CS2020
Data Structures and Algorithms
(Recitation)

Welcome!

Today

Proving an algorithm correct:

- Simple examples
- Loop invariants
- Assertions

Divide-and-Conquer Examples

- Integer multiplication
- Matrix multiplication

Getting it right…

How do you show an algorithm correct?

1. What do you mean by correct?
2. What does the algorithm do?

Getting it right…

Example:

calculate(a, b)

```
x = a;
i = a;
while (i < b)
    x += 0.5;
    i++;
return x;
```

Getting it right…

calculate(a, b) : a < b

```
x = a;
i = a;
while (i < b)
      x += 0.5;
      i++;
return x;
```

average(a, b)

```
return (a+b)/2
```

Loop Invariants

Invariant:

- relationship between variables that is always true.

Loop Invariant:

- relationship between variables that is true at the beginning (or end) of each iteration of a loop.

Loop Invariants

1. PREcondition: holds before the loop
2. POSTcondition: holds after the loop
3. Choose loop invariant L.
4. Prove L using induction.
5. Prove that (L + “loop terminates”) => POST
6. Prove that loop terminates

Loop Invariants

calculate(a, b)

```
x = a;
i = a;
while (i < b)
    x += 0.5;
    i++;
return x;
```

PRE: $x = i = a$

POST: $x = (a+b)/2$

L: $x = (a+i)/2$

- Base: $x = (a+a)/2 = a$
- Inductive step:
  - Before: $x = (a+i)/2$
  - After: $x = (a+i)/2 + 0.5$
    $= (a+i+1)/2$

On exit: $i=b \Rightarrow x = (a+b)/2$

Termination: i++ in every iter

Binary Search (review)

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
                end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

Binary Search

Specification:

- Finds element if it is in the array.
- Returns "NO" if it is not in the array

Binary Search

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
            end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

Binary Search

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
            end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

A[begin] == key?

Binary Search

Prove:

- If element is in the array, return it.

Preconditions:

- begin = 1
- end = n

Postcondition:

- A[begin] = key

Binary Search

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
            end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

Binary Search

Loop invariant:

- $A[begin] \leq key \leq A[end]$

Base case:

- $A[begin] = A[1]$
- $A[1]\ ] \leq key \leq A[n]$
- $A[n] = A[end]$

Binary Search

Loop invariant:

- $A[\text{begin}] \leq \text{key} \leq A[\text{end}]$

Inductive step:

- $\text{end} = (\text{begin}+\text{end})/2 - 1$

  if: $A[(\text{begin}+\text{end})/2] > \text{key}$

  thus: $\text{key} \leq A[(\text{begin}+\text{end})/2 - 1] = A[\text{end}]$

- $\text{begin} = (\text{begin}+\text{end})/2$

  if: $A[(\text{begin}+\text{end})/2] \leq \text{key}$

  thus: $A[(\text{begin}+\text{end})/2\ ] = A[\text{begin}] \leq \text{key}$

Binary Search

Loop invariant:

- $A[begin] \le key \le A[end]$

Conclusion:

- Loop exits when (begin==end)
- By invariant: $A[begin] \le key \le A[begin]$
- key == A[begin]

Done

Binary Search

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
            end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

Binary Search

Sorted array: A[1..n]

2	4	4	5	6	7	8	9	11	17	23	28

```
Search(A, key, n)
    begin = 1
    end = n
    while begin != end do:
        if  A[(begin+end)/2] > key  then
                end = (begin+end)/2 – 1
        else begin = (begin+end)/2
    return A[begin]
```

Does not terminate!

Round down?

Assertions

- Imagine you prove a good loop invariant.
  - Yay!
- You implement your algorithm.
  - It works!
- Someone else changes the code.
  - It breaks.
  - Boo!

Assertions

Include the loop invariant in your code!

Example:

$A[begin] \leq key \leq A[end]$

Code:

```
if (A[begin] > key)
        throw new Exception("Bad search");
if (A[end] < key)
        throw new Exception("Bad search");
```

Divide-and-Conquer Examples

Integer Multiplication

			3	**2**	**5**
X			**6**	**9**	**3**

Integer Multiplication

			3	**2**	**5**
X			**6**	**9**	**3**
			9	7	5
	2	9	2	5	
1	9	5	0		
2	2	5	2	2	5

Integer Multiplication

	0	**0**	**1**	**1**	**0**	**0**	**0**	**1**
X	1	1	0	0	1	1	1	0
?	?	?	?	?	?	?	?	?

Given: two n bit binary integers x and y

Compute: xy

Standard strategy: $O(n^2)$

Integer Multiplication

Other operations:

- Addition: O(n)
- Bitwise shift: O(n)

Example:

left-shift(011101) = 111010

right-shift(11101) = 001110

left-shift(x) = 2x

right-shift(x) = x/2

Integer Multiplication

Divide and Conquer

		xL			xR		
x	=	1	0	1	1	1	0
y	=	0	1	1	0	1	1
		yR			yL		

$$x = (xL*2^{n/2} + xR)$$

$$y = (yL*2^{n/2} + yR)$$

Integer Multiplication

Divide and Conquer

		xL			xR	
x =	1	0	1	1	1	0
y =	0	1	1	0	1	1
		yR			yL	

$$xy = (xL*2^{n/2} + xR)(yL*2^{n/2} + yR)$$

Integer Multiplication

Divide and Conquer

			x_L			x_R	
x	=	1	0	1	1	1	0
y	=	0	1	1	0	1	1
			y_R			y_L	

$$xy = (x_L * 2^{n/2} + x_R)(y_L * 2^{n/2} + y_R)$$

Integer Multiplication

Observation:

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

$$\mathbf{xy = (x_L * 2^{n/2} + x_R)(y_L * 2^{n/2} + y_R)}$$

$$xy = x_L y_L 2^n + x_L y_R 2^{n/2} + x_R y_L 2^{n/2} + x_R y_R$$

Integer Multiplication

Observation:

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

$$\mathbf{xy = (x_L * 2^{n/2} + x_R)(y_L * 2^{n/2} + y_R)}$$

$$xy = x_L y_L 2^n + x_L y_R 2^{n/2} + x_R y_L 2^{n/2} + x_R y_R$$

$$T(n) = 4T(n/2) + O(n)$$

Integer Multiplication

Observation:

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

$$\mathbf{xy = (x_L * 2^{n/2} + x_R)(y_L * 2^{n/2} + y_R)}$$

$$xy = x_L y_L 2^n + x_L y_R 2^{n/2} + x_R y_L 2^{n/2} + x_R y_R$$

$$T(n) = 4T(n/2) + O(n)$$
$$= O(n^2)$$

Integer Multiplication

Observation:

$$\mathbf{xy = (x_L * 2^{n/2} + x_R)(y_L * 2^{n/2} + y_R)}$$

$$xy = x_L y_L 2^n + x_L y_R 2^{n/2} + x_R y_L 2^{n/2} + x_R y_R$$

$$T(n) = 4T(n/2) + O(n)$$
$$= O(n^2)$$

Integer Multiplication

Magic: $ab + cd = (a+c)(b+d) - ad - bc$

$$\mathbf{xy = x_L y_L 2^n + (x_L y_R + x_R y_L) 2^{n/2} + x_R y_R}$$

$$\mathbf{(x_L y_R + x_R y_L)} = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$$

Integer Multiplication

$$xy = x_L y_L 2^n + (x_L y_R + x_R y_L)2^{n/2} + x_L y_R 2^{n/2} + x_R y_R$$

$$(x_L y_R + x_R y_L) = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$$

Three recursive multiplications + additions + shifts:

1. $x_L y_L$
2. $x_R y_R$
3. $(x_L + x_R)(y_R + y_L)$

Integer Multiplication

Algorithm:

multiply(x, y, n)

xL,xR = split(x)

yL,yR = split(y)

a = **multiply**(xL, yL, n/2)

b = **multiply**(xR, yR, n/2)

c = **multiply**(xL+xR, yL+yR, n/2)

return shift(a,n) + b + shift(c-a-b,n/2)

Integer Multiplication

Analysis:

$$
\begin{aligned}
T(n) &= 3T(n/2) + O(n) \\
&= O(n^{\log 3}) \\
&= O(n^{1.58})
\end{aligned}
$$

Integer Multiplication

$$\mathbf{xy = x_L y_L 2^n + (x_L y_R + x_R y_L) 2^{n/2} + x_L y_R 2^{n/2} + x_R y_R}$$

$$\mathbf{(x_L y_R + x_R y_L)} = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$$

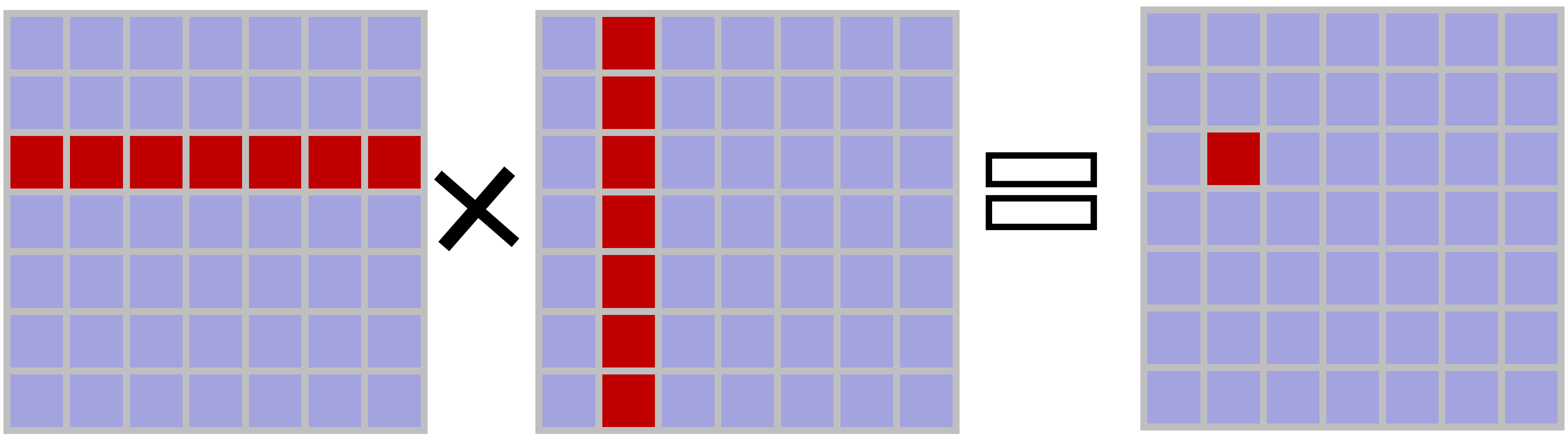
Three recursive multiplications + additions + shifts:

1. $\mathbf{x_L y_L}$
2. $\mathbf{x_R y_R}$
3. $\mathbf{(x_L + x_R)(y_R + y_L)}$

Matrix Multiplication

Given: two matrices A[n,n] and B[n,n]

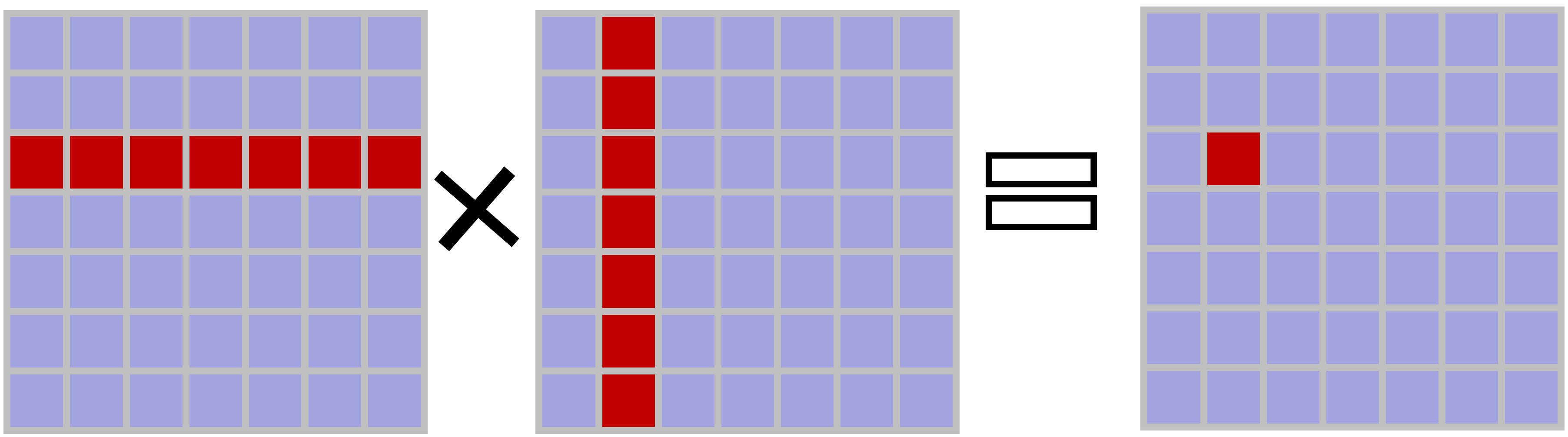
Calculate: matrix C = AB



Matrix Multiplication

Given: two matrices A[n,n] and B[n,n]

Calculate: matrix C = AB



$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

Matrix Multiplication

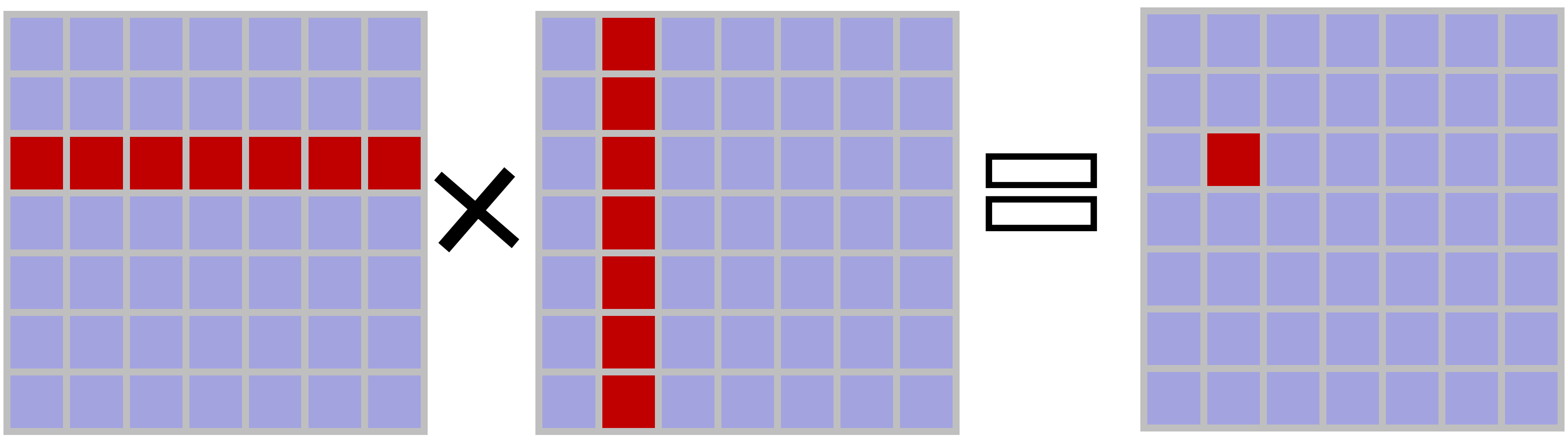
Multiply(A,B)

for $i = 1$ **to** n **do**

for $j = 1$ **to** n **do**

$C_{ij} = 0$

for $k = 1$ **to** n **do** $C_{ij} += A_{ik} * B_{kj}$



Matrix Multiplication

Multiply(A,B)

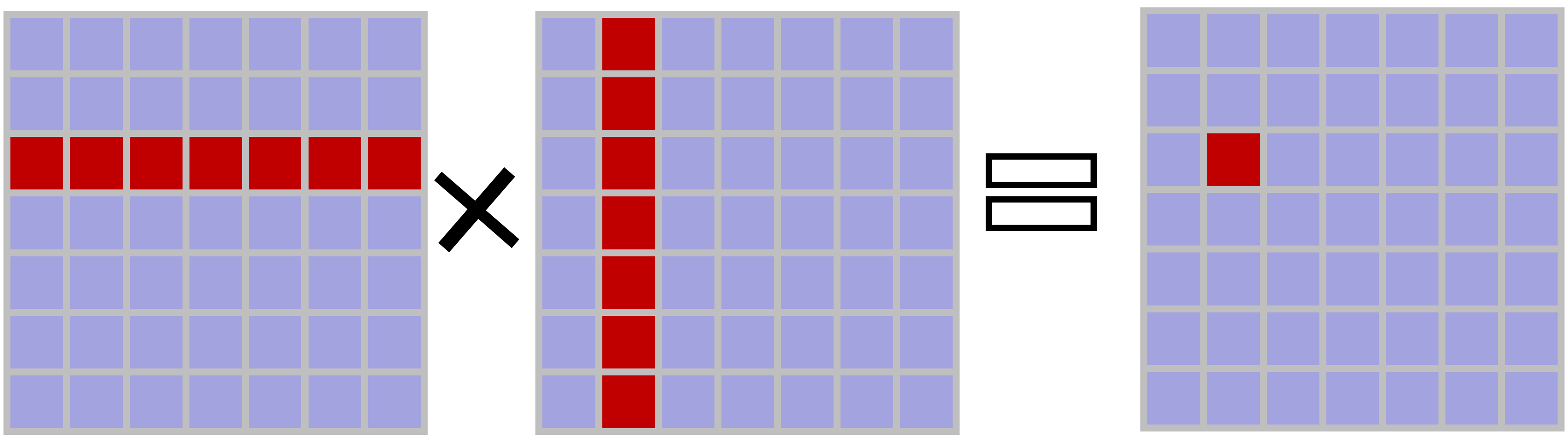
for $i = 1$ **to** n **do**

for $j = 1$ **to** n **do**

$C_{ij} = 0$

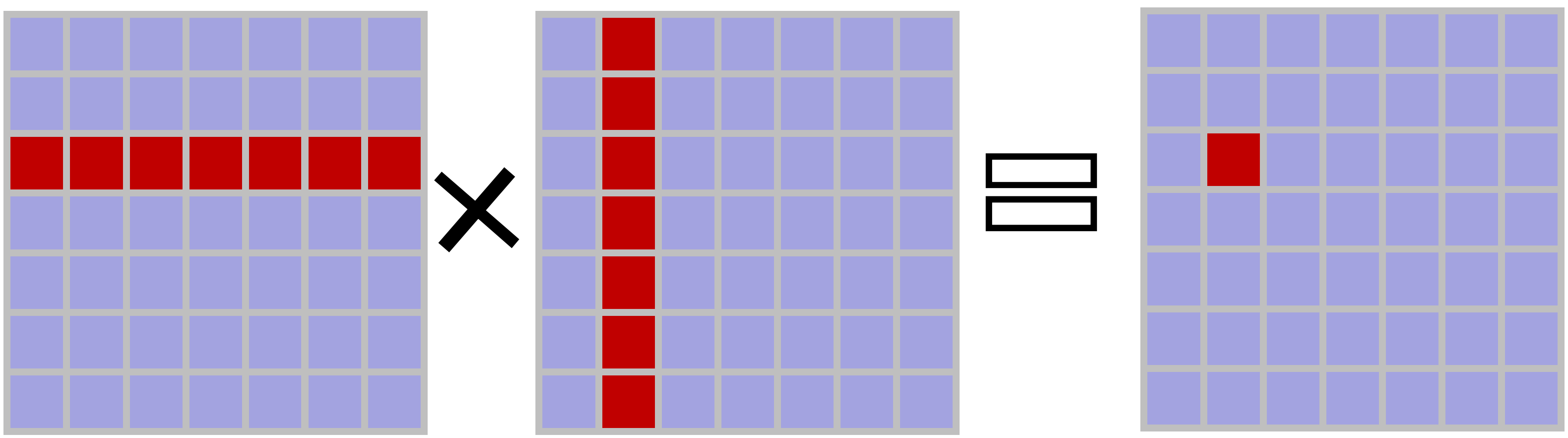
for $k = 1$ **to** n **do** $C_{ij} \mathrel{+}= A_{ik} * B_{kj}$

$O(n^3)$



Matrix Multiplication

Ideas for improvement?



Matrix Multiplication

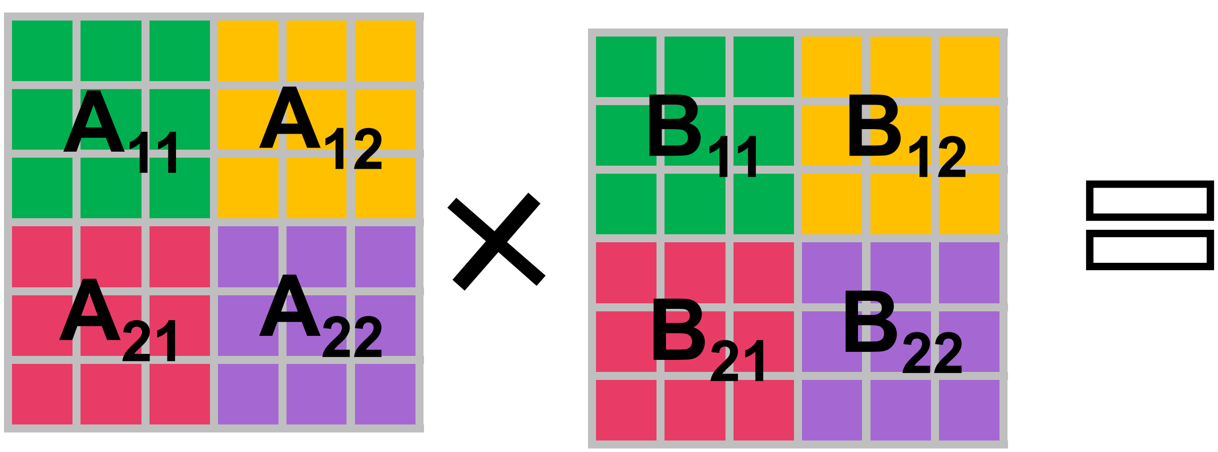
Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



Matrix Multiplication

Example: 6x6 matrix

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + \ldots + a_{26}b_{62}$$

$$C(1,1)_{22} = A(1,1)B(1,1)_{22} + A(1,2)_{12}B(2,1)_{22}$$

$$= A(1,1)_{21}B(1,1)_{12} + \ldots + A(1,1)_{23}B(1,1)_{32}$$

$$+ A(1,2)_{21}B(2,1)_{12} + \ldots + A(1,2)_{23}B(2,1)_{32}$$

$$= A_{21}B_{12} + \ldots + A_{23}B_{32}$$

$$+ A_{24}B_{42} + \ldots + A_{24}B_{62}$$

Matrix Multiplication

Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$T(n) = 8T(n/2) + O(n^2)$$

Substitution Method

Solve:

$$T(n) = 8T(n/2) + kn^2$$

Guess:

$$T(n) = n^3 - kn^2$$

Substitution Method

Solve:

$T(n) = 8T(n/2) + kn^2$

Guess:

$T(n) = n^3 - kn^2$

Test: $8T(n/2) + kn^2$

$$\begin{aligned} T(n/2) &= (n/2)^3 - k(n/2)^2 \\ &= n^3/8 - kn^2/4 \end{aligned}$$

Substitution Method

Solve:

$$T(n) = 8T(n/2) + kn^2$$

Guess:

$$T(n) = n^3 - kn^2$$

Test: $8T(n/2) + kn^2$

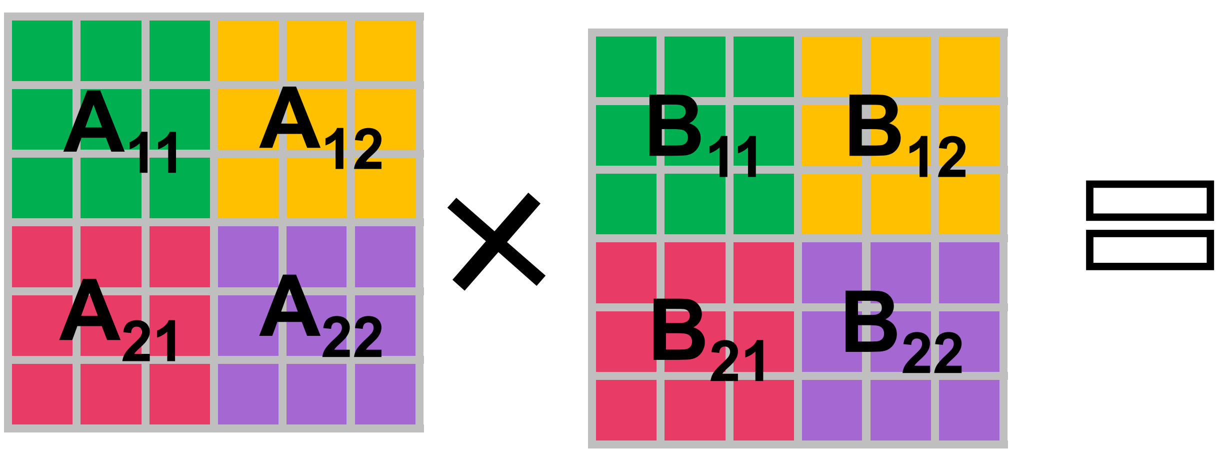
$$T(n/2) = (n/2)^3 - k(n/2)^2$$
$$= n^3/8 - kn^2/4$$

$$8T(n/2)+kn^2 = 8(n^3/8 - kn^2/4)+kn^2$$
$$= n^3 - 2kn^2+kn^2 = T(n)$$

Matrix Multiplication

Divide-and-Conquer

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



Matrix Magic

Define:

$M_1 = (A_{11}+A_{22})(B_{11} + B_{22})$

$M_2 = (A_{21}+ A_{22})B_{11}$

$M_3 = A_{11}(B_{12} - B_{22})$

$M_4 = A_{22}(B_{21} - B_{11})$

$M_5 = (A_{11}+A_{12})B_{22}$

$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$

$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

Notice: **7** multiplications!!

Matrix Magic

Calculate:

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Really!!

Magic!!

Matrix Multiplication

Strassen's Method:

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

About 18 matrix additions/subtractions

Not 8!

Matrix Multiplication

Strassen's Method:

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

$T(n) \cong n^{\log(7)} \cong n^{2.81}$

(Faster when N > 32, approximately)

Best known to date:

$T(n) \cong O(n^{2.376})$

(Theoretical use only.)

Most important algorithm?

Most important **divide-and-conquer** algorithm

Fast Fourier Transform

Signal processing (DSP)

- Linear filtering
- Correlation analysis
- Spectrum analysis