

**15-252**  
**More Great Ideas in**  
**Theoretical Computer Science**  
Lecture 3:  
Power of Algorithms



*September 15th, 2017*

# Poll

```
def isPrime(n):  
    if (n < 2):  
        return False  
    for factor in range(2,n):  
        if (n % factor == 0):  
            return False  
    return True
```

What is the running time as a function of input length?

- logarithmic
- linear
- log-linear
- quadratic
- exponential
- beats me

# Poll Answer

```
def isPrime(n):  
    if (n < 2):  
        return False  
    for factor in range(2,n):  
        if (n % factor == 0):  
            return False  
    return True
```

# iterations:  $\approx n$

$$n = 2^{\log_2 n} = 2^{\text{len}(n)}$$



exponential in  
input length

# Algorithms with number inputs

Algorithms on numbers involve **BIG** numbers.

3618502788666131106986593281521497110455743021169260358536775932020762686101  
7237846234873269807102970128874356021481964232857782295671675021393065473695  
3943653222082116941587830769649826310589717739181525033220266350650989268038  
3194839273881505432422077179121838888281996148408052302196889866637200606252  
6501310964926475205090003984176122058711164567946559044971683604424076996342  
7183046544798021168297013490774140090476348290671822743961203698142307099664  
3455133414637616824423860107889741058131271306226214208636008224651510961018  
9789006815067664901594246966730927620844732714004599013904409378141724958467  
7228950143608277369974692883195684314361862929679227167524851316077587207648  
7845058367231603173079817471417519051357029671991152963580412838184841733782

This is actually still small. Imagine having millions of digits.

# Algorithms with number inputs

$B = 5693030020523999993479642904621911725098567020556258102766251487234031094429$

$B \approx 5.7 \times 10^{75}$  ( 5.7 quattorvigintillion )

**Definition:**  $\text{len}(B) = \# \text{ bits to write } B$

$$n \approx \log_2 B$$

For  $B = 5693030020523999993479642904621911725098567020556258102766251487234031094429$

$$\text{len}(B) = 251$$

# Integer Addition

```
def sum(A, B):  
    for i from 1 to B do:  
        A += 1  
    return A
```

What is the running-time of this algorithm?



# Integer Addition

$$\begin{array}{r} 36185027886661311069865932815214971104 \quad A \\ + 65743021169260358536775932020762686101 \quad B \\ \hline 101928049055921669606641864835977657205 \quad C \end{array}$$

**# steps** to produce  $C$  is  $O(n)$

# Integer Multiplication

3618502788666 | 31 | 10698659328 | 52 | 497 | 104 *A*

*x* 5932020762686101 *B*

---

XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX  
XX

---

21465033672205046394665 | 358202698404452609868 | 37425504 *C*

**# steps:**  $O(\text{len}(A) \cdot \text{len}(B)) = O(n^2)$



# Integer Multiplication

You might think:

Probably this is the best, what else can you really do ?

A good algorithm designer always thinks:

**How can we do better ?**

What algorithm does Python use?

# Integer Multiplication

$$\begin{array}{r} \phantom{x = } \quad a \quad b \\ x = \quad \boxed{5\ 6} \boxed{7\ 8} \\ y = \quad \boxed{1\ 2} \boxed{3\ 4} \\ \phantom{y = } \quad c \quad d \end{array}$$

$$x = a \cdot 10^{n/2} + b$$

$$y = c \cdot 10^{n/2} + d$$

$$\begin{aligned} x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \end{aligned}$$

Use recursion!

# Integer Multiplication

$$\begin{array}{cc} & a & b \\ x = & \boxed{56} & \boxed{78} \\ y = & \boxed{12} & \boxed{34} \\ & c & d \end{array}$$

$$x = a \cdot 10^{n/2} + b$$

$$y = c \cdot 10^{n/2} + d$$

$$\begin{aligned} x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \end{aligned}$$

- Recursively compute  $ac$ ,  $ad$ ,  $bc$ , and  $bd$ .
- Do the multiplications by  $10^n$  and  $10^{n/2}$   $O(n)$
- Do the additions.  $O(n)$

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

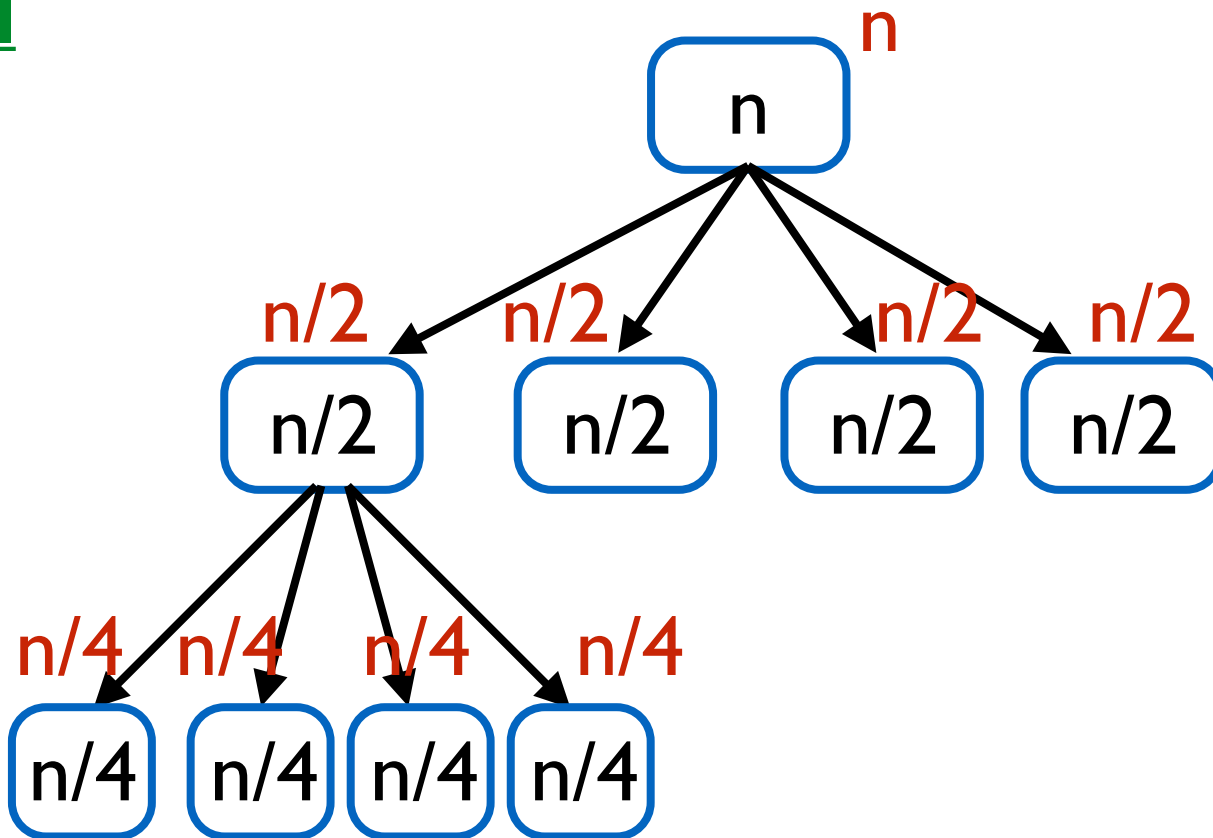
# Integer Multiplication

Level

0

1

2



# distinct nodes at level  $j$ :  $4^j$

work done per node at level  $j$ :  $c(n/2^j)$

# levels:  $\log_2 n$

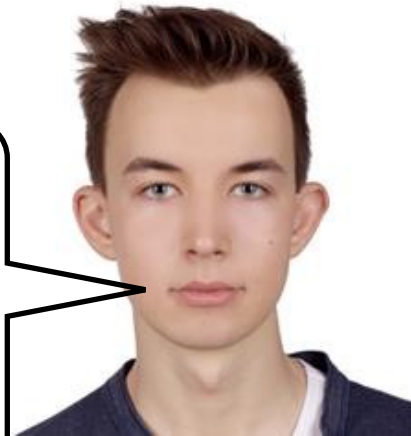
**Total cost:**  $\sum_{j=0}^{\log_2 n} cn2^j \in O(n^2)$

$cn2^j$   
per level

# Integer Multiplication

$$\begin{aligned}x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd\end{aligned}$$

Hmm, we don't really care about  $ad$  and  $bc$ .  
We just care about their sum.  
Maybe we can get away with 3 recursive calls.



# Integer Multiplication

$$\begin{aligned}x \cdot y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd\end{aligned}$$

$$(a + b)(c + d) = ac + \boxed{ad + bc} + bd$$

- Recursively compute  $ac$ ,  $bd$ ,  $(a+b)(c+d)$ .
- Then  $ad + bc = (a+b)(c+d) - ac - bd$ .

$$T(n) \leq 3T(n/2) + O(n) \quad \text{Is this better??}$$

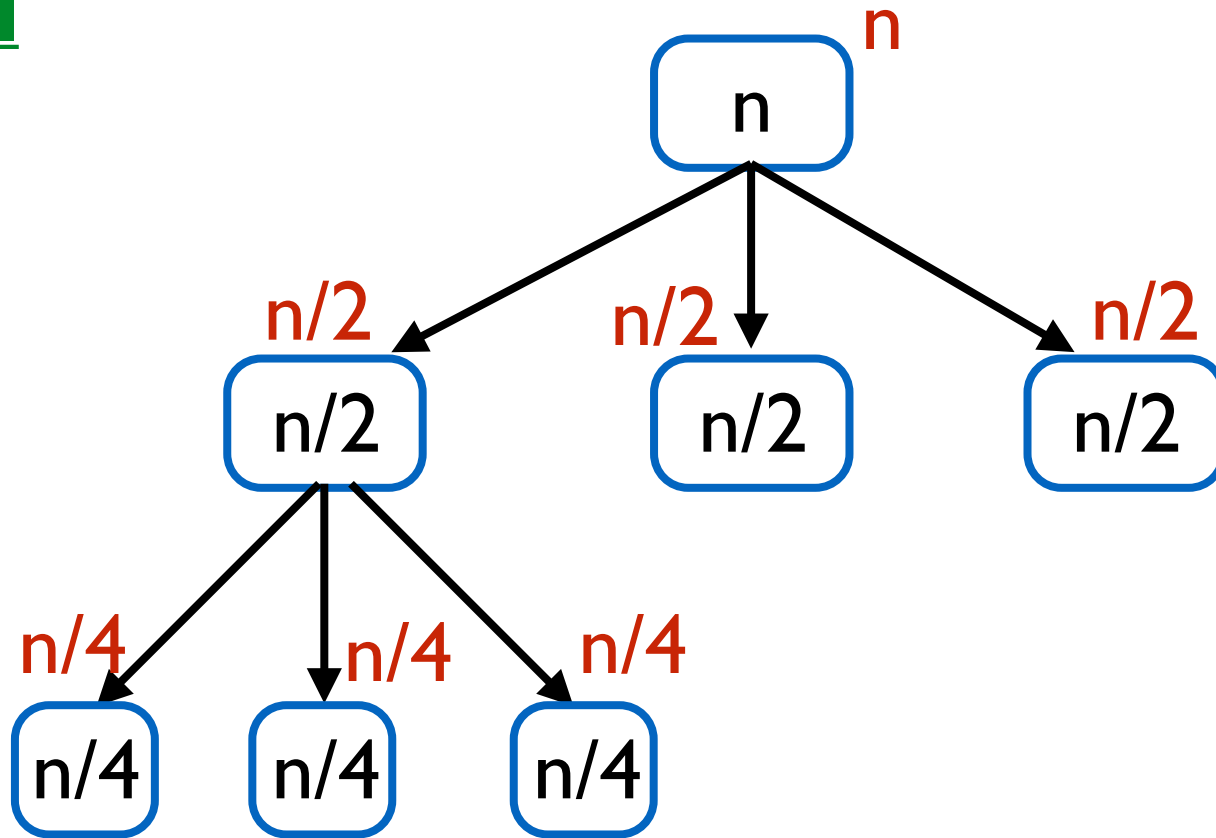
# Integer Multiplication

Level

0

1

2



# distinct nodes at level  $j$ :  $3^j$

work done per node at level  $j$ :  $c(n/2^j)$

# levels:  $\log_2 n$

**Total cost:**

$$\sum_{j=0}^{\log_2 n} cn(3^j / 2^j)$$

$cn(3^j / 2^j)$   
per level

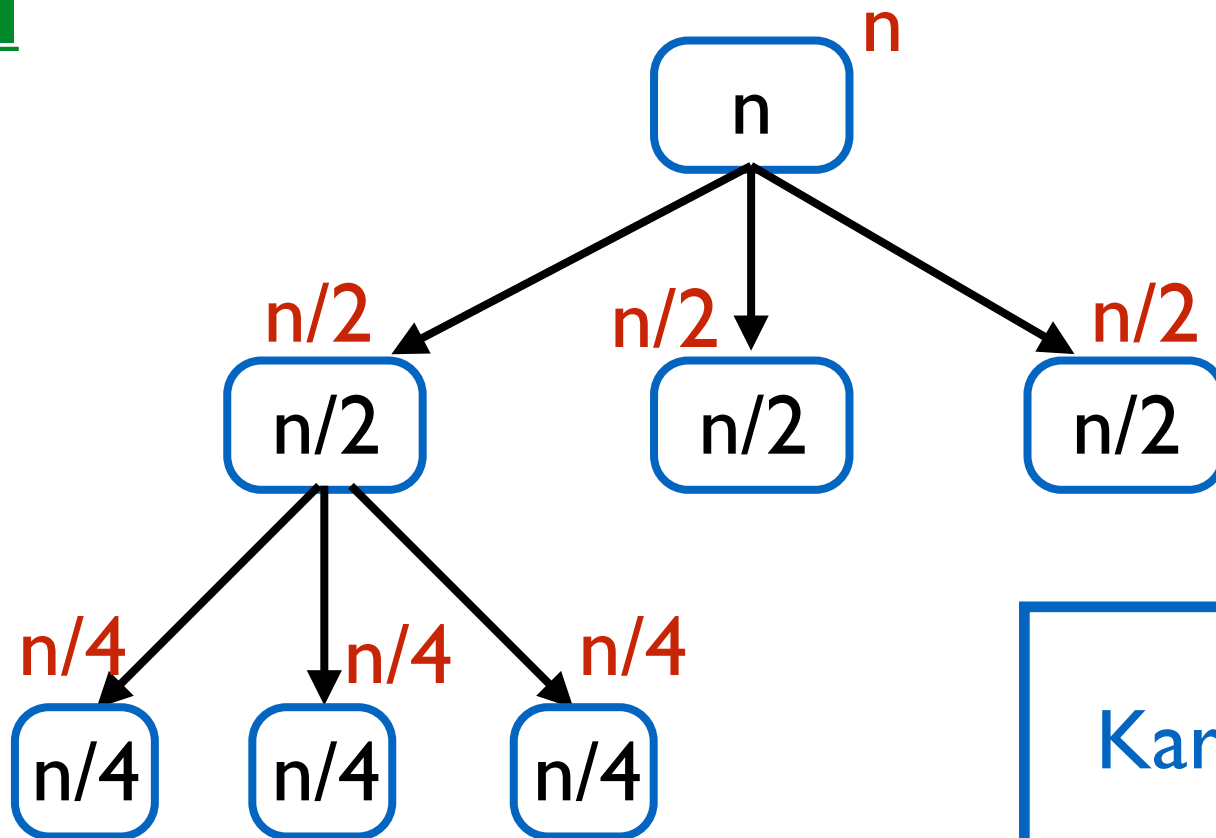
# Integer Multiplication

Level

0

1

2



Karatsuba Algorithm

**Total cost:** 
$$\sum_{j=0}^{\log_2 n} cn(3^j / 2^j) \leq Cn(3^{\log_2 n} / 2^{\log_2 n})$$
$$= C3^{\log_2 n}$$
$$= Cn^{\log_2 3} \in O(n^{\log_2 3})$$



# Integer Multiplication

You might think:

Probably this is the best, what else can you really do ?

A good algorithm designer always thinks:

**How can we do better ?**

Cut the integer into 3 parts of length  $n/3$  each.

Replace 9 multiplications with only 5.

$$T(n) \leq 5T(n/3) + O(n)$$

$$T(n) \in O(n^{\log_3 5})$$

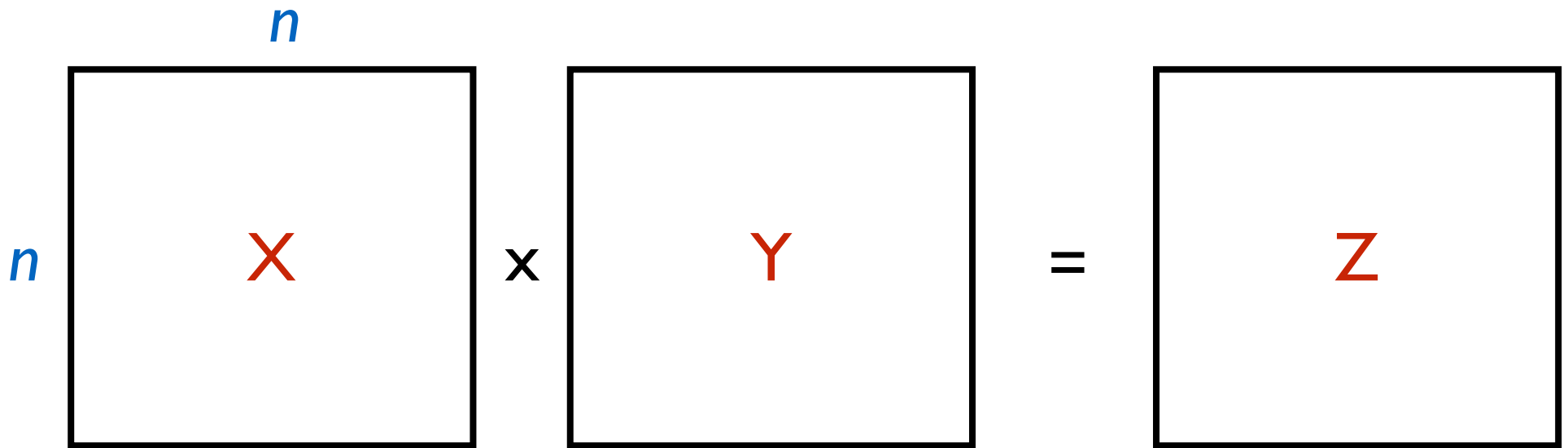
Can do  $T(n) \in O(n^{1+\epsilon})$  for any  $\epsilon > 0$ .

# Integer Multiplication

Fastest known:  $n(\log n)2^{O(\log^* n)}$

Martin Fürer  
(2007)

# Matrix Multiplication



**Input:** 2  $n \times n$  matrices  $X$  and  $Y$ .

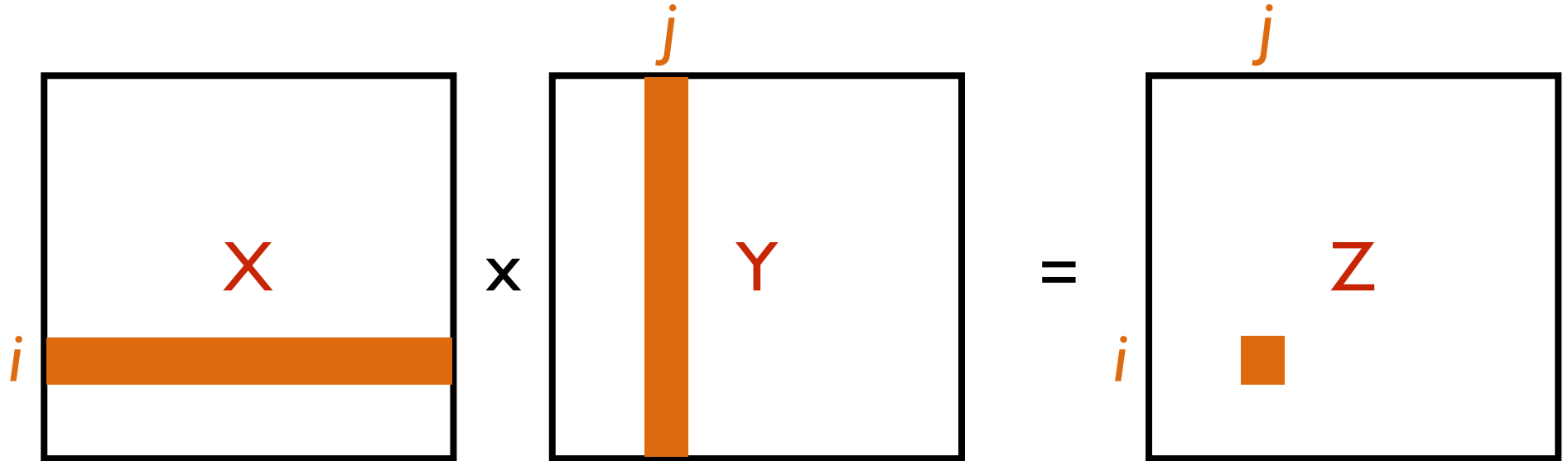
**Output:** The product of  $X$  and  $Y$ .

(Assume entries are objects we can multiply and add.)

# Matrix Multiplication

$$\begin{array}{|c|c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \times \begin{array}{|c|c|} \hline e & f \\ \hline g & h \\ \hline \end{array} = \begin{array}{|c|c|} \hline ae+bg & af+bh \\ \hline ce+dg & cf+dh \\ \hline \end{array}$$

# Matrix Multiplication



$Z[i,j] = (i\text{'th row of } X) \cdot (j\text{'th column of } Y)$

$$= \sum_{k=1}^n X[i,k] Y[k,j]$$

Algorithm 1:  $\Theta(n^3)$

# Matrix Multiplication

$$X = \begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline \end{array}$$

$$Y = \begin{array}{|c|c|} \hline E & F \\ \hline G & H \\ \hline \end{array}$$

$$Z = \begin{array}{|c|c|} \hline AE+BG & AF+BH \\ \hline CE+DG & CF+DH \\ \hline \end{array}$$

**Algorithm 2:** recursively compute 8 products  
+ do the additions.

$$\Theta(n^3)$$

# Matrix Multiplication: Strassen's Algorithm

$$Z = \begin{array}{|c|c|} \hline AE+BG & AF+BH \\ \hline \cdots & \cdots \\ \hline CE+DG & CF+DH \\ \hline \end{array}$$

Can reduce the number of products to 7.

$$Q1 = (A+D)(E+G)$$

$$Q2 = (C+D)E$$

$$Q3 = A(F-H)$$

$$Q4 = D(G-E)$$

$$Q5 = (A+B)H$$

$$Q6 = (C-A)(E+F)$$

$$Q7 = (B-D)(G+H)$$

$$AE+BG = Q1+Q4-Q5+Q7$$

$$AF+BH = Q3+Q5$$

$$CE+DG = Q2+Q4$$

$$CF+DH = Q1+Q3-Q2+Q6$$

# Matrix Multiplication: Strassen's Algorithm

**Running Time:**  $T(n) = 7 \cdot T(n/2) + O(n^2)$

$$\begin{aligned} \implies T(n) &= O(n^{\log_2 7}) \\ &= O(n^{2.81}) \end{aligned}$$





# Matrix Multiplication: Strassen's Algorithm



Volker Strassen

Strassen's Algorithm (1969)

Together with Schönhage (in 1971)  
did  $n$ -bit integer multiplication  
in time  $O(n \log n \log \log n)$



Arnold Schönhage

# The race for the world record

## Improvements since 1969

1978:  $O(n^{2.796})$  by Pan

1979:  $O(n^{2.78})$  by Bini, Capovani, Romani, Lotti

1981:  $O(n^{2.522})$  by Schönhage

1981:  $O(n^{2.517})$  by Romani

1981:  $O(n^{2.496})$  by Coppersmith, Winograd

1986:  $O(n^{2.479})$  by Strassen

1990:  $O(n^{2.376})$  by Coppersmith, Winograd

No improvement for 20 years!

# The race for the world record

No improvement for 20 years!

2010:  $O(n^{2.374})$  by Andrew Stothers (PhD thesis)



2011:  $O(n^{2.373})$  by Virginia Vassilevska Williams



(CMU PhD, 2008)

# The race for the world record

2011:  $O(n^{2.373})$  by Virginia Vassilevska Williams



(CMU PhD, 2008)

Current world record:

2014:  $O(n^{2.372})$  by François Le Gall

# Enormous Open Problem

Is there an  $O(n^2)$  time algorithm  
for matrix multiplication ???