15-251
Great Theoretical Ideas in Computer Science

Lecture 8:
Power of Algorithms

September 22nd, 2016

2 main questions in TOC

Computability of a problem:

Is there an algorithm to solve it?

Complexity of a problem:

Is there an efficient algorithm to solve it?

- time
- space (memory)
- randomness
- quantum resources

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.

Great contributions to foundations of mathematics.

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 running time of recursive functions

Polynomial time vs Exponential time

What is efficient in theory and in practice ?

In practice:

$O(n)$	Awesome! Like really awesome!
$O(n \log n)$	Great!
$O(n^2)$	Kind of efficient.
$O(n^3)$	Barely efficient. (???)
$O(n^5)$	Would not call it efficient.
$O(n^{10})$	Definitely not efficient!
$O(n^{100})$	WTF?

What is efficient in theory and in practice ?

In theory:

Polynomial time	Efficient.
Otherwise	Not efficient.

- Poly-time is not meant to mean "efficient in practice"

- It means "You have done something extraordinarily better than brute force (exhaustive) search."

- Poly-time: mathematical insight into a problem's structure.

- If you show, say Factoring Problem, has running time $O(n^{100})$, it will be the best result in CS history.

What is efficient in theory and in practice ?

In theory:

Polynomial time	Efficient.
Otherwise	Not efficient.

- Robust to notion of what is an elementary step, what model we use, reasonable encoding of input, implementation details.

- Nice closure property: Plug in a poly-time alg. into another poly-time alg. —> poly-time

What is efficient in theory and in practice ?

In theory:

Polynomial time	Efficient.
Otherwise	Not efficient.

- Big exponents don't really arise.

- If it does arise, usually can be brought down.

What is efficient in theory and in practice ?

In theory:

Polynomial time	Efficient.
Otherwise	Not efficient.

- **Summary**: Poly-time vs not poly-time
  is a qualitative difference, not a quantitative one.

Can you cheat exponential time?

Algorithms with integer inputs

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

Technically:

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

Unless specified otherwise, we will use this model.

Integer Summation

Input: 2 n-digit numbers x and y.

Output: The sum of x and y.

Can we assume that this takes 1 step?

Are x and y bounded by some polynomial in n?

No! x and y can be about 10^n.

Imagine $n = 1$ billion (which is a *realistic* value for n).

Integer summation requires an algorithm!

Integer Summation

Input: 2 n-digit numbers x and y.

Output: The sum of x and y.

First attempt at an algorithm.

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

Remember, x can be about 10^n.

The time complexity of this algorithm is $\Omega(10^n)$.

Integer Summation

Input: 2 n-digit numbers x and y.

Output: The sum of x and y.

Second attempt at an algorithm.

```
def sum(x, y):
    carry = 0
    for i from 0 to n-1 do:
        columnSum = x[i] + y[i] + carry
        z[i] = columnSum % 10
        carry = (columnSum - z[i]) / 10
    z[n] = carry
    return z
```

all arithmetic operations here are on bounded ints

Time complexity of algorithm: $O(n)$

Intrinsic complexity of summation: $\Theta(n)$

Integer Multiplication

Input: 2 n-digit numbers x and y.

Output: The product of x and y.

Grade-School Algorithm:

$$
\begin{array}{rrrrrrrrl}
 & & & & 5 & 6 & 7 & 8 & \\
 & \times & & & 1 & 2 & 3 & 4 & \\
\hline
 & & & 2 & 2 & 7 & 1 & 2 & \longrightarrow \quad O(n) \text{ operations} \\
 & & 1 & 7 & 0 & 3 & 4 & & \longrightarrow \quad O(n) \text{ operations} \\
 & 1 & 1 & 3 & 5 & 6 & & & \longrightarrow \quad O(n) \text{ operations} \\
+ & 5 & 6 & 7 & 8 & & & & \longrightarrow \quad O(n) \text{ operations} \\
\hline
 & 7 & 0 & 0 & 6 & 6 & 5 & 2 &
\end{array}
$$

n rows

Total: $O(n^2)$

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 ?

Let's try a different approach and see what happens…

Integer Multiplication

$$
\begin{array}{cc}
 & \begin{array}{cc} \text{a} & \text{b} \end{array} \\
x = & \boxed{5\ 6}\boxed{7\ 8} \\
y = & \boxed{1\ 2}\boxed{3\ 4} \\
 & \begin{array}{cc} \text{c} & \text{d} \end{array}
\end{array}
$$

$$
\begin{aligned}
x &= a \cdot 10^{n/2} + b \\
y &= c \cdot 10^{n/2} + d
\end{aligned}
$$

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

Use recursion!

Integer Multiplication

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

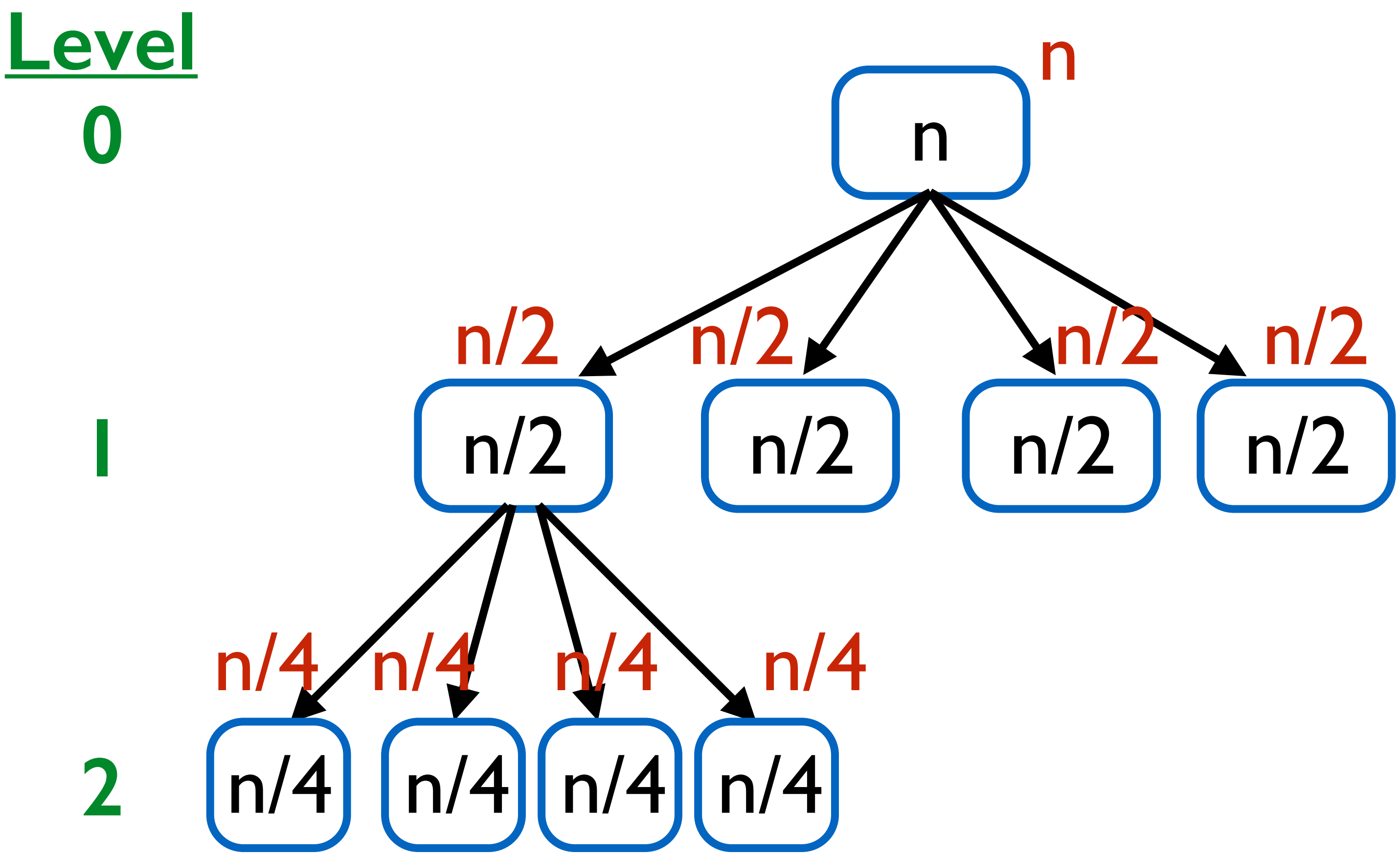
$$\begin{aligned} x &= a \cdot 10^{n/2} + b \\ y &= c \cdot 10^{n/2} + d \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned} x &= a \cdot 10^{n/2} + b \\ y &= c \cdot 10^{n/2} + d \end{aligned}$$

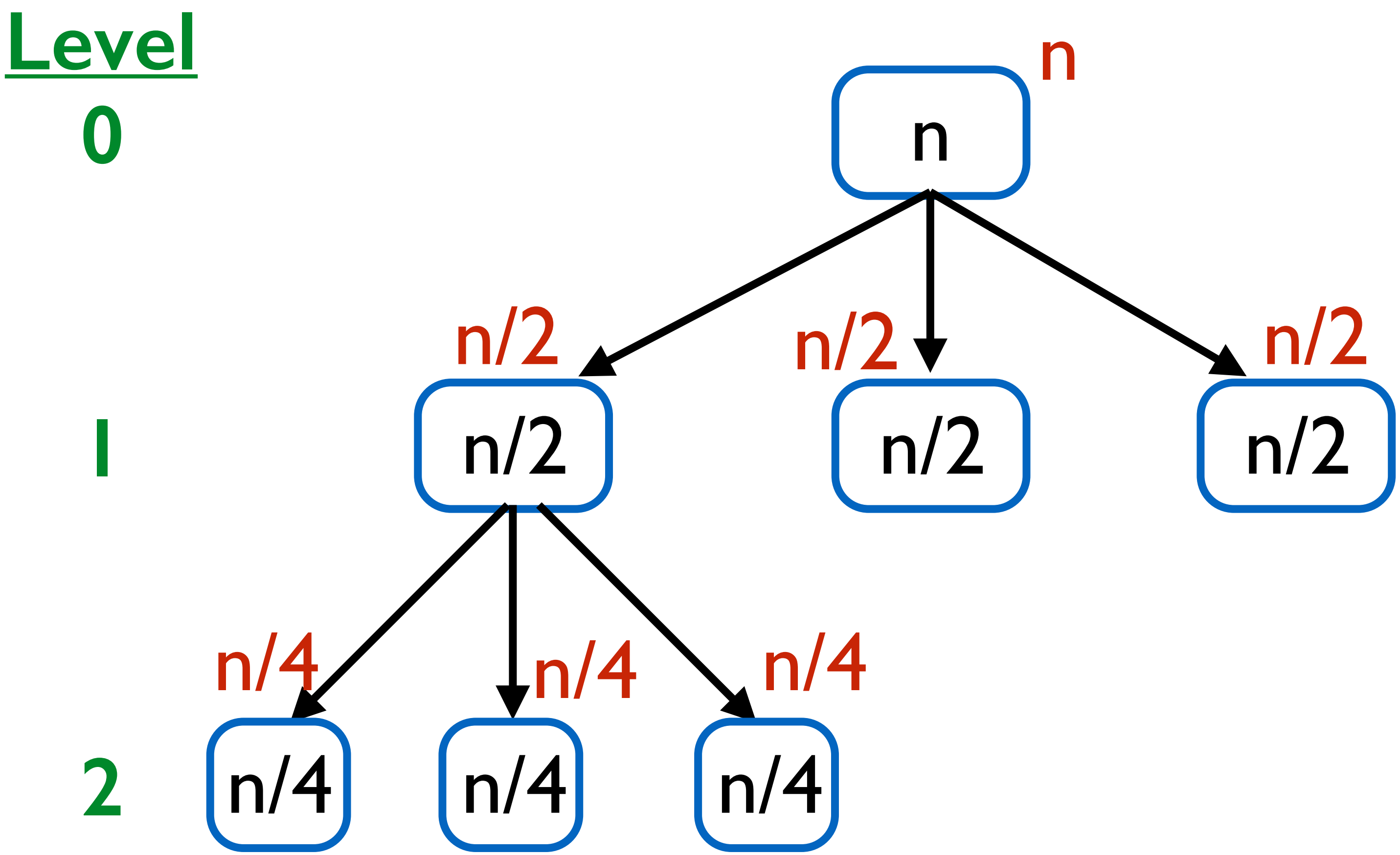
$$\begin{aligned} x \cdot y = \ & (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ & = ac \cdot 10^{n} + (ad + bc) \cdot 10^{n/2} + bd \end{aligned}$$

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

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

Is this better??

Integer Multiplication



distinct nodes at level j: 3^j

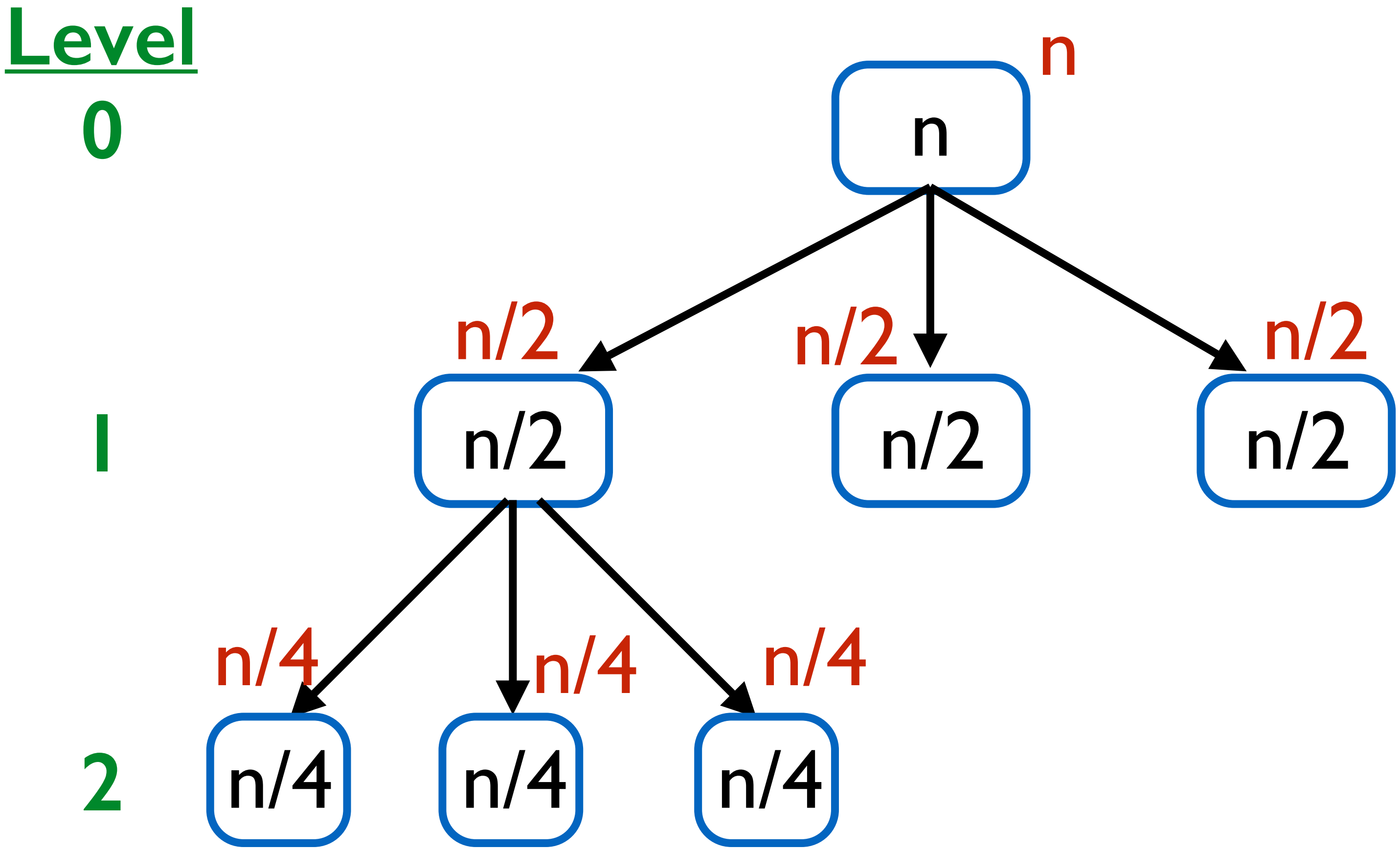
work done per node at level j: $c(n/2^j)$

levels: $\log_2 n$

$$\boxed{cn(3^j/2^j) \text{ per level}}$$

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

Integer Multiplication



Total cost:

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

$$= C3^{\log_2 n}$$

Karatsuba Algorithm

$$= Cn^{\log_2 3} \in O(n^{\log_2 3})$$

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.

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

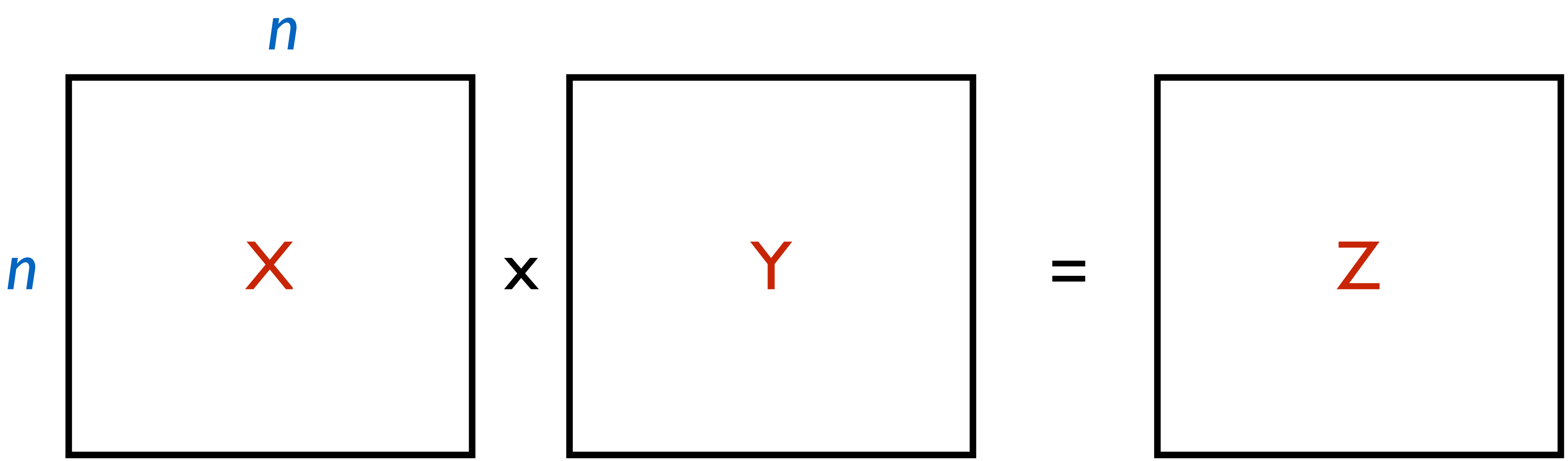
Can do $T(n) \in O(n^{1+\epsilon})$ for any $\epsilon > 0$.

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.

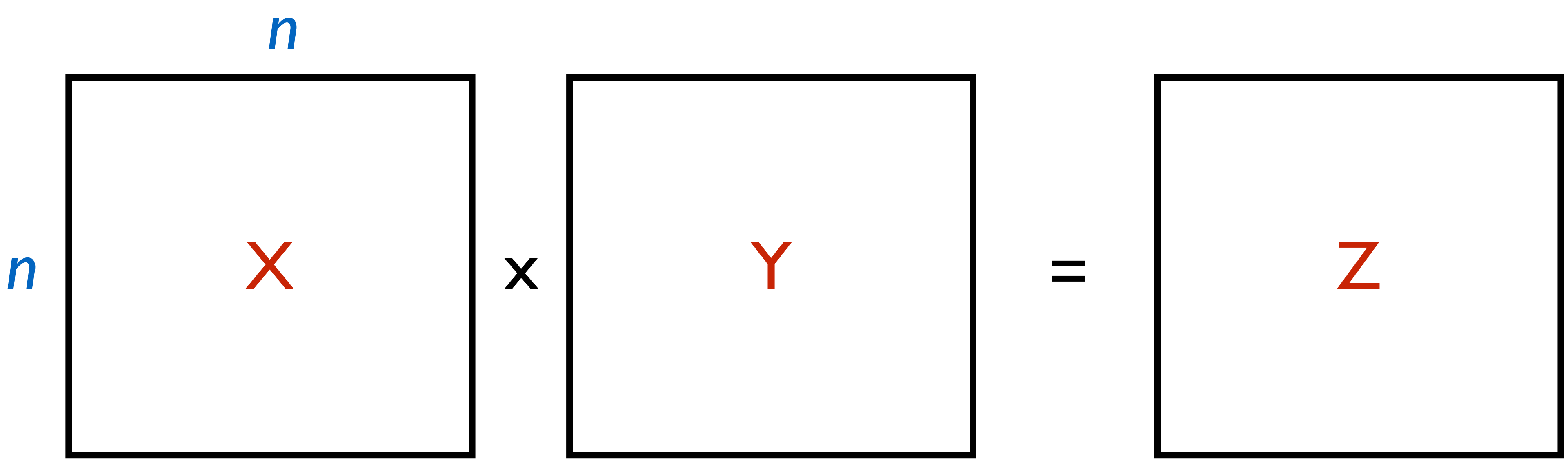
Output: The product of X and Y.

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

Matrix Multiplication

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

Matrix Multiplication



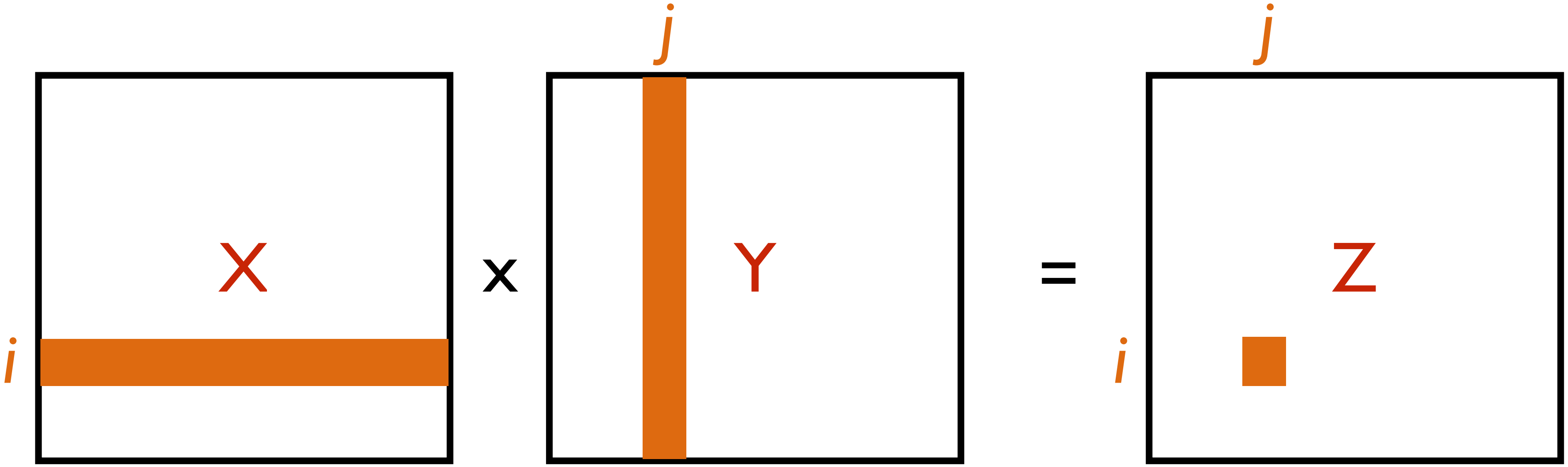
Input: 2 n x n matrices X and Y.

Output: The product of X and Y.

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

Note: we are interested in the number of multiplications needed to solve this problem.

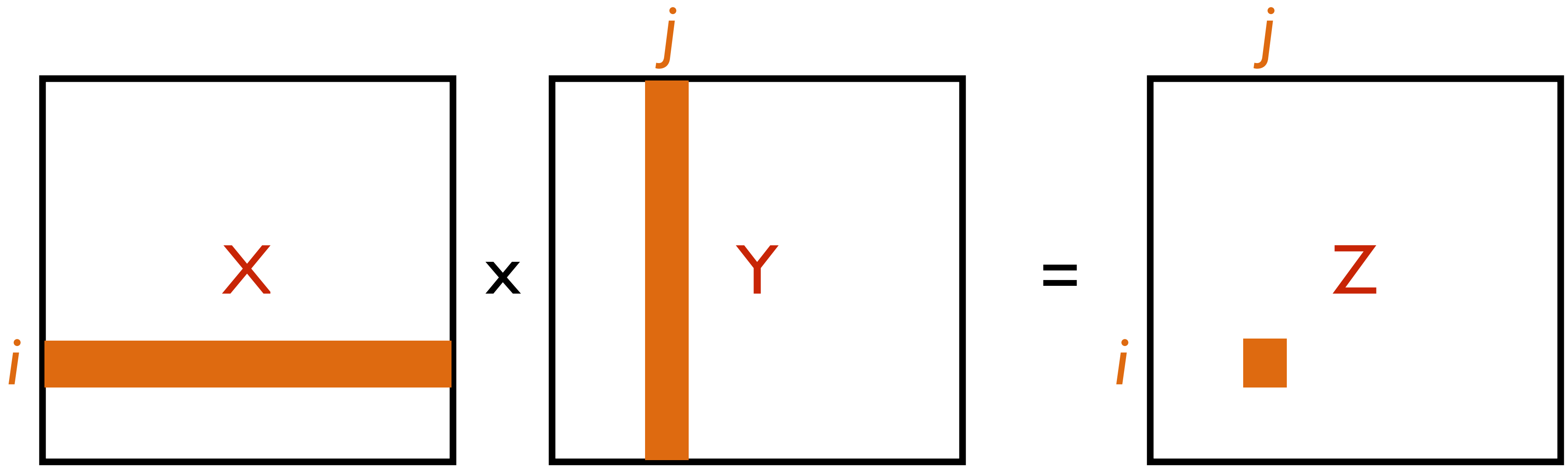
Matrix Multiplication



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



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

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

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)

Arnold Schönhage

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

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 ???