15-251

Great Theoretical Ideas in Computer Science

Introduction to Computational Complexity II

February 5th, 2015

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

A computational problem

Input: A FOL formula F, and m

Output: YES if there is a proof F of length m
NO otherwise

Clearly this is decidable.

Can do 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

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

$$\varphi(m) = \max_F \Psi(F, m)$$ (a worst-case notion of running time)

Question: How fast does $\varphi(m)$ 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

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

$$\varphi(m) = \max_F \Psi(F, m)$$ (a worst-case notion of running time)

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

He claims $\varphi(m) \geq k \cdot m$ (a lower bound)

If $\varphi(m) \sim k \cdot m$ or even $\varphi(m) \sim k \cdot m^2$

(if we could really beat Brute Force Search)

"this would have consequences of the greatest importance"

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.

Running time analysis:

Dealing with summations

Dealing with recursion

Dealing with summations

1. Rough bounding
2. Exact computation
3. Induction
4. Telescoping series
5. Comparison with an integral

Dealing with summations

1. Rough bounding

$$\begin{aligned}\sum_{i=1}^{n} i &= 1 + 2 + 3 + \cdots + n \\ &\leq n + n + n + \cdots + n \\ &= n^2\end{aligned}$$

$$\begin{aligned}\sum_{i=1}^{n} i &\geq \sum_{i=n/2}^{n} i \\ &\geq \frac{n}{2} + \frac{n}{2} + \cdots + \frac{n}{2} \\ &= \frac{n^2}{4}\end{aligned}$$

$\Theta(n^2)$

Dealing with summations

2. Exact computation

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n+1}{2}$$

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$$

If $|x| < 1$:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Dealing with summations

2. Exact computation

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1}$$

If $|x| < 1$:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x}$$

$$\sum_{i=0}^{\infty} i x^{i-1} = \frac{1}{(1 - x)^2} \qquad \sum_{i=0}^{\infty} i x^i = \frac{x}{(1 - x)^2}$$

Dealing with summations

3. Induction

$$\sum_{i=0}^{n} 3^i \quad \leq \quad C \cdot 3^n$$

Prove by induction on *n*.

Dealing with summations

4. Telescoping series

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \sum_{i=1}^{n} \left(\frac{1}{i} - \frac{1}{i+1} \right)$$

$$= \left(\frac{1}{1} - \frac{1}{2} \right) + \left(\frac{1}{2} - \frac{1}{3} \right) + \left(\frac{1}{3} - \frac{1}{4} \right) + \cdots + \left(\frac{1}{n} - \frac{1}{n+1} \right)$$

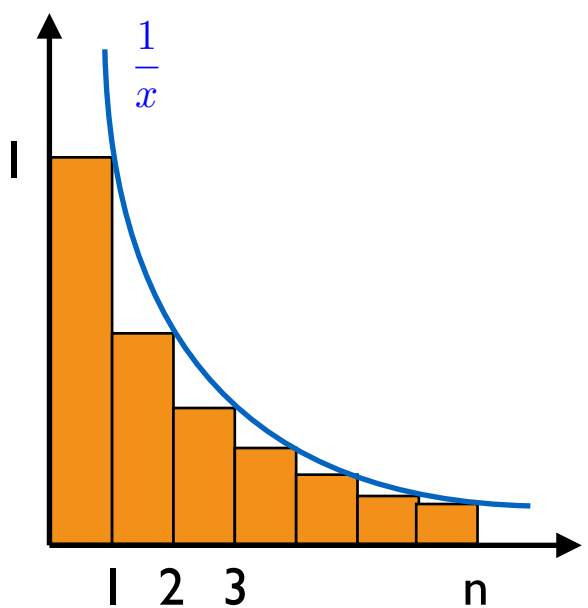
$$= 1 - \frac{1}{n+1}$$

Dealing with summations

5. Comparison with an integral

$$\sum_{i=1}^{n} f(i) \quad \approx \quad \int_{x=1}^{n} f(x)dx$$

$$\sum_{i=1}^{n} \frac{1}{i} \quad \approx \quad \int_{x=1}^{n} \frac{1}{x}dx \quad = \quad \ln(n)$$



$$\sum_{i=1}^{n} \frac{1}{i} \quad \leq \quad 1 + \int_{x=1}^{n} \frac{1}{x}dx$$

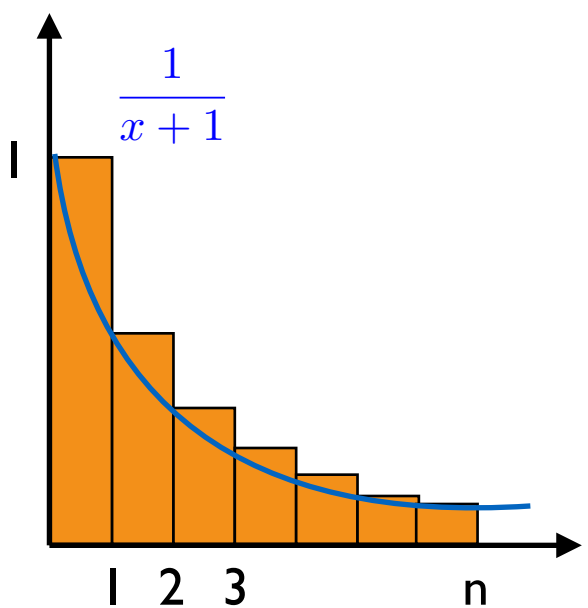
$$\leq \quad 1 + \ln(n)$$

Dealing with summations

5. Comparison with an integral

$$\sum_{i=1}^{n} f(i) \quad \approx \quad \int_{x=1}^{n} f(x)dx$$

$$\sum_{i=1}^{n} \frac{1}{i} \quad \approx \quad \int_{x=1}^{n} \frac{1}{x}dx \quad = \quad \ln(n)$$



$$\sum_{i=1}^{n} \frac{1}{i} \quad \geq \quad \int_{x=0}^{n} \frac{1}{x+1}dx$$

$$= \quad \ln(n+1)$$

Running time analysis:

Dealing with summations

Dealing with recursion

Example: merge sort

Sorting a given list/array of elements:

Merge Sort

1. Recursively sort right half of the list
2. Recursively sort left half of the list
3. Combine (merge) the two sorted lists.

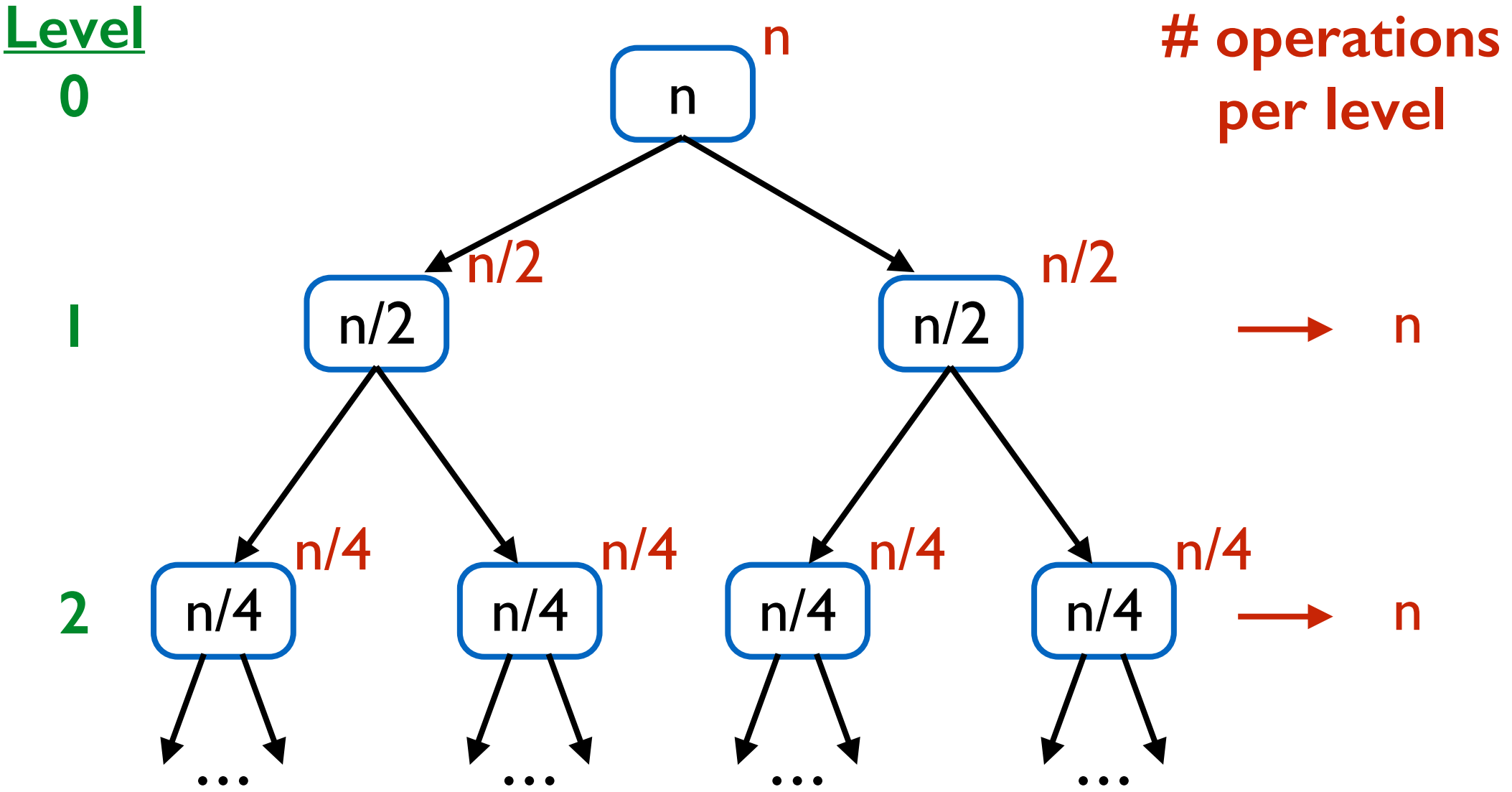
Input size = length of the list = n

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

# of steps not counting the work done by recursive calls:

$$O(n)$$

Recursion tree for merge sort

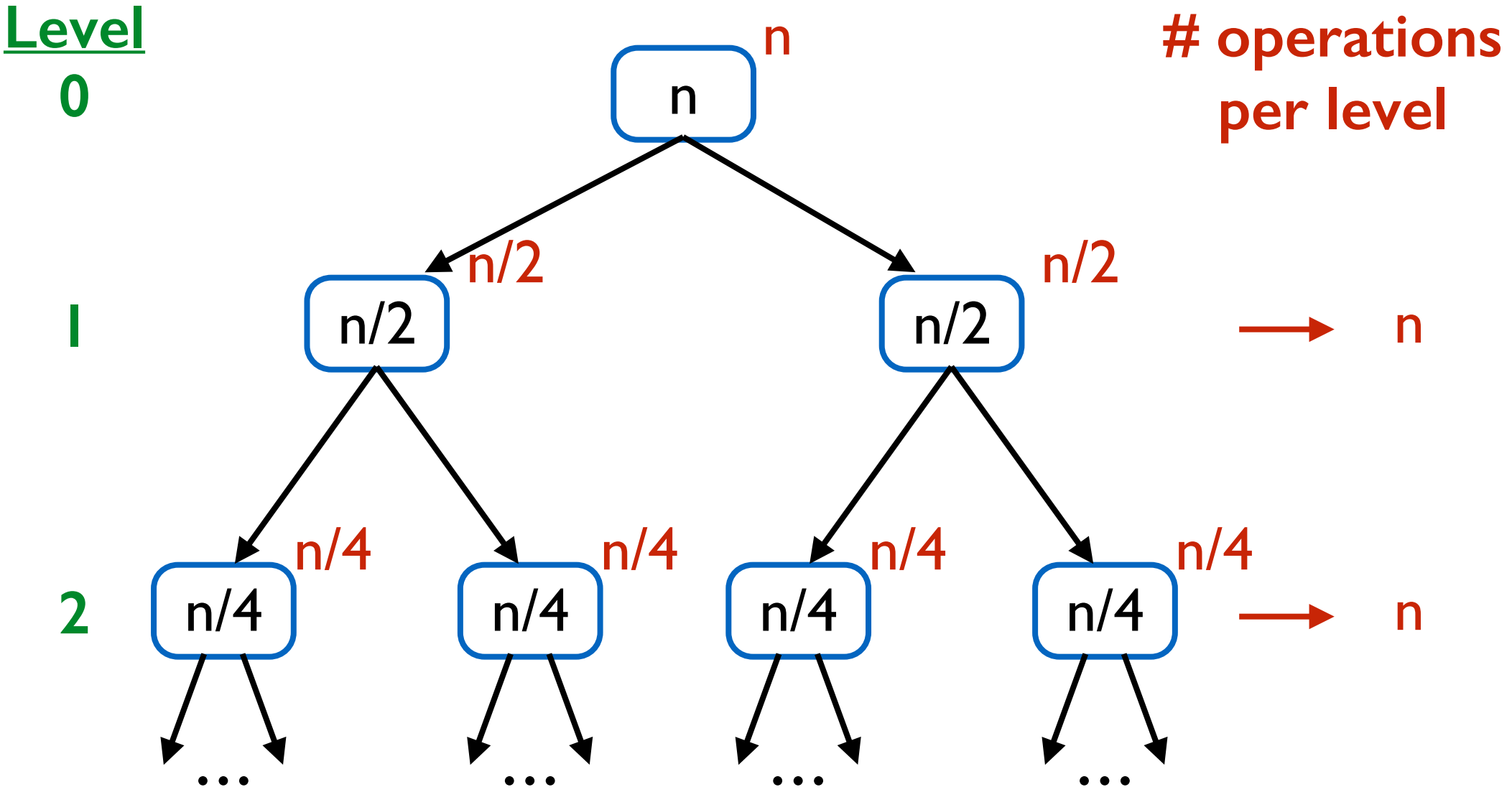


distinct problems at level j: 2^j

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

cn per level

Recursion tree for merge sort



levels: $\log_2 n$

Total cost: $O(n \log n)$

The Master Theorem

Base case: $T(n) \leq C$ for all sufficiently small *n*.

Recursive relation:

$$T(n) \leq a \cdot T(n/b) + O(n^d) \qquad a \geq 1,\ b > 1,\ d \geq 0$$

a: # recursive calls

b: input size shrinkage factor

d: exponent of "combine step"

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

The power of computation/algorithms

(and more exercise with recursion)

Integer Multiplication

Input: 2 n-digit numbers x and y.

Output: The product of x and y.

Grade-School Algorithm:

```
          5 6 7 8
      x   1 2 3 4
  -------------------
        2 2 7 1 2
      1 7 0 3 4
    1 1 3 5 6
+   5 6 7 8
  -------------------
    7 0 0 6 6 5 2
```

n rows

$\longrightarrow$ $O(n)$ operations

$\longrightarrow$ $O(n)$ operations

$\longrightarrow$ $O(n)$ operations

$\longrightarrow$ $O(n)$ operations

Total: $O(n^2)$

Integer Multiplication

You might think:

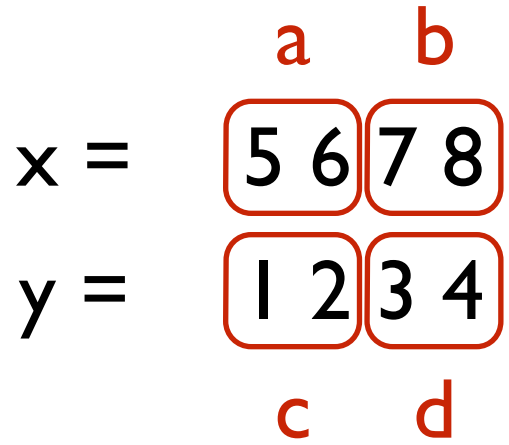
Probably this is the best, what else can you really do ?

A good algorithm designer thinks:

How can we do better ?

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

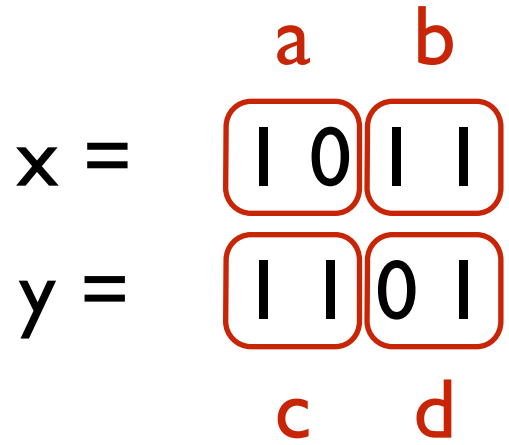
Integer Multiplication



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

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

Integer Multiplication



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

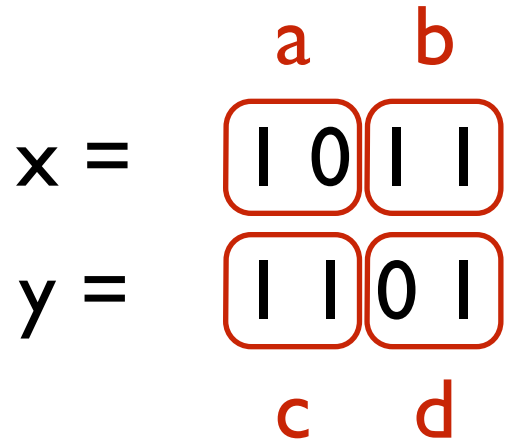
$$y = 2^{n/2}c + d$$

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

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

Why not try recursion then?

Integer Multiplication



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

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

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

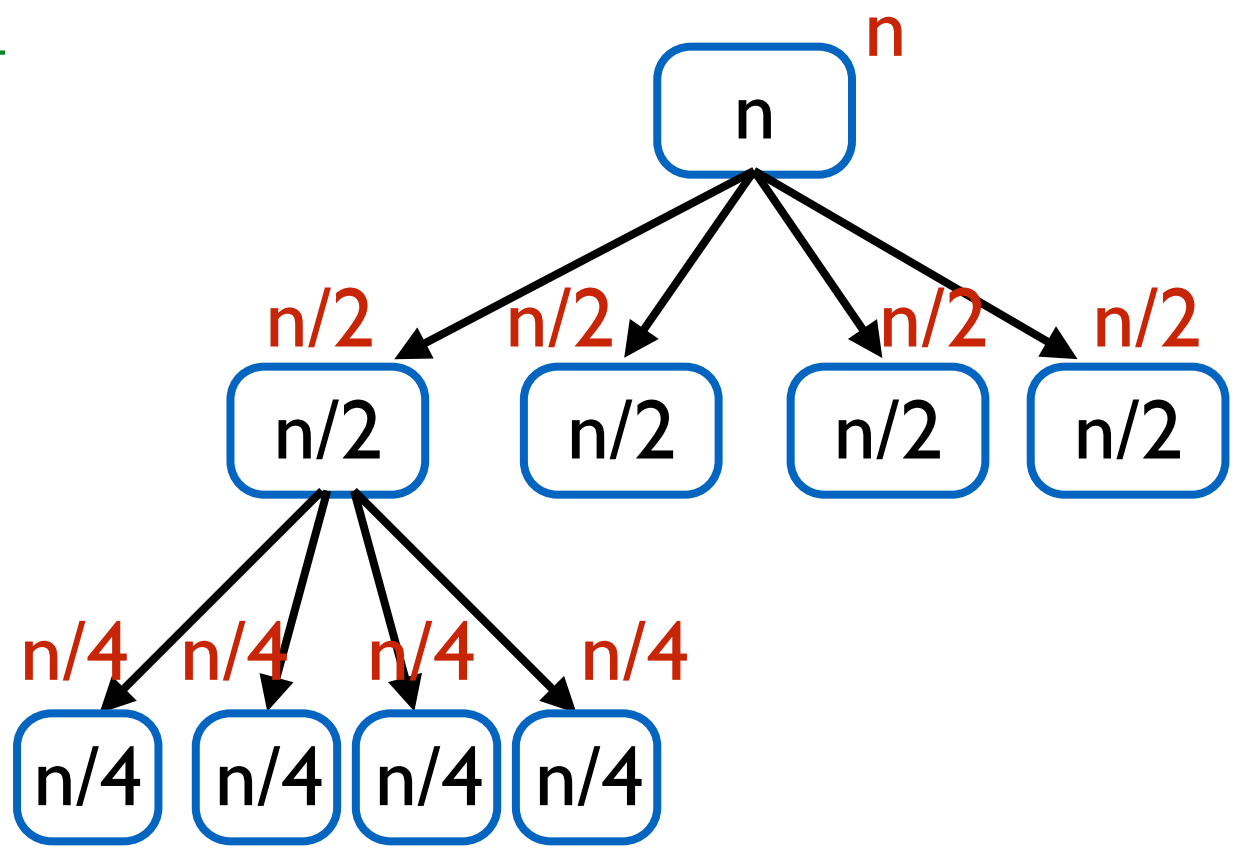
$$= 2^n ac + 2^{n/2}(ad + bc) + bd$$

Recursively compute *ac*, *ad*, *bc*, and *bd*. Do the additions.

Base case: 1 digit numbers.

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

Integer Multiplication



 # operations per level

→ 2n

→ 4n

distinct problems at level j: 4^j

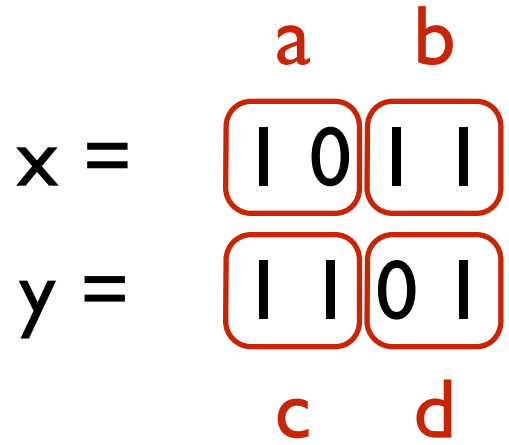
operations per node at level j: $c(n/2^j)$

levels: $\log_2 n$

$cn2^j$ per level

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

Integer Multiplication



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

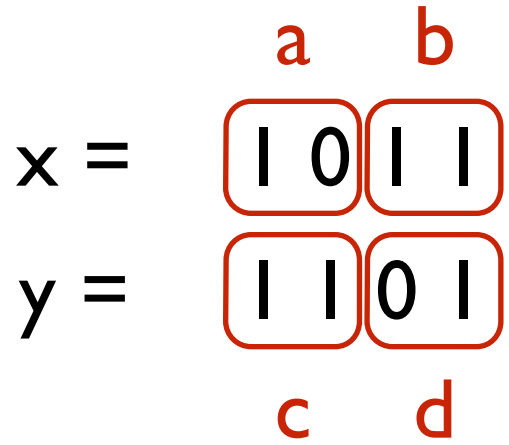
$$y = 2^{n/2}c + d$$

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

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

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



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

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

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

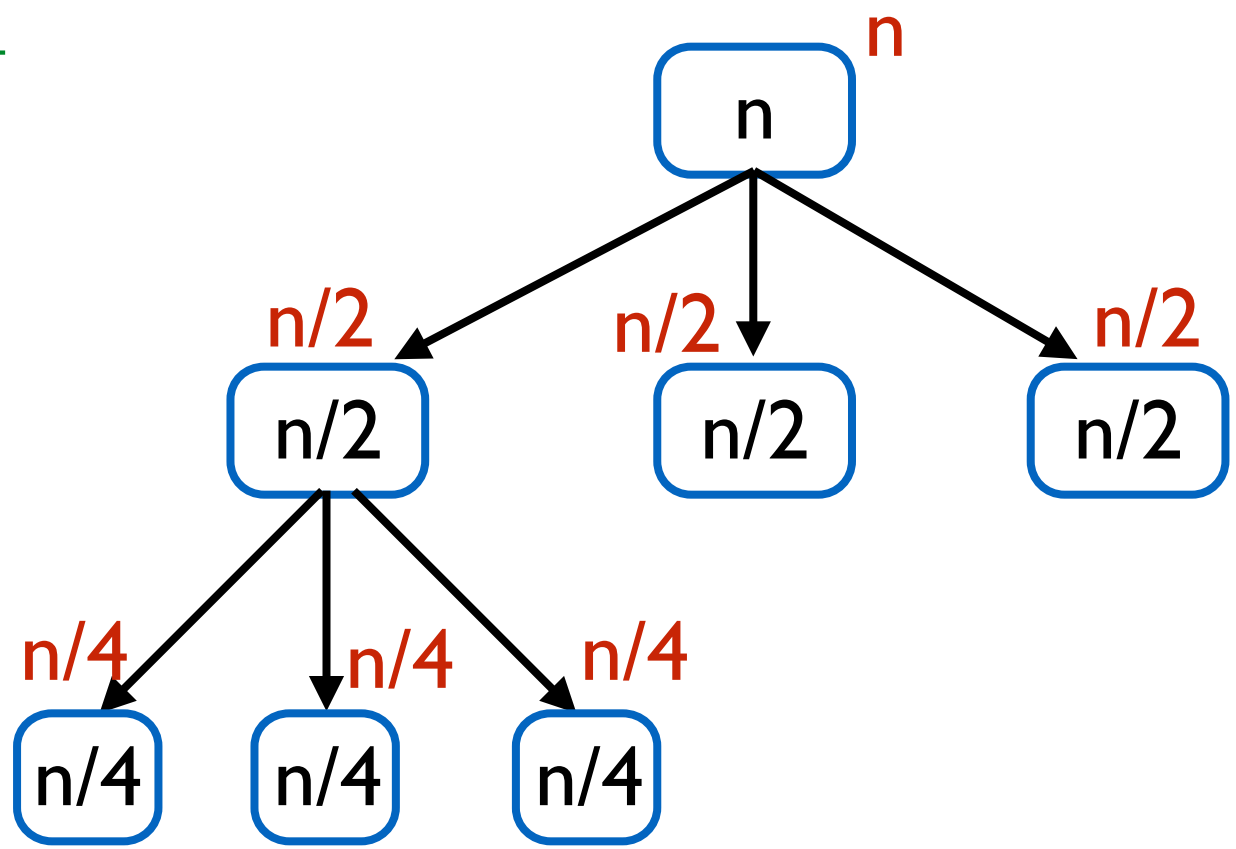
$$= 2^n ac + 2^{n/2}(ad + bc) + bd$$

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

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

Is this better??

Integer Multiplication



 # operations per level

→ 3n/2

→ 9n/4

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

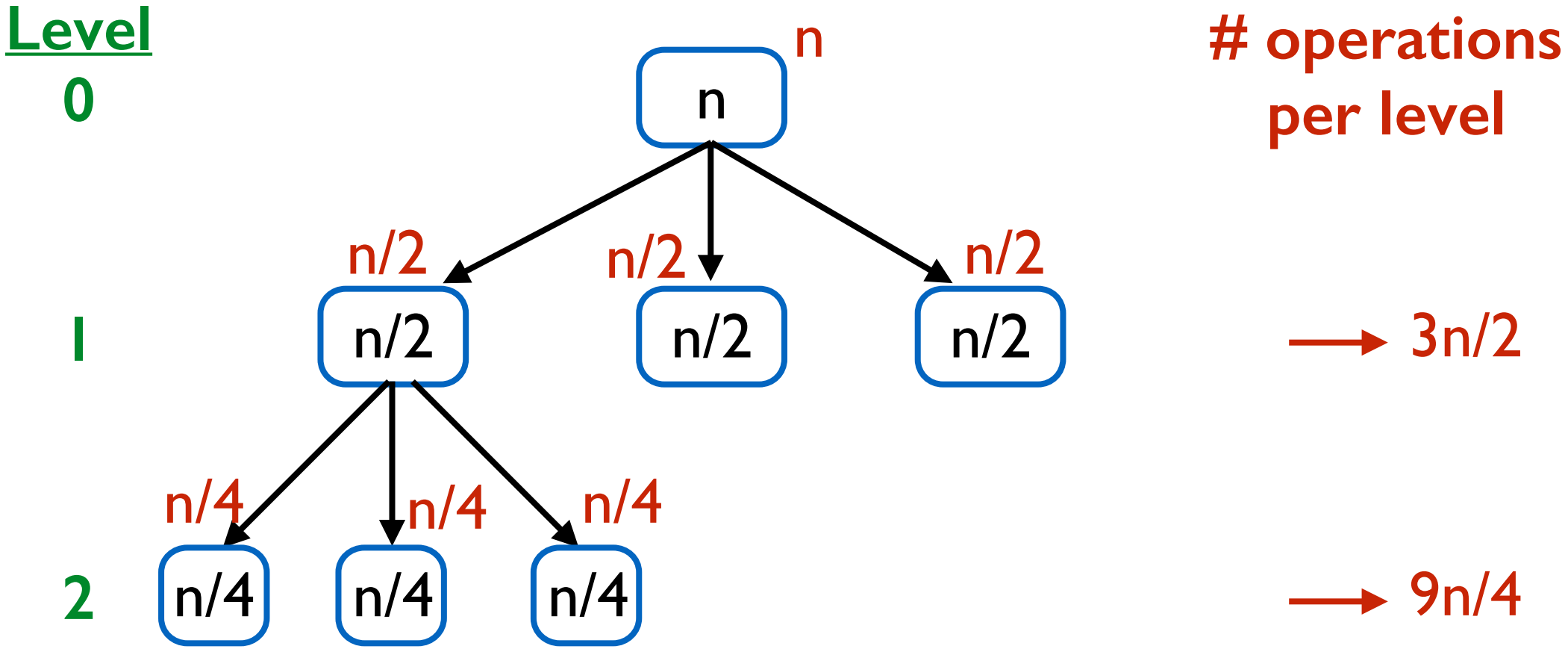
distinct problems at level j: 3^j

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

levels: $\log_2 n$

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

Integer Multiplication



Total cost:

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

Karatsuba Algorithm

Integer Multiplication

You might think:

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

A good algorithm designer thinks:

How can we do better ?

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

Replace 9 multiplications with only 5.

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

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

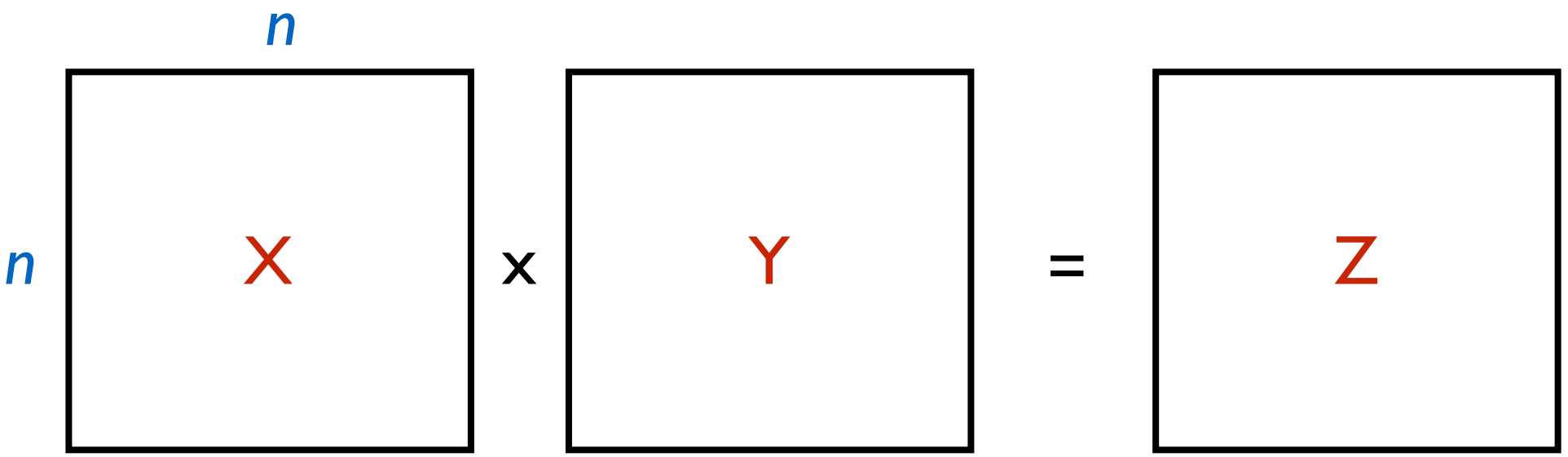
Can do $T(n) \in O(n^{1+\epsilon})$ for any $\epsilon > 0$.

Integer Multiplication

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

Martin Fürer (2007)

Matrix Multiplication



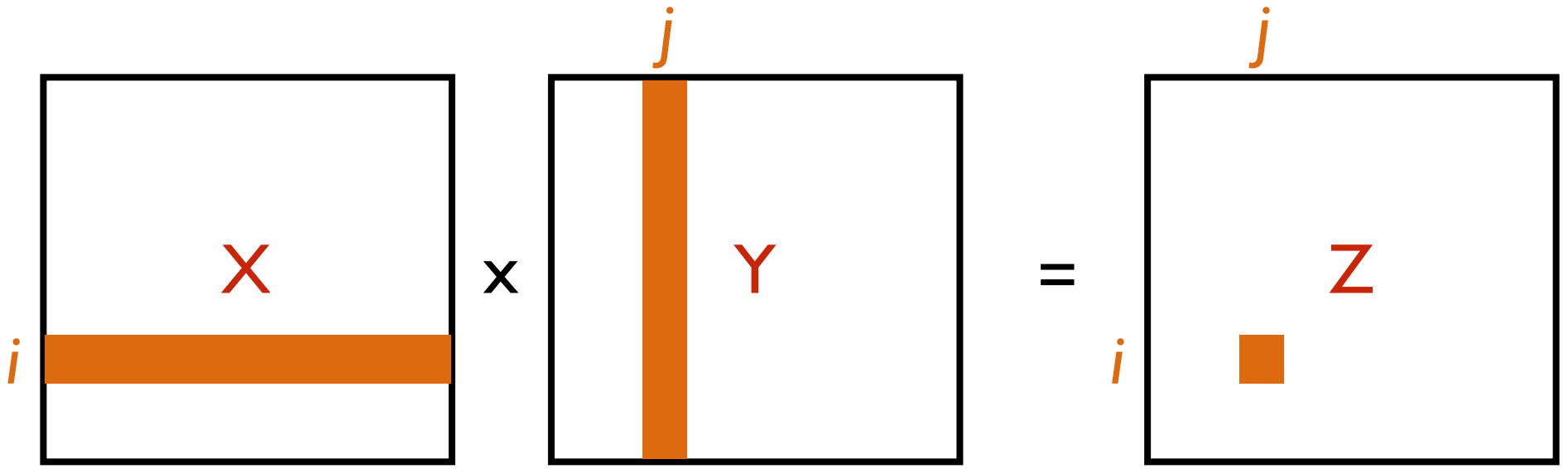
Input: 2 n x n matrices X and Y.

Output: The product of X and Y.

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

Note: input size is $O(n^2)$.

Matrix Multiplication



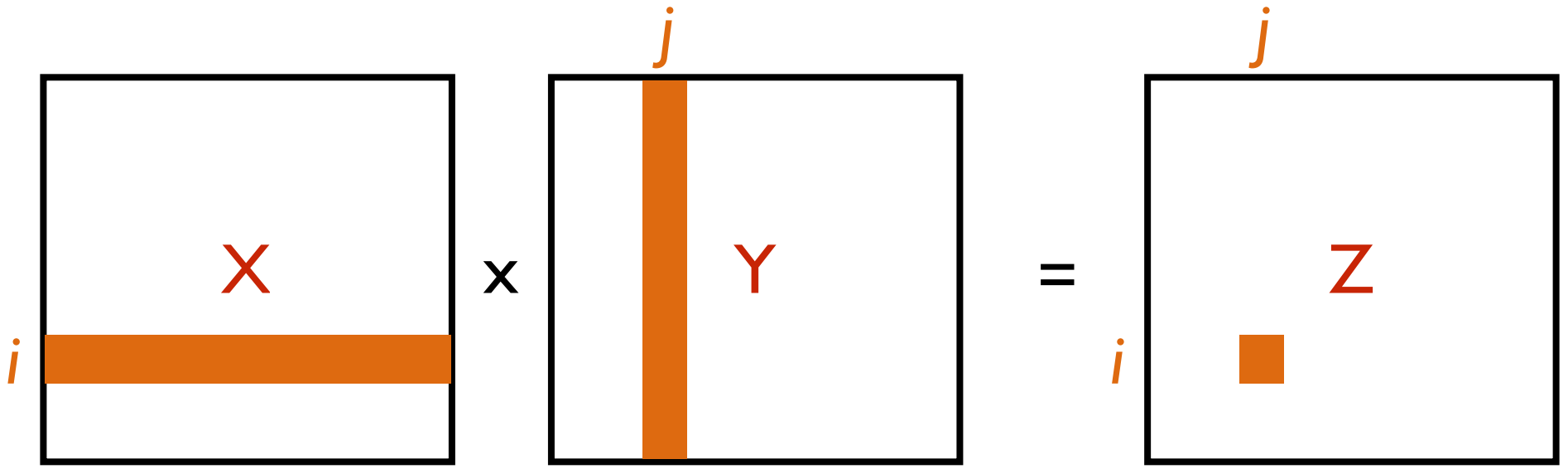
Z[i,j] = (i'th row of X) · (j'th column of Y)

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

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)

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

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

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

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)

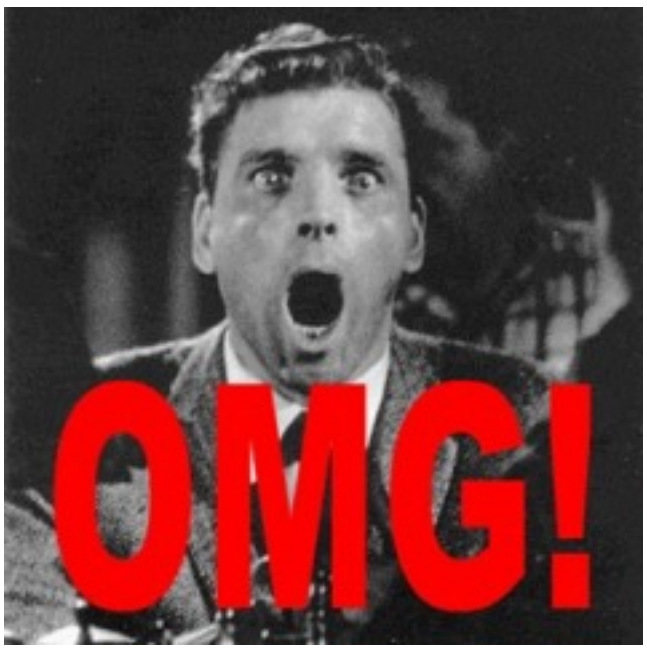
AE+BG = Q1+Q4-Q5+Q7
AF+BH = Q3+Q5
CE+DG = Q2+Q4
CF+DH = Q1+Q3-Q2+Q6

Matrix Multiplication

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

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

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



Matrix Multiplication

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

Matrix Multiplication

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!

Matrix Multiplication

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)

Enormous Open Problem

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

Some other interesting problems

Theorem Proving

Given a mathematical statement and and integer k, is there a proof in ZFC set theory with at most k symbols?

Testing Primality

Given an integer k, is k a prime number?

Factoring

Given an integer k, find its prime factors.

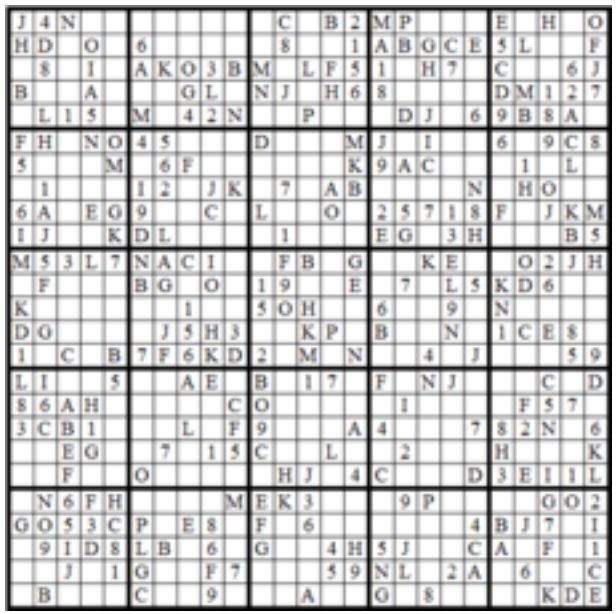
Some other interesting problems

Satisfiability (SAT)

Given a Boolean formula, is it satisfiable?

$$(x_1 \vee x_2) \wedge (x_3 \vee \neg x_2) \wedge \neg x_1$$

Sudoku (arbitrary dimension)



Polynomial time and the class P

Complexity classes

$$\mathrm{DTIME}(T(n)) = \{L : L \text{ is decided by an } O(T(n)) \text{ time algorithm.}\}$$

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathrm{DTIME}(n^k)$$

$$\mathsf{EXP} = \bigcup_{k \in \mathbb{N}} \mathrm{DTIME}(2^{n^k})$$

$$\mathsf{P} \subseteq \mathsf{EXP}$$

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:

In P	Efficient.
Not in P	Not efficient.

- P is not meant to mean "efficient in practice"

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

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

What is efficient in theory and in practice ?

In theory:

In P	Efficient.
Not in P	Not efficient.

- Being in P is a fundamental property of a problem, rather than a property of how we solve the problem.

- P is about mathematical insight into a problem's structure.

- Whether, say "Theorem Proving" is in P or not is a mathematical question about the nature of the problem.

What is efficient in theory and in practice ?

In theory:

In P	Efficient.
Not in P	Not efficient.

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

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

- Wouldn't make sense to cut it off at some specific exponent.

What is efficient in theory and in practice ?

In theory:

In P	Efficient.
Not in P	Not efficient.

- Plus, big exponents don't really arise.

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

- **Summary**: Being in P vs not being in P is a qualitative difference, not a quantitative one.

Efficiency limits on computation

Is every decidable problem in P ?

The field of polynomial time algorithms is very rich!

Polynomial time algorithms can do really amazing things.

Maybe they can solve every decidable problem…

Well, they can't!

This can be proved using a diagonalization argument.

Recall how we showed HALT is undecidable

$$\text{HALT} = \{\langle M, x\rangle : M \text{ halts on input } x.\}$$

Suppose M_{HALT} decides HALT.

Then we can define M_{TURING} :

$M_{\text{TURING}}(\langle M\rangle)$:

run $M_{\text{HALT}}(\langle M, M\rangle)$ and "flip the answer"

if $M_{\text{HALT}}(\langle M, M\rangle) =$ YES

run for infinity

if $M_{\text{HALT}}(\langle M, M\rangle) =$ NO

halt

Contradiction when you look at $M_{\text{TURING}}(\langle M_{\text{TURING}}\rangle)$

Showing a limit of efficient computation

We can use a similar strategy to show that there is a decidable language that takes, say, at lest n^2 time.

$$\text{HWTB} = \text{HALT WITH TIME BOUND}$$

$$\text{HWTB} = \{\langle M \rangle : M(\langle M \rangle) \text{ takes at most } n^3 \text{ steps.}\}$$

Claim 1: HWTB is decidable.

Claim 2: HWTB cannot be decided in n^2 steps.

Suppose it can be decided in n^2 steps.

Let M_{HWTB} be a decider with this property.

We'll describe M_{TURING} that uses M_{HWTB} :

Showing a limit of efficient computation

$$\text{HWTB} = \{\langle M \rangle : M(\langle M \rangle) \text{ takes at most } n^3 \text{ steps.}\}$$

We'll describe M_{TURING} that uses M_{HWTB} :

$M_{\text{TURING}}(\langle M \rangle)$:

run $M_{\text{HWTB}}(\langle M \rangle)$ and "flip the answer"

if $M_{\text{HWTB}}(\langle M \rangle) = \text{YES}$

run for infinity

if $M_{\text{HWTB}}(\langle M \rangle) = \text{NO}$

halt

What happens when we run $M_{\text{TURING}}(\langle M_{\text{TURING}} \rangle)$?

Showing a limit of efficient computation

$M_{\text{TURING}}(\langle M \rangle)$:

run $M_{\text{HWTB}}(\langle M \rangle)$ and "flip the answer"

if $M_{\text{HWTB}}(\langle M \rangle) = \text{YES}$

run for infinity

if $M_{\text{HWTB}}(\langle M \rangle) = \text{NO}$

halt

What happens when we run $M_{\text{TURING}}(\langle M_{\text{TURING}} \rangle)$?

If $M_{\text{HWTB}}(\langle M_{\text{TURING}} \rangle) = \text{YES}$

$M_{\text{TURING}}(\langle M_{\text{TURING}} \rangle)$ should stop in n^3 steps.

But it goes into an infinite loop.

If $M_{\text{HWTB}}(\langle M_{\text{TURING}} \rangle) = \text{NO}$

$M_{\text{TURING}}(\langle M_{\text{TURING}} \rangle)$ should take more than n^3 steps.

But it takes $n^2 + c$ steps.

Showing a limit of efficient computation

So our assumption that there was a decider for HWTB that used n^2 steps was false.

Nothing very special about n^2.
Could also consider, say, exponential running time.

Showing a limit of efficient computation

If you are a bit more careful about it, you can prove a much stronger statement:

Time Hierarchy Theorem:

Let $T(n)$ be a time-constructible function, and $\epsilon > 0$.

Then there is a problem which cannot be decided in time $T(n)$, but can be decided in time $T(n)^{1+\epsilon}$.

i.e., $\mathrm{DTIME}(T(n)) \subsetneq \mathrm{DTIME}(T(n)^{1+\epsilon})$

Can you cheat exponential time?

How could you try to cheat exponential time?

Make every step exponentially fast.

Time travel to the future.