15-251
Great Theoretical Ideas in Computer Science

Lecture 10:
Power of Algorithms

February 16th, 2017

Computable cousins of uncomputable problems

Halting Problem

Input: Description of a TM **M** and an input **x**

Question: Does **M**(**x**) halt?

This is undecidable.

Halting Problem with Time Bound

Input: Description of a TM **M**, an input **x**, a number **k**

Question: Does **M**(**x**) halt in at most **k** steps?

This is decidable.   (Simulate for k steps)

Computable cousins of uncomputable problems

Theorem Proving Problem

Input: A FOL **statement** (a mathematical statement)

Question: Is the **statement** provable?

This is undecidable.

Theorem Proving Problem with a Bound

Input: A FOL **statement** (a mathematical statement), k

Question: Is the **statement** provable
using at most k symbols?

This is decidable. (Brute-force search)

Kurt Friedrich Gödel (1906-1978)

Logician, mathematician, philosopher.

Considered to be one of the most important logicians in history.

Incompleteness Theorems.

Completeness Theorem.

John von Neumann (1903-1957)

Contents [hide]

- Mathematical formulation of quantum mechanics

- Founded the field of game theory in mathematics.

- Created some of the first general-purpose computers.

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Gödel's letter to von Neumann

Theorem Proving Problem with a Bound

Input: A FOL **statement** (a mathematical statement), **k**

Question: Is the **statement** provable
using at most **k** symbols?

This is decidable. (Brute-force search)

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Gödel's letter to von Neumann

$\Psi(F, n)$ = the number of steps required for input (*F*, *n*)

$$\varphi(n) = \max_F \Psi(F, n)$$

(a worst-case notion of running time)

Question: How fast does $\varphi(n)$ grow for an optimal machine?

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Gödel's letter to von Neumann (1956)

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n (length = number of symbols). Let $\psi(F,n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F,n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem. Now it seems to me, however, to be completely within the realm of possibility that $\varphi(n)$ grows that slowly.

Goals for the week

1. **What is the right way to study complexity?**

   - using the right language and level of abstraction

   - upper bounds vs lower bounds

   - polynomial time vs exponential time

2. **Appreciating the power of algorithms.**

   - analyzing some cool (recursive) algorithms

Algorithms with integer inputs

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

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

iterations: ≈ 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: $\text{len}(B) = \#$ bits to write B

$n \approx \log_2 B$

For $B = 5693030020523999993479642904621911725098567020556258102766251487234031094429$

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

Algorithms with number inputs

$B =$ 5693030020523999993479642904621911725098567020556258102766251487234031094429

<u>Goal</u>: find one (non-trivial) factor of B

> **for** A = 2, 3, 4, 5, …
>   test **if** B mod A = 0.

It turns out:

$B =$ 68452332409801603635385895997250919383 x
83167801886452917478124266362673045163

Each factor ≈ age of the universe in Planck time.

Worst case: $\sqrt{B}$ iterations.

$$\sqrt{B} = \sqrt{2^{\log_2 B}} = \sqrt{2^{\text{len}(B)}} = 2^{\text{len}(B)/2}$$

exponential in input length

Recall our model

The Random-Access Machine (RAM) model

Good combination of reality/simplicity.

+ , - , / , *, <, >, etc.	e.g. 245*12894	takes 1 step
memory access	e.g. A[94]	takes 1 step

Actually:

We'll assume arithmetic operations take 1 step
if the numbers are bounded by a polynomial in **n**.

Unless specified otherwise, we will use this model.

Example

```
def double(B):
    return B+B
```

arithmetic operation

Are the numbers involved bounded by poly(n)?

What is the running-time of this algorithm?

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

$$
\begin{array}{rrl}
 & 36185027886661311069865932815214971104 & A \\
+ & 65743021169260358536775932020762686101 & B \\
\hline
 & 101928049055921669606641864835977657205 & C
\end{array}
$$

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

Integer Multiplication

```
                    3618502788666131106986593281521497110 4   A
  x                                        5932020762686101   B
  -----------------------------------------------------------
                XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
               XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
             XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
           XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
         XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  -----------------------------------------------------------
 214650336722050463946651358202698404452609868137425504       C
```

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

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

	a	b
x =	5 6	7 8
y =	1 2	3 4
	c	d

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

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

$$x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$$

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

Use recursion!

Integer Multiplication

	a	b
x =	5 6	7 8
y =	1 2	3 4
	c	d

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

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

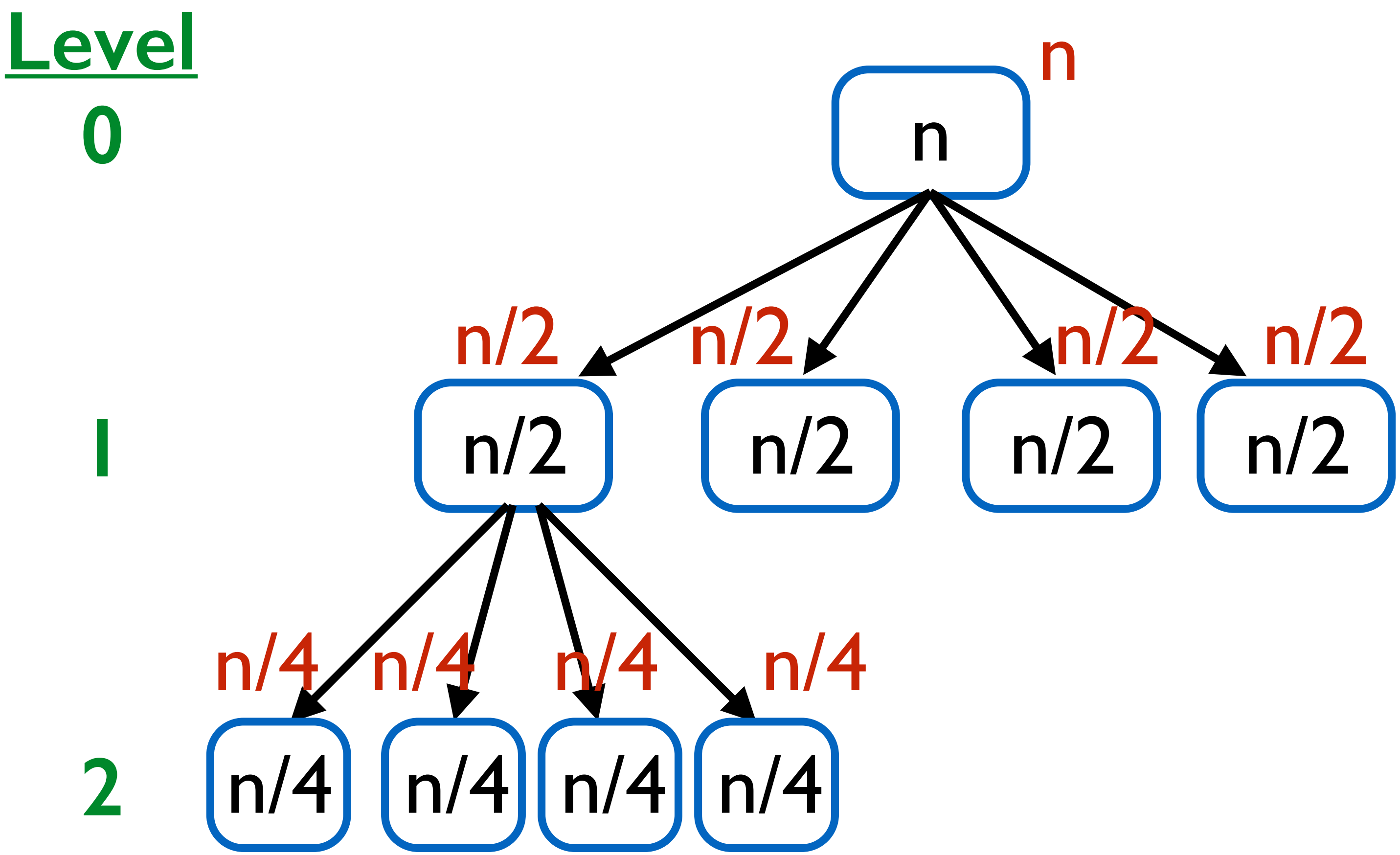
$$x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$$

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

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

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

Integer Multiplication



distinct nodes at level j: 4^j

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

levels: $\log_2 n$

$cn2^j$ per level

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

Integer Multiplication

$$x \cdot y = (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$$

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

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

$$x \cdot y = \quad (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d)$$

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

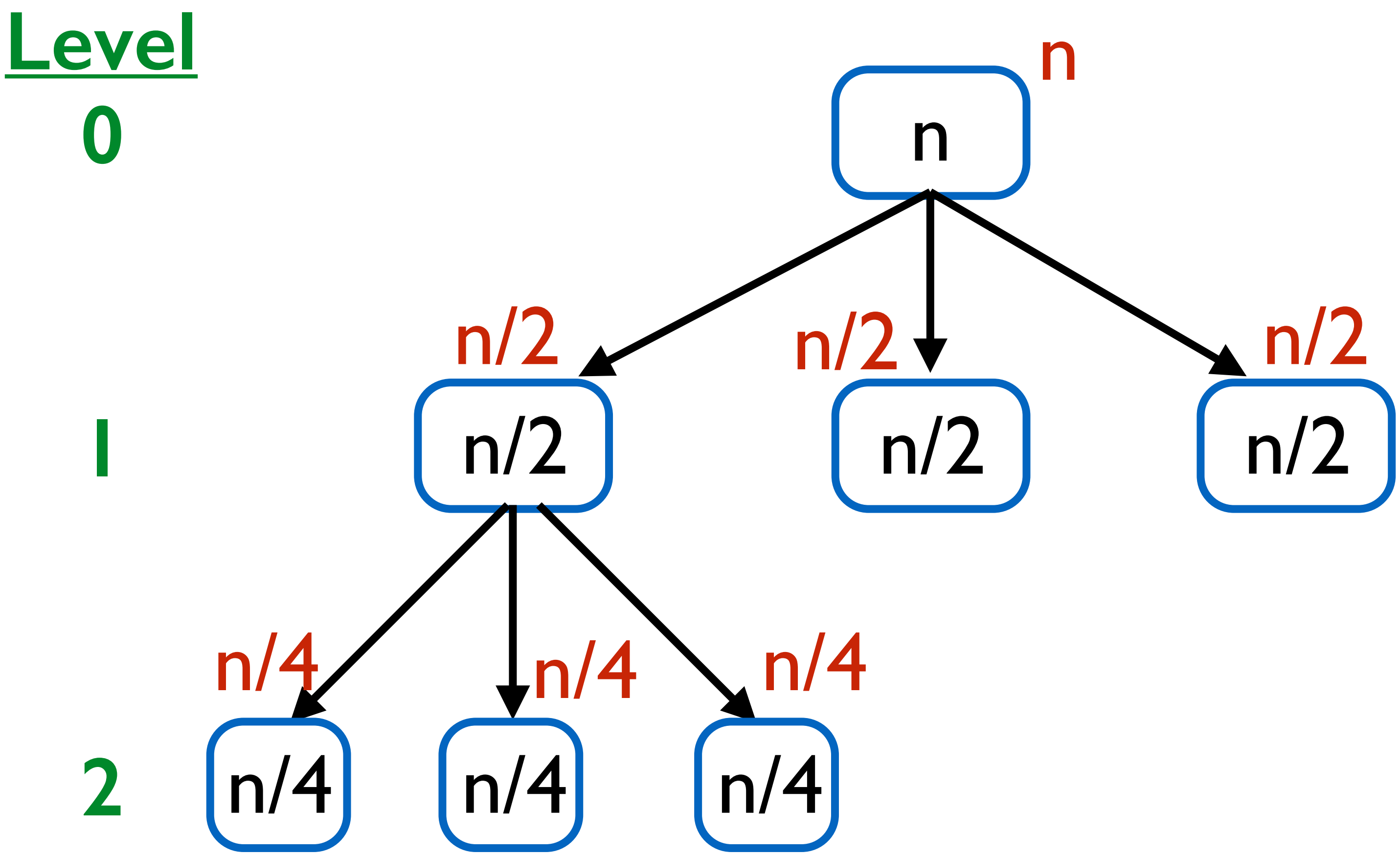
$$(a + b)(c + d) \quad = ac + \boxed{ad + bc} + bd$$

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

$$T(n) \leq \quad 3T(n/2) + O(n)$$

Is this better??

Integer Multiplication



distinct nodes at level j: 3^j

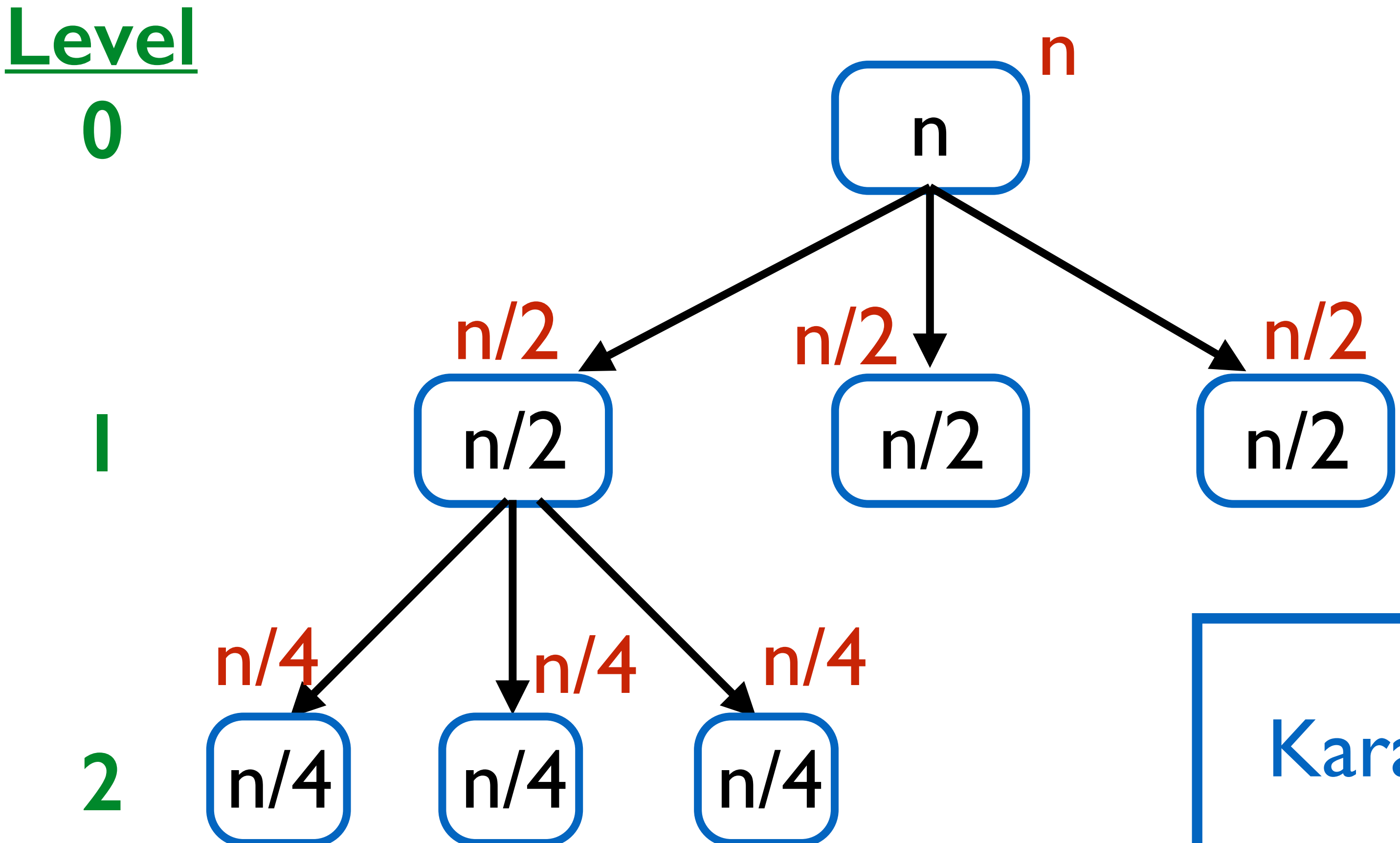
work done per node at level j: $c(n/2^j)$

levels: $\log_2 n$

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

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

Integer Multiplication



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.

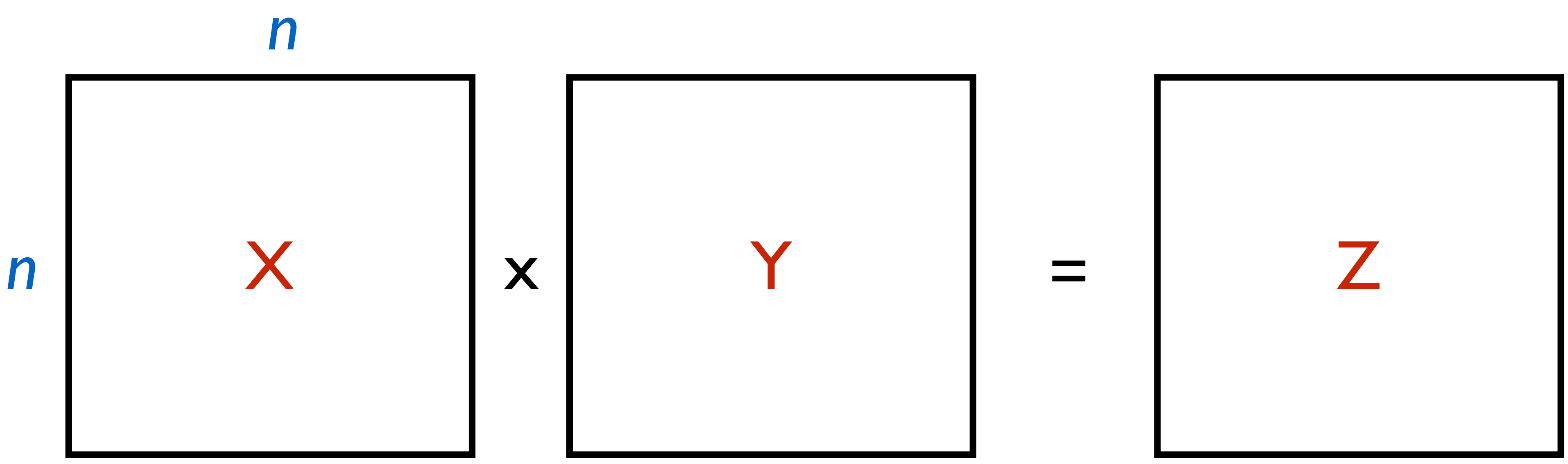
$$T(n) \leq 5T(n/3) + O(n)$$
$$T(n) \in O(n^{\log_3 5})$$

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 x n matrices X and Y.

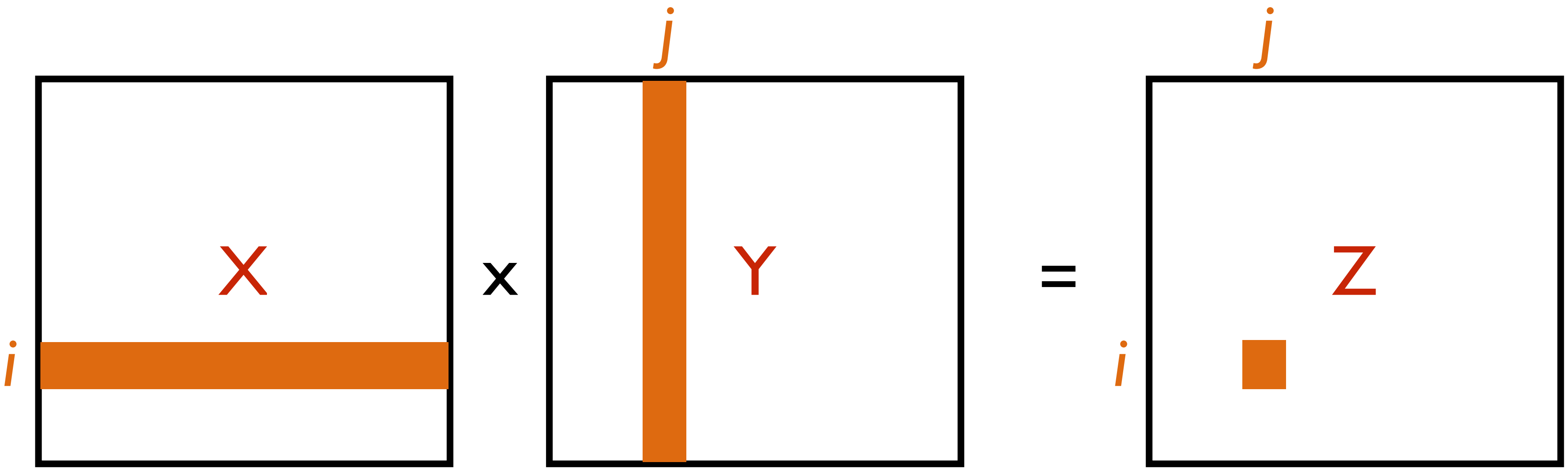
Output: The product of X and Y.

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

Matrix Multiplication

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

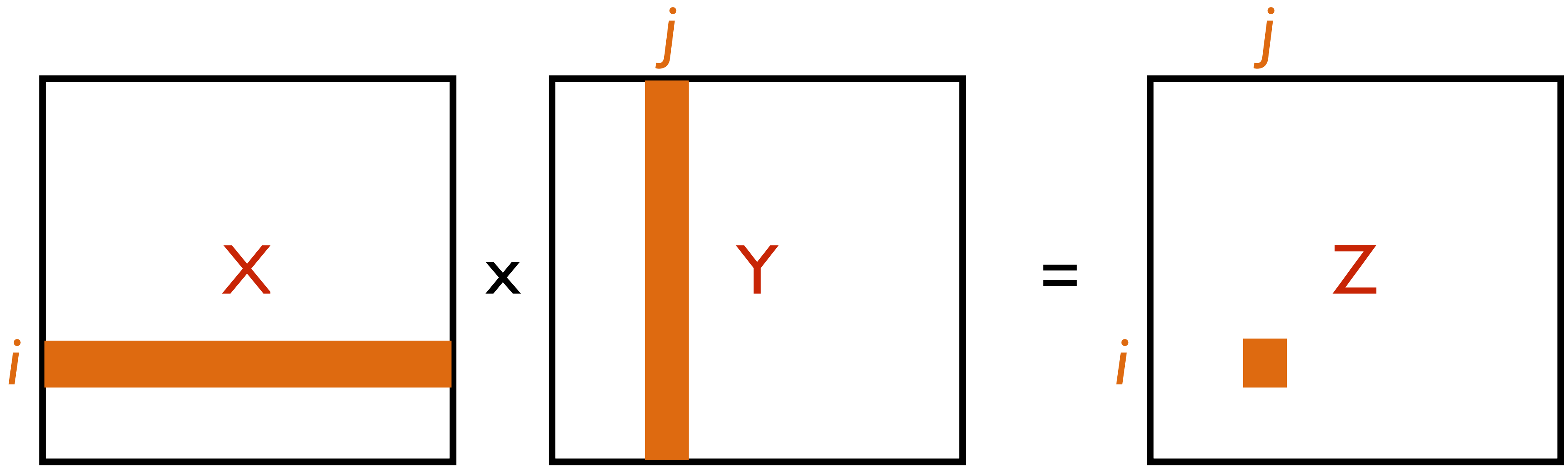
Matrix Multiplication



Z[i,j] = (i'th row of X) · (j'th column of Y)

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

Matrix Multiplication



Z[i,j] = (i'th row of X) · (j'th column of Y)

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

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

Matrix Multiplication

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$Z = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

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

$$\Theta(n^3)$$

Matrix Multiplication: Strassen's Algorithm

$$Z = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

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

$$\implies \quad T(n) = O(n^{\log_2 7})$$

$$= O(n^{2.81})$$

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 ???