

Lecture 7: Divide & Conquer 2

Integer Multiplication

& Matrix Multiplication

CS 341: Algorithms

Tuesday, Jan 29th 2019

Outline For Today

1. Integer Multiplication
2. Matrix Multiplication

Outline For Today

1. Integer Multiplication
2. Matrix Multiplication

Integer Multiplication

- ◆ Input: 2 n -digit integers, X , Y
- ◆ Output $Z = XY$
- ◆ E.g: $X = 2345$ and $Y = 6789$ then $Z = 15920205$
- ◆ Will work in base 10 b/c it is easier to think about base 10
- ◆ Same argument & analysis as in base 2 (done in handout)
- ◆ Warning: There is no intuition to the DC algorithm we'll see

Grade School Algorithm

$$\begin{array}{r} 2345 \\ 6789 \\ \times \\ \hline 21105 \end{array}$$

Grade School Algorithm

Diagram illustrating the multiplication of 2345 by 6789. The first step shows the partial product 21105, which is the result of 2345 multiplied by 9. A red arrow points from the text "shift by 1 digit" to the next line of the multiplication, indicating that the next partial product will be shifted one digit to the left.

Grade School Algorithm

$$\begin{array}{r} \overline{2345} \\ 6789 \\ \hline 21105 \\ 18760 \leftarrow \text{shift by 1 digit} \\ 16415 \leftarrow \text{shift by 2 digits} \end{array}$$

Grade School Algorithm

$$\begin{array}{r} \begin{array}{c} \boxed{2}\boxed{3}\boxed{4}\boxed{5} \\ \boxed{6}\boxed{7}\boxed{8}\boxed{9} \\ \times \end{array} \\ \hline \begin{array}{r} 21105 \\ 18760 \leftarrow \text{shift by 1 digit} \\ 16415 \leftarrow \text{shift by 2 digits} \\ 14070 \leftarrow \text{shift by 3 digits} \\ + \\ \hline 15920205 \end{array} \end{array}$$

Observe that even the total work for shifting is $(1+2+3 + \dots + n-1) = O(n^2)$!

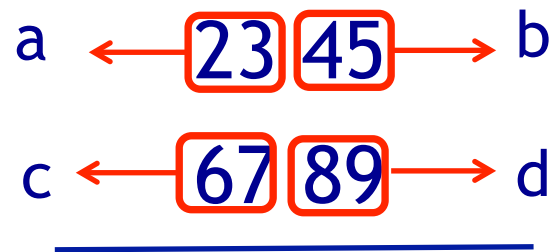
- ◆ Multiplying two digits \Rightarrow 1 operation
- ◆ Addition of two digits \Rightarrow 1 operation
- ◆ Shift by 1 digit \Rightarrow 1 operation (shift by x digits, x ops)

TOTAL WORK: $O(n^2)$

Question: Can we do better?

◆ Upshot: A set of mysterious looking multiplications, additions and subtractions giving the correct answer!

◆ $X = (a10^{n/2} + b); Y = (c10^{n/2} + d)$



each of a, b, c, d
is of size $n/2$

◆ Ex: $a=23, b=45$, so $2345 = 23 \cdot 10^{n/2} + 45$

◆ $XY = ac10^n + ad10^{n/2} + bc10^{n/2} + bd$

◆ $XY = \boxed{ac10^n + (ad + bc)10^{n/2} + bd}$

Call this
expression (★)

DC-Multiplication-1

procedure DC-Mult1(X, Y both n digit numbers):

Base Case: if (X or Y is single digit): ...

set a, b, c, d defined as before $\longrightarrow O(n)$

ac = DC-Mult(a, c)

ad = DC-Mult(a, d)

bc = DC-Mult(b, c)

bd = DC-Mult(b, d)

return $ac10^n + (ad + bc)10^{n/2} + bd$

shifts: $O(n)$

3 additions of n digit numbers: $O(n)$

Total Work Outside of Recursive Calls: $O(n)$

Recurrence: $4T(n/2) + O(n)$

Total Runtime: $O(n^2)$

Not better than Grade School Algorithm

Observation

Observation: We care about only 3 quantities in (★):

$$ac10^n + (ad + bc)10^{n/2} + bd$$

- (1) $ac \Rightarrow$ with 10^n padding
- (2) $(ad + bc) \Rightarrow$ with $10^{n/2}$ padding
- (3) $bd \Rightarrow$ with 10 padding

Question: If we care about 3 quantities, can we get these quantities with only 3 recursive calls?

Karatsuba-Ofman Algorithm (1962)

$$\boxed{ac}10^n + \boxed{(ad + bc)}10^{n/2} + \boxed{bd}$$

(1) ac as before

(2) bd as before

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

Observation: $(3) - (2) - (1) = (ac+ad+bc+bd)-ac-bd = ad + bc$

procedure $KO(X, Y \text{ both } n \text{ digit numbers})$:

Base Case: if $(X \text{ or } Y \text{ is single digit})$: ...

set a, b, c, d defined as before

$ac = KO(a, c)$

$bd = KO(b, d)$

$X = KO(a+b, c+d)$

return $(ac)10^n + (X-ac-bd)10^{n/2} + bd$

Karatsuba-Ofman Algorithm (1962)

procedure KO(X, Y both n digit numbers):
Base Case: if (X or Y is single digit): ...
set a, b, c, d defined as before $\longrightarrow O(n)$
ac = KO(a, c)
bd = KO(b, d) $\longrightarrow O(n)$
X = KO(a+b, c+d)
return (ac) 10^n + (X - ac - bd) $10^{n/2}$ + bd $\longrightarrow O(n)$

Total Work Outside of Recursive Calls: $O(n)$

Recurrence: $3T(n/2) + O(n)$

Total Runtime: $O(n^{\log_2(3)} = n^{1.59})$

Facts About Multiplication

Fact 1: (By Toom & Stephen Cook): Can generalize KO to divide X and Y into k pieces instead of 2 (Math gets very messy) and get

$$O(n^{\log_k(2k-1)}) \Rightarrow O(n^{1+\varepsilon}) \text{ for any } \varepsilon > 0$$

Stephen Cook is a Canadian Computer Scientist currently @ UToronto.

Fact 2: Best known alg: $O(n \log(n) \log \log(n))$ (Schönhage & Strassen)

Outline For Today

1. Integer Multiplication
2. Matrix Multiplication

Matrix Multiplication

◆ Input: 2 $n \times n$ matrices A, B

◆ Output: $C = A \times B$

The diagram shows the multiplication of two $n \times n$ matrices, A and B, to produce an $n \times n$ matrix C. Matrix A has elements $a_{11}, a_{12}, \dots, a_{1n}$ in its first row, and $a_{n1}, a_{n2}, \dots, a_{nn}$ in its last row. A red box highlights the i -th row of A, with the index i written to the left. Matrix B has elements $b_{11}, b_{12}, \dots, b_{1n}$ in its first column, and $b_{n1}, b_{n2}, \dots, b_{nn}$ in its last column. A red box highlights the j -th column of B, with the index j written above. Matrix C has elements $c_{11}, c_{12}, \dots, c_{1n}$ in its first row, and $c_{n1}, c_{n2}, \dots, c_{nn}$ in its last row. A red box highlights the c_{ij} element in matrix C. The multiplication is represented by $A \times B = C$.

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} = (a_{i1} b_{1j} + a_{i2} b_{2j} + \dots + a_{in} b_{nj})$$

Q: By definition, how many multip. and additions needed to compute c_{ij} ?

A: n multiplications, $n-1$ additions

Then there are $O(n^3)$ basic operations (by definition) to compute C.

◆ Warning: Again, no intuition to the DC algorithm we'll see!

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \quad \times \quad \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} \quad = \quad \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \quad \times \quad \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} \quad = \quad \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

a_{11}	a_{12}	a_{1n}
a_{21}	a_{22}	a_{2n}
...
...
a_{n1}	a_{n2}	a_{nn}

 \times

b_{11}	b_{12}	b_{1n}
b_{21}	b_{22}	b_{2n}
...
...
b_{n1}	b_{n2}	b_{nn}

 $=$

c_{11}	c_{12}	c_{1n}
c_{21}	c_{22}	c_{2n}
...
...
c_{n1}	c_{n2}	c_{nn}

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \quad \times \quad \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} = \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \quad \times \quad \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} = \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \times \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} = \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ \boxed{a_{21}} & \boxed{a_{22}} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \times \begin{array}{ccccc} \boxed{b_{11}} & b_{12} & \dots & \dots & b_{1n} \\ \boxed{b_{21}} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \boxed{b_{n1}} & b_{n2} & \dots & \dots & b_{nn} \end{array} = \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ \boxed{c_{21}} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

Standard Algorithm

a_{11}	a_{12}	a_{1n}
a_{21}	a_{22}	a_{2n}
...
...
a_{n1}	a_{n2}	a_{nn}

 \times

b_{11}	b_{12}	b_{1n}
b_{21}	b_{22}	b_{2n}
...
...
b_{n1}	b_{n2}	b_{nn}

 $=$

c_{11}	c_{12}	c_{1n}
c_{21}	c_{22}	c_{2n}
...
...
c_{n1}	c_{n2}	c_{nn}

The diagram illustrates the standard algorithm for matrix multiplication. It shows three matrices: a matrix A with elements a_{ij} , a matrix B with elements b_{ij} , and their product matrix C with elements c_{ij} . The matrix A is a 5x5 grid with elements $a_{11}, a_{12}, \dots, a_{1n}$ in the first row, $a_{21}, a_{22}, \dots, a_{2n}$ in the second row, and $a_{n1}, a_{n2}, \dots, a_{nn}$ in the last row. The matrix B is a 5x5 grid with elements $b_{11}, b_{12}, \dots, b_{1n}$ in the first row, $b_{21}, b_{22}, \dots, b_{2n}$ in the second row, and $b_{n1}, b_{n2}, \dots, b_{nn}$ in the last row. The matrix C is a 5x5 grid with elements $c_{11}, c_{12}, \dots, c_{1n}$ in the first row, $c_{21}, c_{22}, \dots, c_{2n}$ in the second row, and $c_{n1}, c_{n2}, \dots, c_{nn}$ in the last row. The multiplication is indicated by a large \times symbol between A and B , and the result is indicated by an equals sign $=$ between B and C . Red boxes highlight the elements $a_{21}, a_{22}, \dots, a_{2n}$ in the second row of A , $b_{12}, b_{22}, \dots, b_{n2}$ in the second column of B , and c_{22} in the second row and second column of C , illustrating the dot product of the second row of A and the second column of B to compute the element c_{22} .

Standard Algorithm

$$\begin{array}{ccccc} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{array} \times \begin{array}{ccccc} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{array} = \begin{array}{ccccc} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{array}$$

procedure StandardMM(A, B nxn matrices):

$C = \text{int}[n][n];$

for $i = 1..n$:

for $j = 1..n$:

$C[i][j] = 0;$

for $k=1..n$:

$C[i][j] += a_{ik}b_{kj}$

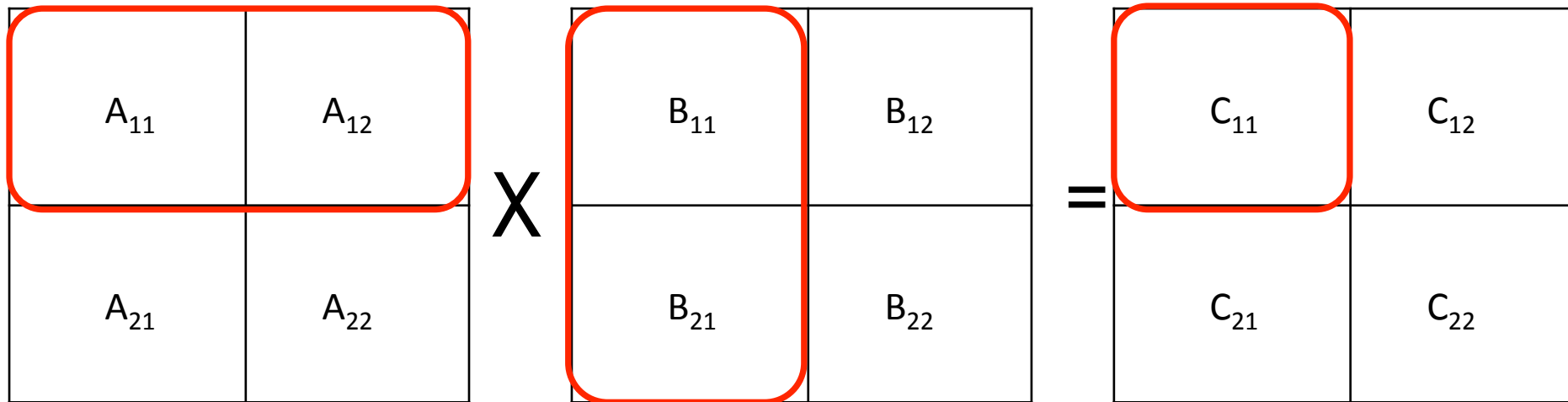
return C

Runtime: $O(n^3)$

Question: Can we Divide & Conquer?

$$\begin{array}{|ccccc|} \hline a_{11} & a_{12} & \dots & \dots & a_{1n} \\ \hline a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline a_{n1} & a_{n2} & \dots & \dots & a_{nn} \\ \hline \end{array} \quad \times \quad \begin{array}{|ccccc|} \hline b_{11} & b_{12} & \dots & \dots & b_{1n} \\ \hline b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline b_{n1} & b_{n2} & \dots & \dots & b_{nn} \\ \hline \end{array} \quad = \quad \begin{array}{|ccccc|} \hline c_{11} & c_{12} & \dots & \dots & c_{1n} \\ \hline c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline c_{n1} & c_{n2} & \dots & \dots & c_{nn} \\ \hline \end{array}$$

Question: Can we Divide & Conquer?



◆ Where $A_{11}, \dots, A_{22}, B_{11}, \dots, B_{22}$, & C_{11}, \dots, C_{22} are $n/2 \times n/2$ matrices.

Fact: When you split matrices into blocks & multiply, the blocks behave as they are atomic elements.

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj} = (A_{i1} B_{1j} \boxed{+} A_{i2} B_{2j} \boxed{+} \dots \boxed{+} A_{in} B_{nj})$$

where $+$ is matrix addition operation (coordinate-wise addition)

Matrix Addition

$$\begin{bmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{bmatrix}$$

Matrix Addition

$$\begin{bmatrix} a_{11} & \boxed{a_{12}} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & \dots & a_{nn} \end{bmatrix} + \begin{bmatrix} b_{11} & \boxed{b_{12}} & \dots & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \dots & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & \dots & b_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & \boxed{c_{12}} & \dots & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & \dots & c_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & \dots & c_{nn} \end{bmatrix}$$

Matrix Addition

a_{11}	a_{12}	a_{1n}
a_{21}	a_{22}	a_{2n}
...
...
a_{n1}	a_{n2}	a_{nn}

 $+$

b_{11}	b_{12}	b_{1n}
b_{21}	b_{22}	b_{2n}
...
...
b_{n1}	b_{n2}	b_{nn}

 $=$

c_{11}	c_{12}	c_{1n}
c_{21}	c_{22}	c_{2n}
...
...
c_{n1}	c_{n2}	c_{nn}

$O(n^2)$ operation

Example of Block Multiplication

2	4	3	2
3	5	4	2
6	5	9	3
2	2	3	5

X

3	2	5	2
2	4	8	4
5	2	9	3
3	4	2	2

=

$$2*3 + 4*2 + 3*5 + 2*3 = 35$$

Example of Block Multiplication

2	4	3	2
3	5	4	2
6	5	9	3
2	2	3	5

X

3	2	5	2
2	4	8	4
5	2	9	3
3	4	2	2

=

35			

$$2*2 + 4*4 + 3*2 + 2*4 = 34$$

Example of Block Multiplication

2	4	3	2
3	5	4	2
6	5	9	3
2	2	3	5

X

3	2	5	2
2	4	8	4
5	2	9	3
3	4	2	2

=

35	34		

$$3*3 + 5*2 + 4*5 + 2*3 = 45$$

Example of Block Multiplication

2	4	3	2
3	5	4	2
6	5	9	3
2	2	3	5

X

3	2	5	2
2	4	8	4
5	2	9	3
3	4	2	2

=

35	34		
45			

$$3*2 + 5*4 + 4*2 + 2*4 = 42$$

Example of Block Multiplication

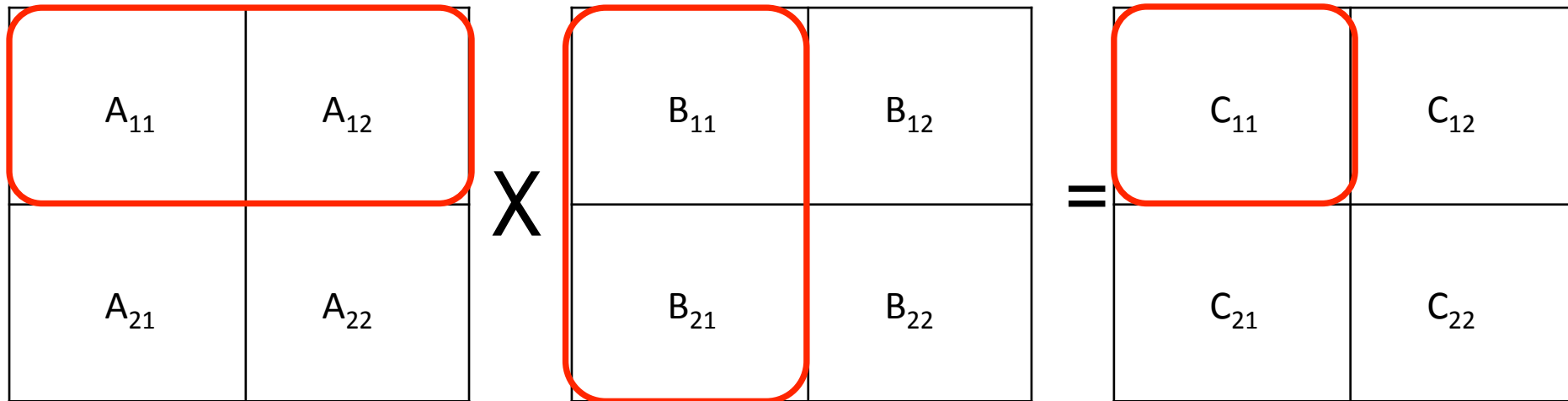
$$\begin{bmatrix} 2 & 4 & 3 & 2 \\ 3 & 5 & 4 & 2 \\ 6 & 5 & 9 & 3 \\ 2 & 2 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} 3 & 2 & 5 & 2 \\ 2 & 4 & 8 & 4 \\ 5 & 2 & 9 & 3 \\ 3 & 4 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 35 & 34 & & \\ 45 & 42 & & \\ & & & \\ & & & \end{bmatrix}$$

$$A_{11}B_{11} = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 14 & 20 \\ 19 & 26 \end{bmatrix}$$

$$A_{12}B_{21} = \begin{bmatrix} 3 & 2 \\ 4 & 2 \end{bmatrix} \times \begin{bmatrix} 5 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 21 & 14 \\ 26 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 14 & 20 \\ 19 & 26 \end{bmatrix} + \begin{bmatrix} 21 & 14 \\ 26 & 16 \end{bmatrix} = \begin{bmatrix} 35 & 34 \\ 45 & 42 \end{bmatrix}$$

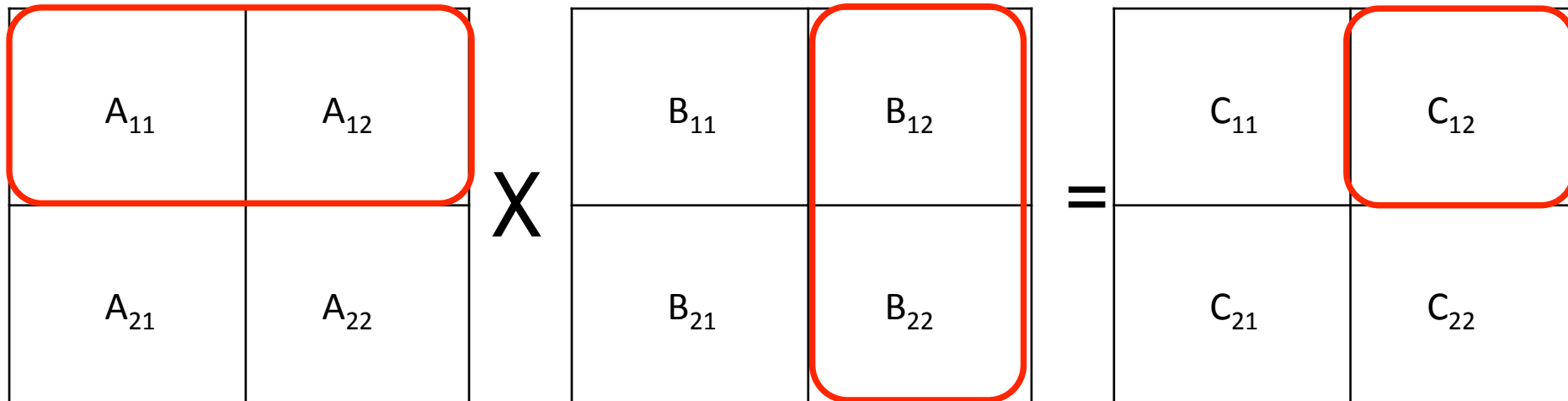
DC-1 Matrix Multiplication



procedure DC1-MM(A, B nxn matrices):

$$C_{11} = \text{DC1-MM}(A_{11}, B_{11}) + \text{DC1-MM}(A_{12}, B_{21})$$

DC-1 Matrix Multiplication

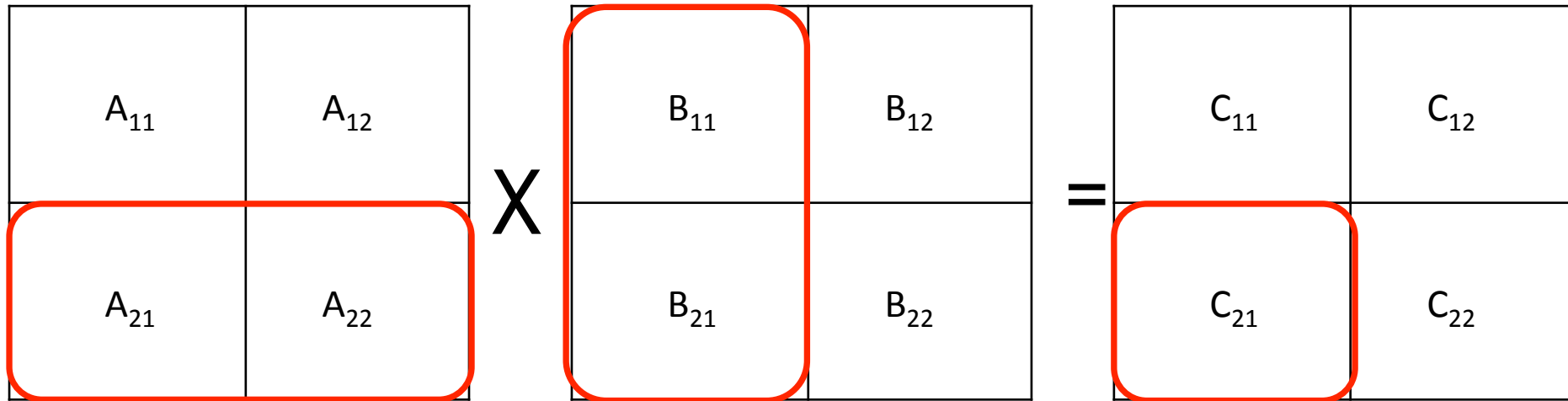


procedure DC1-MM(A, B nxn matrices):

$C_{11} = \text{DC1-MM}(A_{11}, B_{11}) + \text{DC1-MM}(A_{12}, B_{21})$

$C_{12} = \text{DC1-MM}(A_{11}, B_{12}) + \text{DC1-MM}(A_{12}, B_{22})$

DC-1 Matrix Multiplication



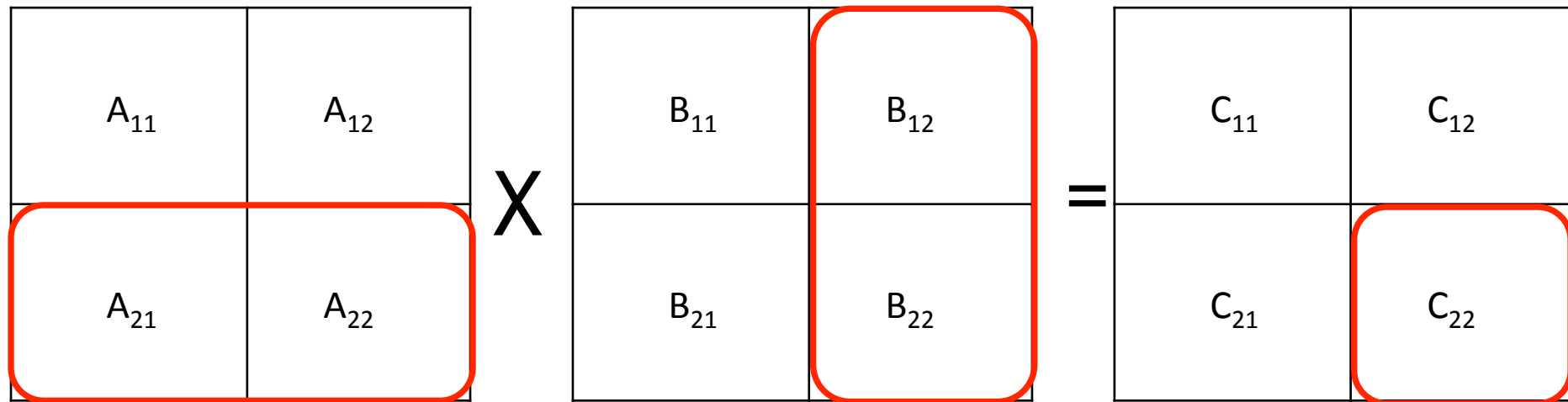
procedure DC1-MM(A, B nxn matrices):

$$C_{11} = \text{DC1-MM}(A_{11}, B_{11}) + \text{DC1-MM}(A_{12}, B_{21})$$

$$C_{21} = \text{DC1-MM}(A_{11}, B_{12}) + \text{DC1-MM}(A_{12}, B_{22})$$

$$C_{21} = \text{DC1-MM}(A_{21}, B_{11}) + \text{DC1-MM}(A_{22}, B_{21})$$

DC-1 Matrix Multiplication



procedure DC1-MM(A, B nxn matrices):

$C_{11} = \text{DC1-MM}(A_{11}, B_{11}) + \text{DC1-MM}(A_{12}, B_{21})$

$C_{21} = \text{DC1-MM}(A_{11}, B_{12}) + \text{DC1-MM}(A_{12}, B_{22})$

$C_{12} = \text{DC1-MM}(A_{21}, B_{11}) + \text{DC1-MM}(A_{22}, B_{21})$

$C_{22} = \text{DC1-MM}(A_{21}, B_{12}) + \text{DC1-MM}(A_{22}, B_{22})$

return $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

Outside recursion: 4 additions of $n/2 \times n/2$ matrices $= 4n^2/4 = n^2$ work

Runtime: $T(n) = 8T(n/2) + O(n^2) = O(n^3)$ (by Master Thm)

So not faster than standard method!

Strassen's (Mysterious) Algorithm (1969)

A_{11}	A_{12}
A_{21}	A_{22}

X

B_{11}	B_{12}
B_{21}	B_{22}

=

$C_{11} = Z_5 + Z_4 - Z_2 + Z_6$	$C_{12} = Z_1 + Z_2$
$C_{21} = Z_3 + Z_4$	$C_{22} = Z_5 + Z_1 - Z_3 - Z_7$

$$\begin{aligned}
 Z_1 &= A_{11}(B_{12} - B_{22}) \\
 Z_2 &= (A_{11} + A_{12})B_{22} \\
 Z_3 &= (A_{21} + A_{22})B_{11} \\
 Z_4 &= A_{22}(B_{21} + B_{11}) \\
 Z_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
 Z_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
 Z_7 &= (A_{11} - A_{21})(B_{11} + B_{12})
 \end{aligned}$$

Exercise: Check the rest of the quadrants to believe Strassen!

$$\text{Ex: } C_{12} = Z_1 + Z_2 = A_{11}B_{12} - A_{11}B_{22} + A_{11}B_{22} + A_{12}B_{22} = A_{11}B_{12} + A_{12}B_{22}$$

Strassen's (Mysterious) Algorithm

A_{11}	A_{12}
A_{21}	A_{22}

X

B_{11}	B_{12}
B_{21}	B_{22}

=

$C_{11} = Z_5 + Z_4 - Z_2 + Z_6$	$C_{12} = Z_1 + Z_2$
$C_{21} = Z_3 + Z_4$	$C_{22} = Z_5 + Z_1 - Z_3 - Z_7$

procedure Strassen(A, B nxn matrices):

$Z_1 = \text{Strassen}(A_{11}, (B_{12} - B_{22})); Z_2 = \text{Strassen}((A_{11} + A_{12}), B_{22})$

$Z_3 = \text{Strassen}((A_{21} + A_{22}), B_{11}); Z_4 = \text{Strassen}(A_{22}, (B_{21} + B_{11}))$

$Z_5 = \text{Strassen}((A_{11} + A_{22}), (B_{11} + B_{22}))$

$Z_6 = \text{Strassen}((A_{12} - A_{22}), (B_{21} + B_{22}))$

$Z_7 = \text{Strassen}((A_{11} - A_{21}), (B_{11} + B_{12}))$

$C_{11} = Z_5 + Z_4 - Z_2 + Z_6; C_{12} = Z_1 + Z_2; C_{21} = Z_3 + Z_4; C_{22} = Z_5 + Z_1 - Z_3 - Z_7$

return C = $C_{11}C_{21}C_{12}C_{22}$

Runtime: $T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7}) = O(n^{2.81})$ (by Master Thm)

(Asymptotically) much faster than standard algorithm!

History of Matrix Multiplication

Runtime	Year	Authors
$O(n^3)$	until 1969	N/A
$O(n^{2.81})$	1969	Strassen
$O(n^{2.796})$	1978	Pan
$O(n^{2.780})$	1979	Bini et. al.
$O(n^{2.522})$	1981	Schoenhage
$O(n^{2.517})$	1982	Romani
$O(n^{2.496})$	1983	Coppersmith & Winograd
$O(n^{2.479})$	1986	Strassen
$O(n^{2.376})$	1989	Coppersmith and Winograd
$O(n^{2.374})$	2011	Stothers
$O(n^{2.3728642})$	2012	V. Williams
$O(n^{2.3728639})$	2013	Le Gall