #1):
  - Write $a = A_1\, 2^{n/2} + A_0$
    $b = B_1\, 2^{n/2} + B_0$
  - We want $ab = A_1B_1\, 2^n + (A_1B_0 + B_1A_0)\, 2^{n/2} + A_0B_0$
  - Multiply $n/2$ –bit integers recursively
  - $T(n) = 4T(n/2) + \Theta(n)$
  - Alas! this is still $\Theta(n^2)$ (Master Theorem, Case 1)

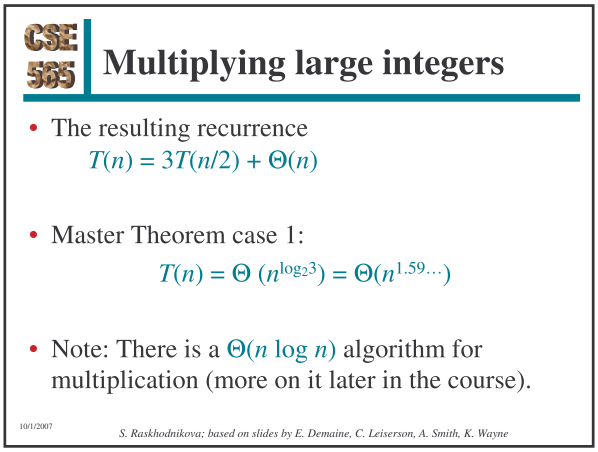
Multiplying large integers

- **Divide and Conquer** (Attempt #1):
  - Write $a = A_1\, 2^{n/2} + A_0$
    $b = B_1\, 2^{n/2} + B_0$
  - We want $ab = A_1B_1\, 2^n + (A_1B_0 + B_1A_0)\, 2^{n/2} + A_0B_0$
  - Karatsuba's idea:
    $(A_0+A_1)\,(B_0 + B_1) = A_0B_0 + A_1B_1 + (A_0B_1 + B_1A_0)$
  - We can get away with 3 multiplications! (in yellow)
    $\boldsymbol{x} = A_1B_1 \quad \boldsymbol{y} = A_0B_0 \quad \boldsymbol{z} = (A_0+A_1)(B_0+B_1)$
  - Now we use $ab = A_1B_1\, 2^n + (A_1B_0 + B_1A_0)\, 2^{n/2} + A_0B_0$
    $= \boldsymbol{x}\ 2^n + (\boldsymbol{z}-\boldsymbol{x}-\boldsymbol{y})\, 2^{n/2} + \boldsymbol{y}$



Multiplying large integers

```
MULTIPLY ($n$, $a$, $b$)
        ▷ $a$ and $b$ are $n$-bit integers
        ▷ Assume $n$ is a power of 2 for simplicity
    If $n \le 2$ then use grade-school algorithm else
        $A_1 \leftarrow a$ div $2^{n/2}$ ; $B_1 \leftarrow b$ div $2^{n/2}$ ;
        $A_0 \leftarrow a$ mod $2^{n/2}$ ; $B_0 \leftarrow b$ mod $2^{n/2}$ .
        $x \leftarrow$ MULTIPLY($n/2$ , $A_1$, $B_1$)
        $y \leftarrow$ MULTIPLY($n/2$ , $A_0$, $B_0$)
        $z \leftarrow$ MULTIPLY($n/2$ , $A_1+A_0$ , $B_1+B_0$)
        Output $x\,2^n + (z-x-y)2^{n/2} + y$
```



Multiplying large integers

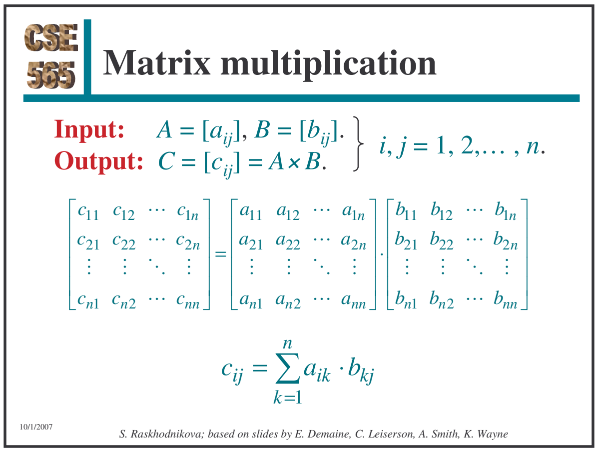
- The resulting recurrence

$$T(n) = 3T(n/2) + \Theta(n)$$

- Master Theorem case 1:

$$T(n) = \Theta\,(n^{\log_2 3}) = \Theta(n^{1.59\ldots})$$

- Note: There is a $\Theta(n \log n)$ algorithm for multiplication (more on it later in the course).

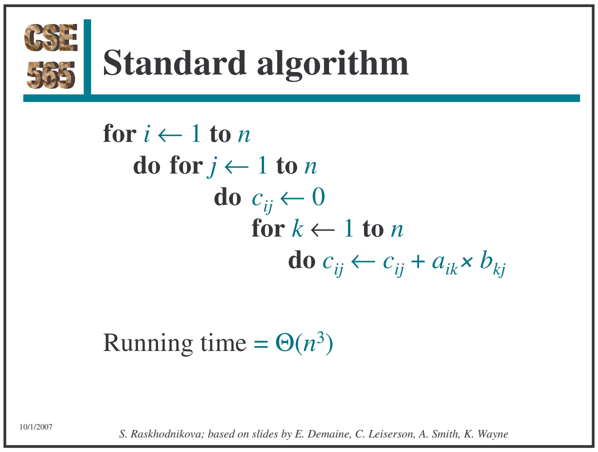


Matrix multiplication

Input: $A = [a_{ij}]$, $B = [b_{ij}]$.
Output: $C = [c_{ij}] = A \times B$.
$\Big\}\; i, j = 1, 2,\ldots , n.$

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

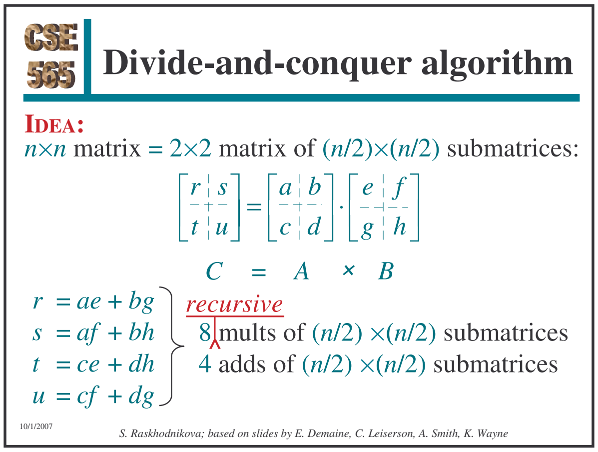
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$



Standard algorithm

```
for $i \leftarrow 1$ to $n$
    do for $j \leftarrow 1$ to $n$
        do $c_{ij} \leftarrow 0$
            for $k \leftarrow 1$ to $n$
                do $c_{ij} \leftarrow c_{ij} + a_{ik} \times b_{kj}$
```

Running time = $\Theta(n^3)$



Divide-and-conquer algorithm

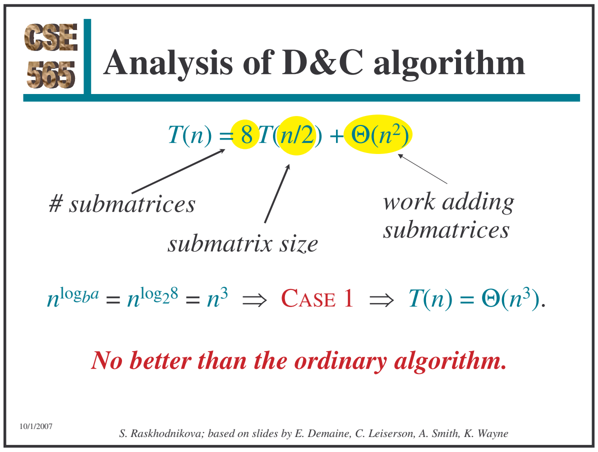
IDEA:
$n \times n$ matrix = 2×2 matrix of $(n/2)\times(n/2)$ submatrices:

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \times \quad B$$

$r = ae + bg$
$s = af + bh$
$t = ce + dh$
$u = cf + dg$

recursive
8 mults of $(n/2) \times (n/2)$ submatrices
4 adds of $(n/2) \times (n/2)$ submatrices



Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\;$ CASE 1 $\;\Rightarrow\; T(n) = \Theta(n^3)$.

No better than the ordinary algorithm.

CSE 565

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$\begin{aligned}
P_1 &= a \times (f - h) & r &= P_5 + P_4 - P_2 + P_6 \\
P_2 &= (a + b) \times h & s &= P_1 + P_2 \\
P_3 &= (c + d) \times e & t &= P_3 + P_4 \\
P_4 &= d \times (g - e) & u &= P_5 + P_1 - P_3 - P_7 \\
P_5 &= (a + d) \times (e + h) \\
P_6 &= (b - d) \times (g + h) \\
P_7 &= (a - c) \times (e + f)
\end{aligned}$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of multiplication!

CSE 565

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$\begin{aligned}
P_1 &= a \times (f - h) \\
P_2 &= (a + b) \times h \\
P_3 &= (c + d) \times e \\
P_4 &= d \times (g - e) \\
P_5 &= (a + d) \times (e + h) \\
P_6 &= (b - d) \times (g + h) \\
P_7 &= (a - c) \times (e + f)
\end{aligned}$$

$$\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}$$

CSE 565

Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)$ x $(n/2)$ submatrices. Form terms to be multiplied using + and – .
2. ***Conquer:*** Perform 7 multiplications of $(n/2)$ x $(n/2)$ submatrices recursively.
3. ***Combine:*** Form product matrix C using + and – on $(n/2)$ x $(n/2)$ submatrices.

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

CSE 565

Analysis of Strassen

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

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^{\lg 7}).$$

•Number 2.81 may not seem much smaller than 3.
•But the difference is in the exponent.
•The impact on running time is significant.
•Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.