



<http://algs4.cs.princeton.edu>

## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

# Overview: introduction to advanced topics

---

## Main topics. [next 3 lectures]

- Reduction: design algorithms, establish lower bounds, classify problems.
- Linear programming: the ultimate practical problem-solving model.
- Intractability: problems beyond our reach.

## Shifting gears.

- From individual problems to problem-solving models.
- From linear/quadratic to polynomial/exponential scale.
- From details of implementation to conceptual framework.

## Goals.

- Place algorithms we've studied in a larger context.
- Introduce you to important and essential ideas.
- Inspire you to learn more about algorithms!



## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

## Bird's-eye view

---

**Desiderata.** Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	$N$	min, max, median, Burrows-Wheeler transform, ...
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, ...
quadratic	$N^2$	?
$\vdots$	$\vdots$	$\vdots$
exponential	$c^N$	?

**Frustrating news.** Huge number of problems have defied classification.

## Bird's-eye view

---

**Desiderata.** Classify **problems** according to computational requirements.

**Desiderata'.**

Suppose we could (could not) solve problem  $X$  efficiently.  
What else could (could not) we solve efficiently?

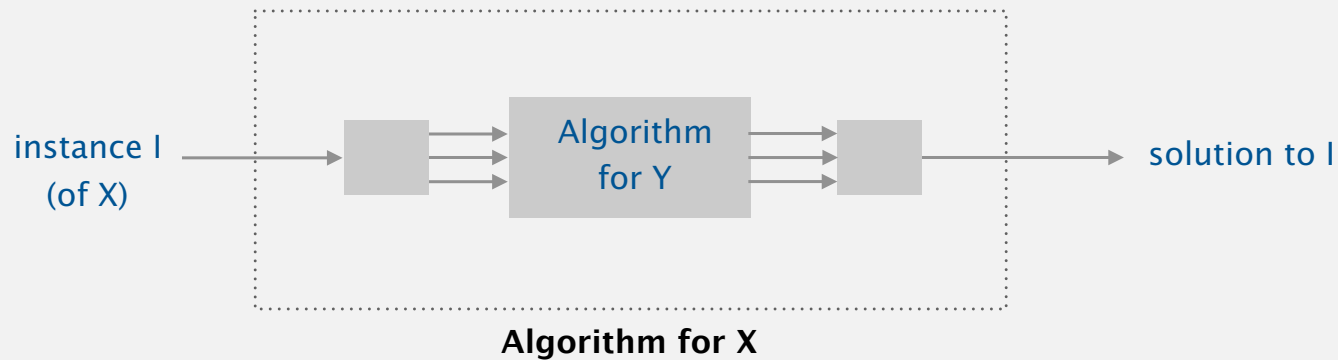


*“ Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. ” — Archimedes*

# Reduction

---

**Def.** Problem  $X$  **reduces to** problem  $Y$  if you can use an algorithm that solves  $Y$  to help solve  $X$ .



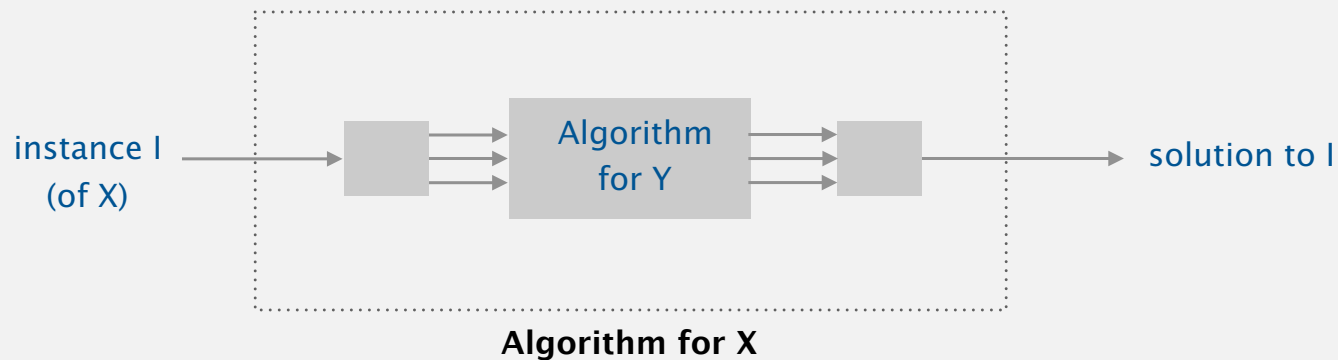
Cost of solving  $X$  = total cost of solving  $Y$  + cost of reduction.

↑  
perhaps many calls to  $Y$   
on problems of different sizes

↑  
preprocessing and postprocessing

# Reduction

**Def.** Problem  $X$  **reduces to** problem  $Y$  if you can use an algorithm that solves  $Y$  to help solve  $X$ .



**Ex 1.** [finding the median reduces to sorting]

To find the median of  $N$  items:

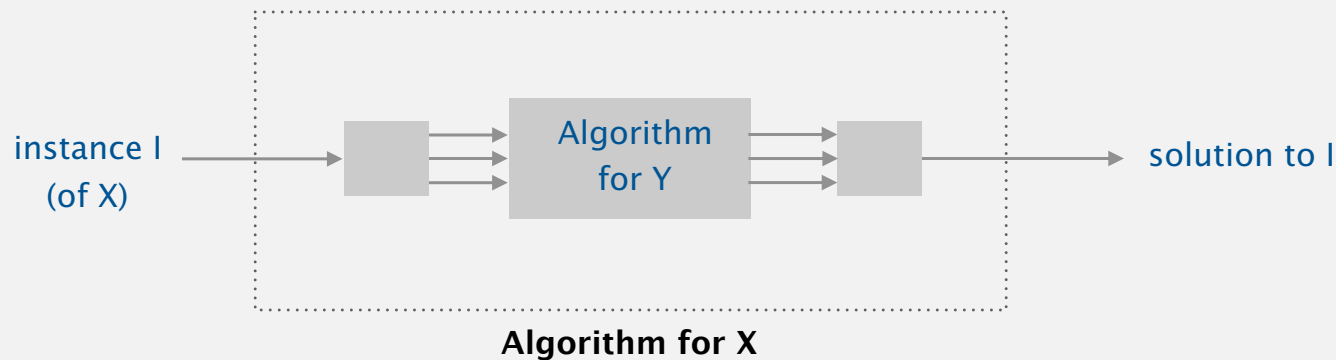
- Sort  $N$  items.
- Return item in the middle.

Cost of solving finding the median.  $N \log N + 1$ .

cost of sorting  $\swarrow$   $N \log N$   $\nwarrow$  cost of reduction  $+ 1$

# Reduction

**Def.** Problem  $X$  **reduces to** problem  $Y$  if you can use an algorithm that solves  $Y$  to help solve  $X$ .



**Ex 2.** [element distinctness reduces to sorting]

To solve element distinctness on  $N$  items:

- Sort  $N$  items.
- Check adjacent pairs for equality.

Cost of solving element distinctness.  $N \log N + N$ .

cost of sorting  
cost of reduction





## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*



## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

## Reduction: design algorithms

---

**Def.** Problem  $X$  **reduces to** problem  $Y$  if you can use an algorithm that solves  $Y$  to help solve  $X$ .

**Design algorithm.** Given algorithm for  $Y$ , can also solve  $X$ .

**Ex.**

- 3-collinear reduces to sorting. [assignment]
- Finding the median reduces to sorting.
- Element distinctness reduces to sorting.
- CPM reduces to topological sort. [shortest paths lecture]
- Arbitrage reduces to shortest paths. [shortest paths lecture]
- Burrows-Wheeler transform reduces to suffix sort. [assignment]
- ...

**Mentality.** Since I know how to solve  $Y$ , can I use that algorithm to solve  $X$ ?



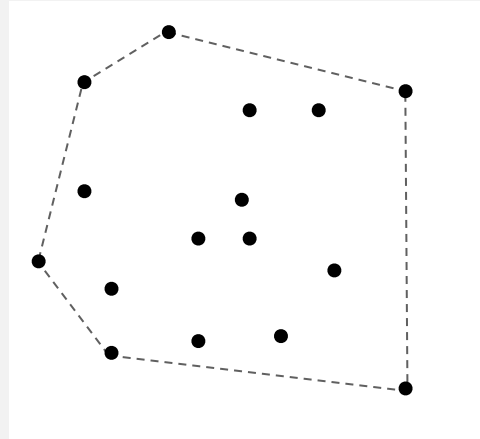
programmer's version: I have code for  $Y$ . Can I use it for  $X$ ?

# Convex hull reduces to sorting

---

**Sorting.** Given  $N$  distinct integers, rearrange them in ascending order.

**Convex hull.** Given  $N$  points in the plane, identify the extreme points of the convex hull (in counterclockwise order).



convex hull

```
1251432
2861534
3988818
4190745
8111033
13546464
89885444
43434213
34435312
```

sorting

**Proposition.** Convex hull reduces to sorting.

**Pf.** Graham scan algorithm (see next slide).

**Cost of convex hull.**  $N \log N + N$ .

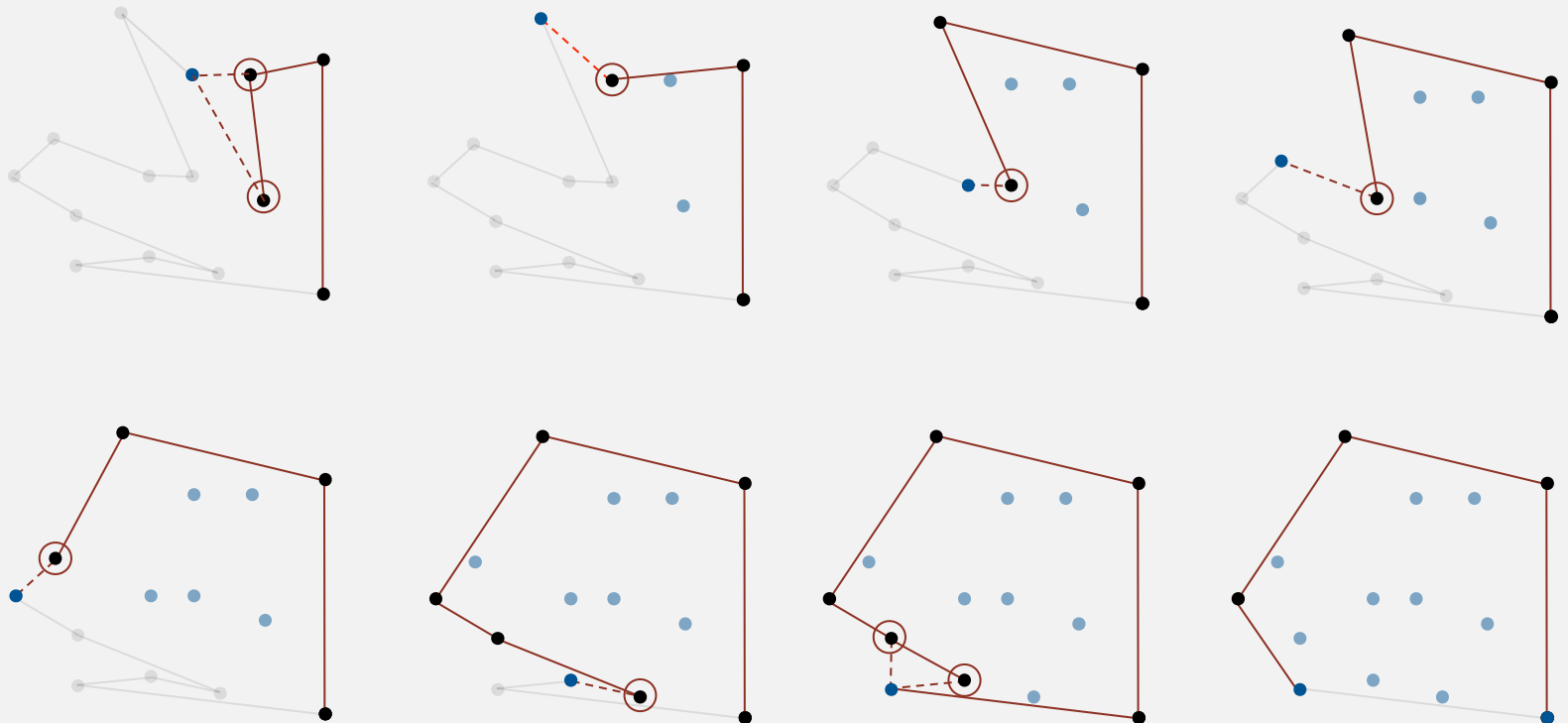
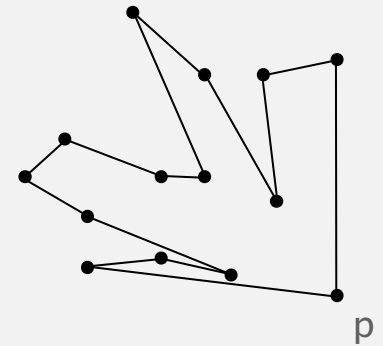
cost of sorting      cost of reduction

↙                      ↘

# Graham scan algorithm

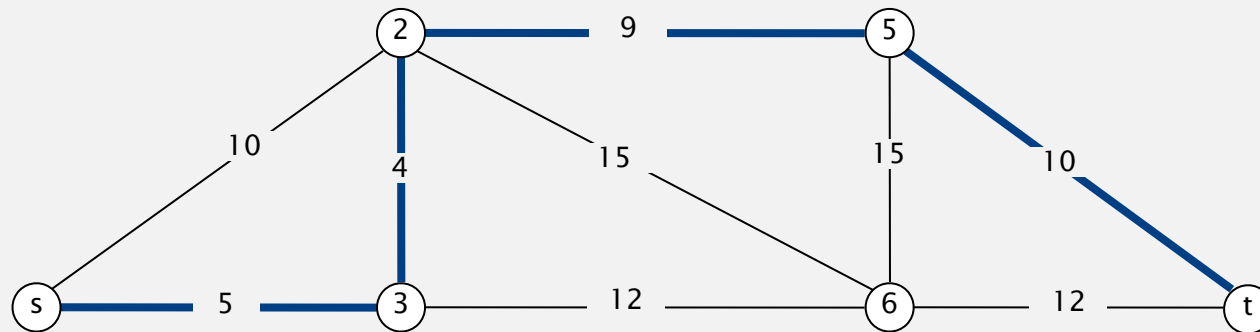
## Graham scan.

- Choose point  $p$  with smallest (or largest)  $y$ -coordinate.
- **Sort** points by polar angle with  $p$  to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.

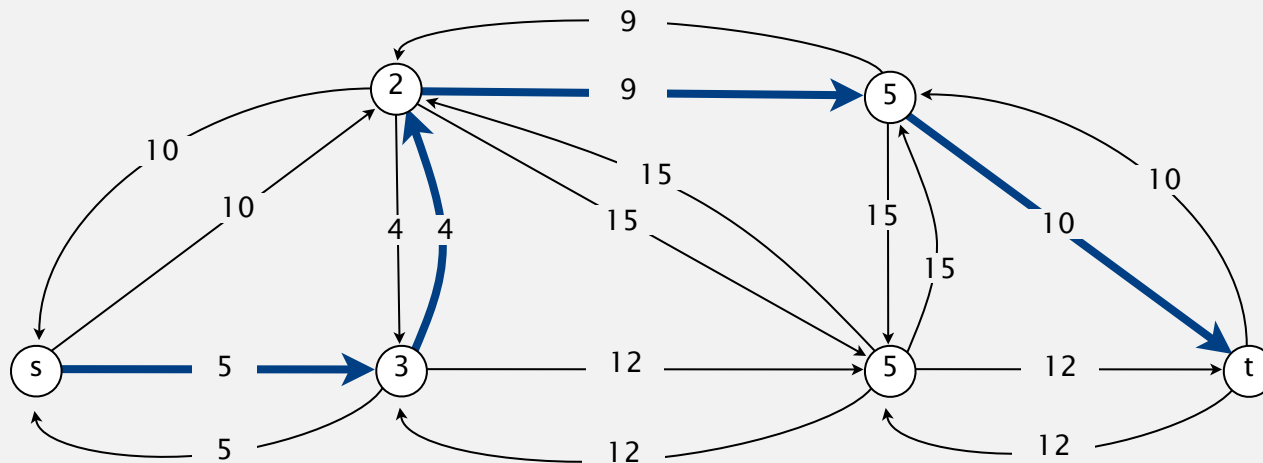


# Shortest paths on edge-weighted graphs and digraphs

**Proposition.** Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

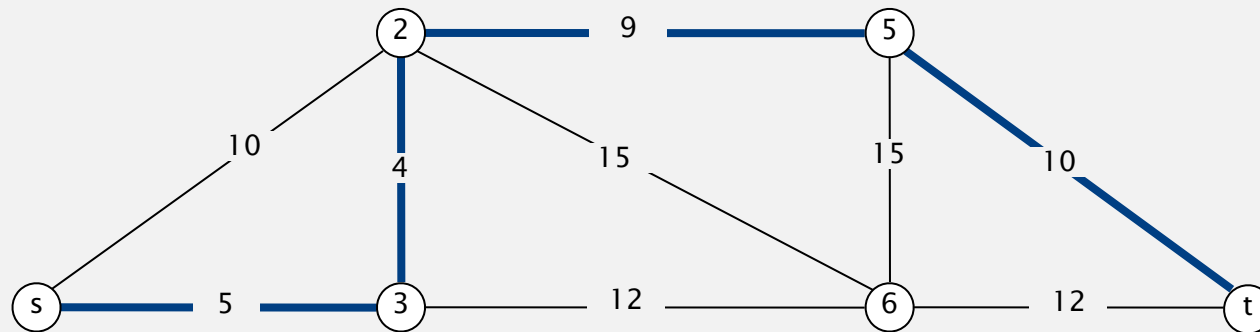


**Pf.** Replace each undirected edge by two directed edges.



# Shortest paths on edge-weighted graphs and digraphs

**Proposition.** Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.



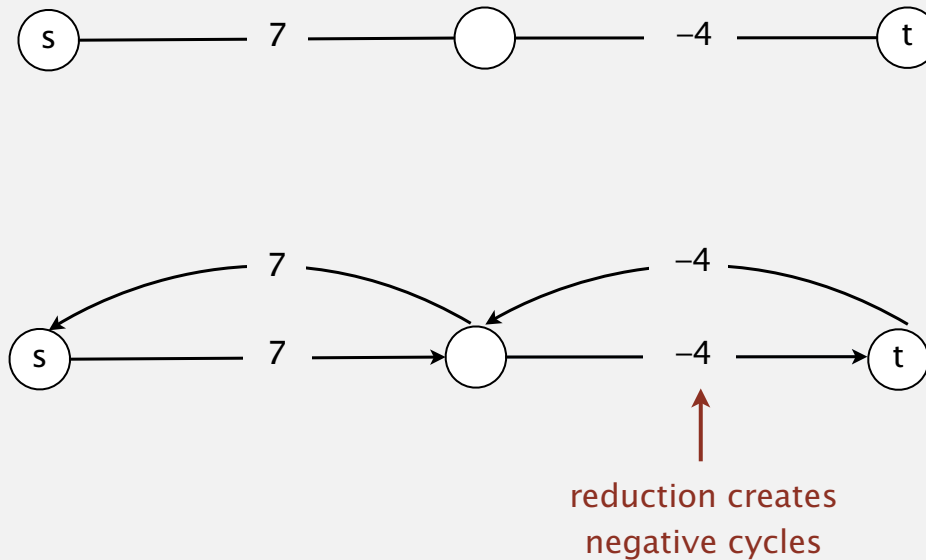
cost of shortest  
paths in digraph

cost of reduction

Cost of undirected shortest paths.  $E \log V + E$ .

# Shortest paths with negative weights

**Caveat.** Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles).



**Remark.** Can still solve shortest-paths problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

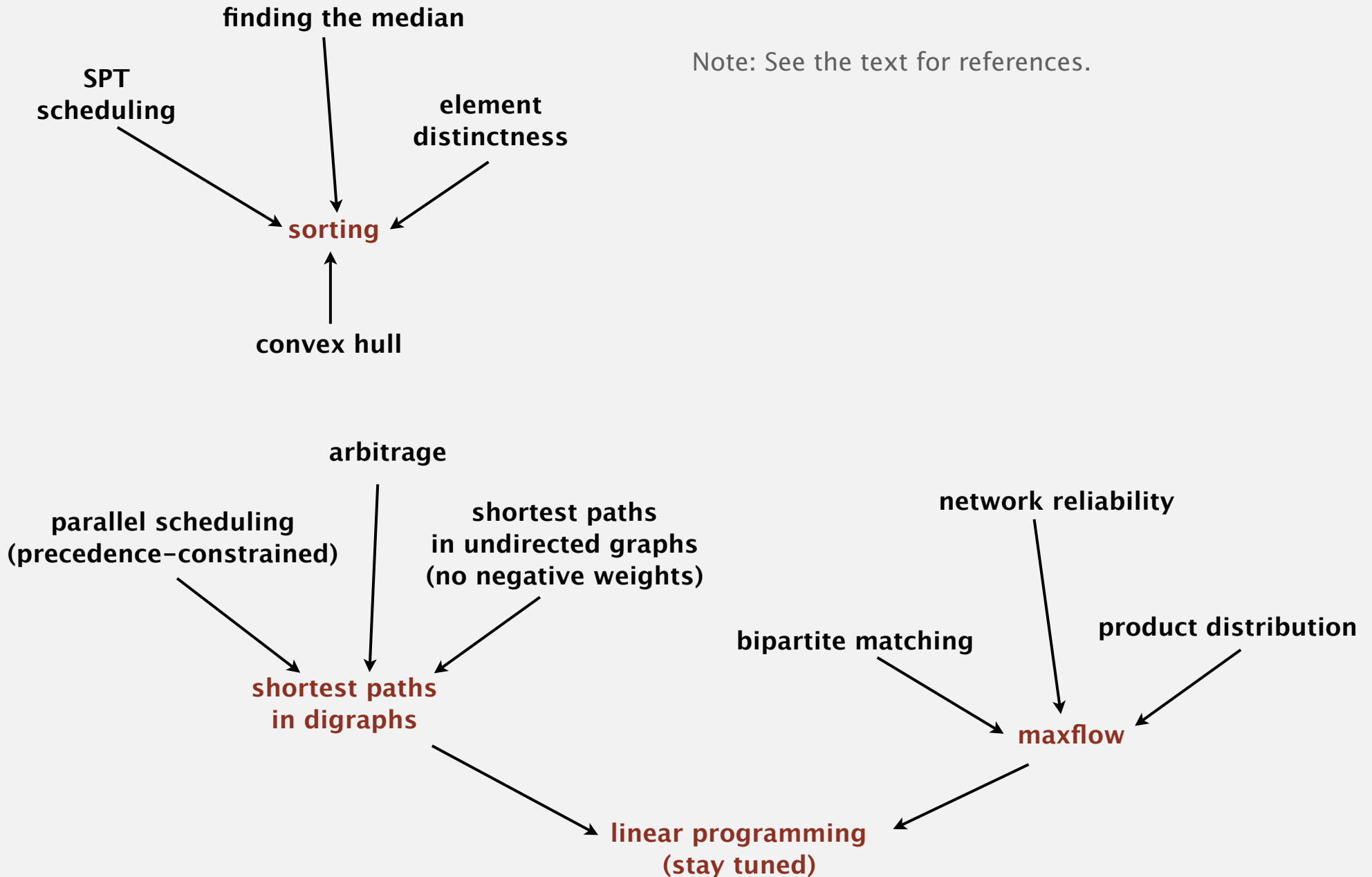
reduces to weighted  
non-bipartite matching (!)



# Linear-time reductions involving familiar problems

---

Note: See the text for references.





## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*



## 6.5 REDUCTIONS

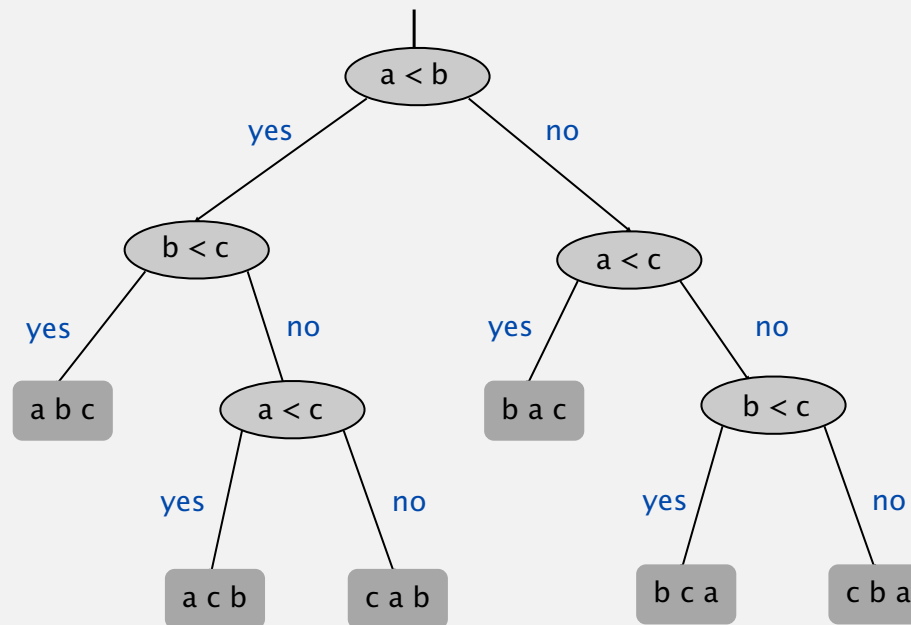
---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

## Bird's-eye view

**Goal.** Prove that a problem requires a certain number of steps.

**Ex.** In decision tree model, any compare-based sorting algorithm requires  $\Omega(N \log N)$  compares in the worst case.



argument must apply to all conceivable algorithms

**Bad news.** Very difficult to establish lower bounds from scratch.

**Good news.** Spread  $\Omega(N \log N)$  lower bound to  $Y$  by reducing sorting to  $Y$ .

assuming cost of reduction is not too high

## Linear-time reductions

---

**Def.** Problem  $X$  **linear-time reduces** to problem  $Y$  if  $X$  can be solved with:

- Linear number of standard computational steps.
- Constant number of calls to  $Y$ .

**Ex.** Almost all of the reductions we've seen so far. [Which ones weren't?]

**Establish lower bound:**

- If  $X$  takes  $\Omega(N \log N)$  steps, then so does  $Y$ .
- If  $X$  takes  $\Omega(N^2)$  steps, then so does  $Y$ .

**Mentality.**

- If I could easily solve  $Y$ , then I could easily solve  $X$ .
- I can't easily solve  $X$ .
- Therefore, I can't easily solve  $Y$ .

# Lower bound for convex hull

**Proposition.** In quadratic decision tree model, any algorithm for sorting  $N$  integers requires  $\Omega(N \log N)$  steps.

allows linear or quadratic tests:

$$\underline{x}_i < \underline{x}_j \text{ or } (x_j - x_i)(x_k - x_i) - (x_j)(\underline{x}_i - x_i) < 0$$

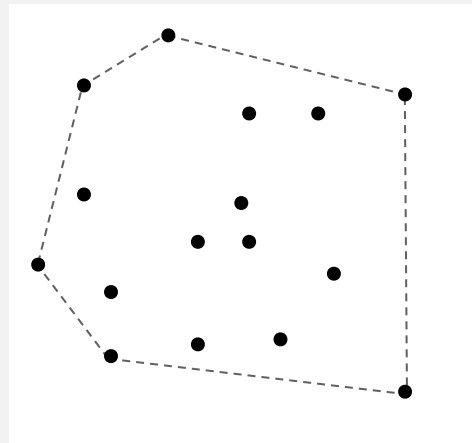
**Proposition.** Sorting linear-time reduces to convex hull.

**Pf.** [see next slide]

lower-bound mentality:  
if I can solve convex hull  
efficiently, I can sort efficiently

1251432  
2861534  
3988818  
4190745  
8111033  
13546464  
89885444  
43434213  
34435312

sorting



convex hull

linear or  
quadratic tests

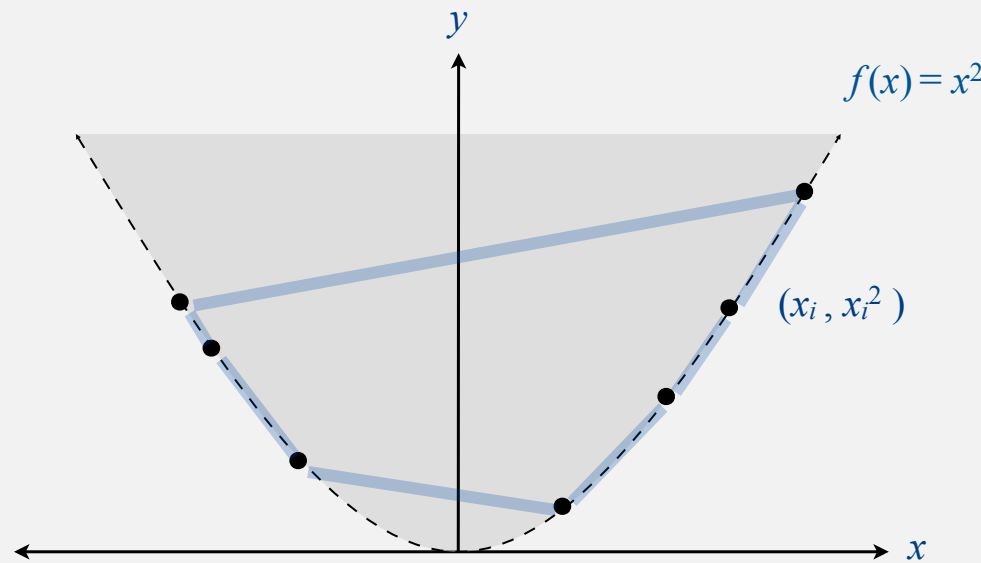
**Implication.** Any ccw-based convex hull algorithm requires  $\Omega(N \log N)$  ops.

# Sorting linear-time reduces to convex hull

**Proposition.** Sorting linear-time reduces to convex hull.

- Sorting instance:  $x_1, x_2, \dots, x_N$ .
- Convex hull instance:  $(x_1, x_1^2), (x_2, x_2^2), \dots, (x_N, x_N^2)$ .

lower-bound mentality:  
if I can solve convex hull  
efficiently, I can sort efficiently



**Pf.**

- Region  $\{x : x^2 \geq x\}$  is convex  $\Rightarrow$  all points are on hull.
- Starting at point with most negative  $x$ , counterclockwise order of hull points yields integers in ascending order.

## Establishing lower bounds: summary

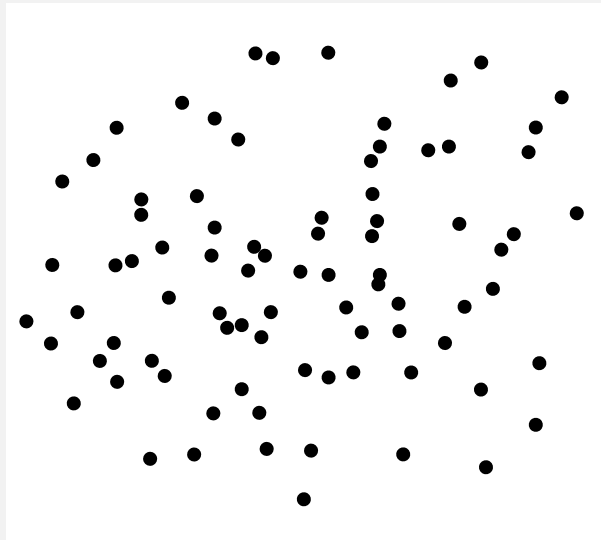
---

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time convex hull algorithm exists?

A1. [hard way] Long futile search for a linear-time algorithm.

A2. [easy way] Linear-time reduction from sorting.



convex hull







## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*



## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

# Classifying problems: summary

---

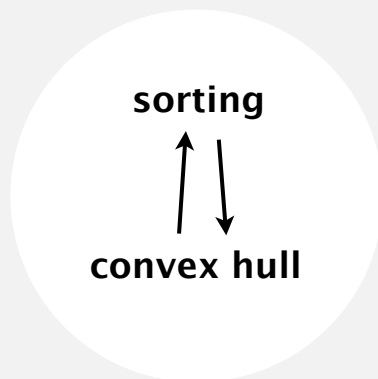
**Desiderata.** Problem with algorithm that matches lower bound.

**Ex.** Sorting and convex hull have complexity  $N \log N$ .

**Desiderata'.** Prove that two problems  $X$  and  $Y$  have the same complexity.

- First, show that problem  $X$  linear-time reduces to  $Y$ .
- Second, show that  $Y$  linear-time reduces to  $X$ .
- Conclude that  $X$  and  $Y$  have the same complexity.

even if we don't know what it is!



## Caveat

---

**Sort.** Given  $N$  distinct integers, rearrange them in ascending order.


**Convex Hull.** Given  $N$  points in the plane, identify the extreme points of the convex hull (in counterclockwise order).

**Proposition.** Sort linear-time reduces to Convex Hull.

**Proposition.** Convex Hull linear-time reduces to Sort.

**Conclusion.** Sort and Convex Hull have the same complexity.

### A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements `sort()` using `convexHull()`. 
- Bob implements `convexHull()` using `sort()`.
- Infinite reduction loop!
- Who's fault?

well, maybe not so realistic

# Integer arithmetic reductions

**Integer multiplication.** Given two  $N$ -bit integers, compute their product.

Brute force.  $N^2$  bit operations.

[illegible]

# Integer arithmetic reductions

---

**Integer multiplication.** Given two  $N$ -bit integers, compute their product.

**Brute force.**  $N^2$  bit operations.

problem	arithmetic	order of growth
integer multiplication	$a \times b$	$M(N)$
integer division	$a / b, a \bmod b$	$M(N)$
integer square	$a^2$	$M(N)$
integer square root	$\lfloor \sqrt{a} \rfloor$	$M(N)$

integer arithmetic problems with the same complexity as integer multiplication

**Q.** Is brute-force algorithm optimal?

# History of complexity of integer multiplication

---

year	algorithm	order of growth
?	brute force	$N^2$
1962	Karatsuba-Ofman	$N^{1.585}$
1963	Toom-3, Toom-4	$N^{1.465}$ , $N^{1.404}$
1966	Toom-Cook	$N^{1+\epsilon}$
1971	Schönhage–Strassen	$N \log N \log \log N$
2007	Fürer	$N \log N 2^{\log^* N}$
?	?	$N$

number of bit operations to multiply two  $N$ -bit integers

used in Maple, Mathematica, gcc, cryptography, ...

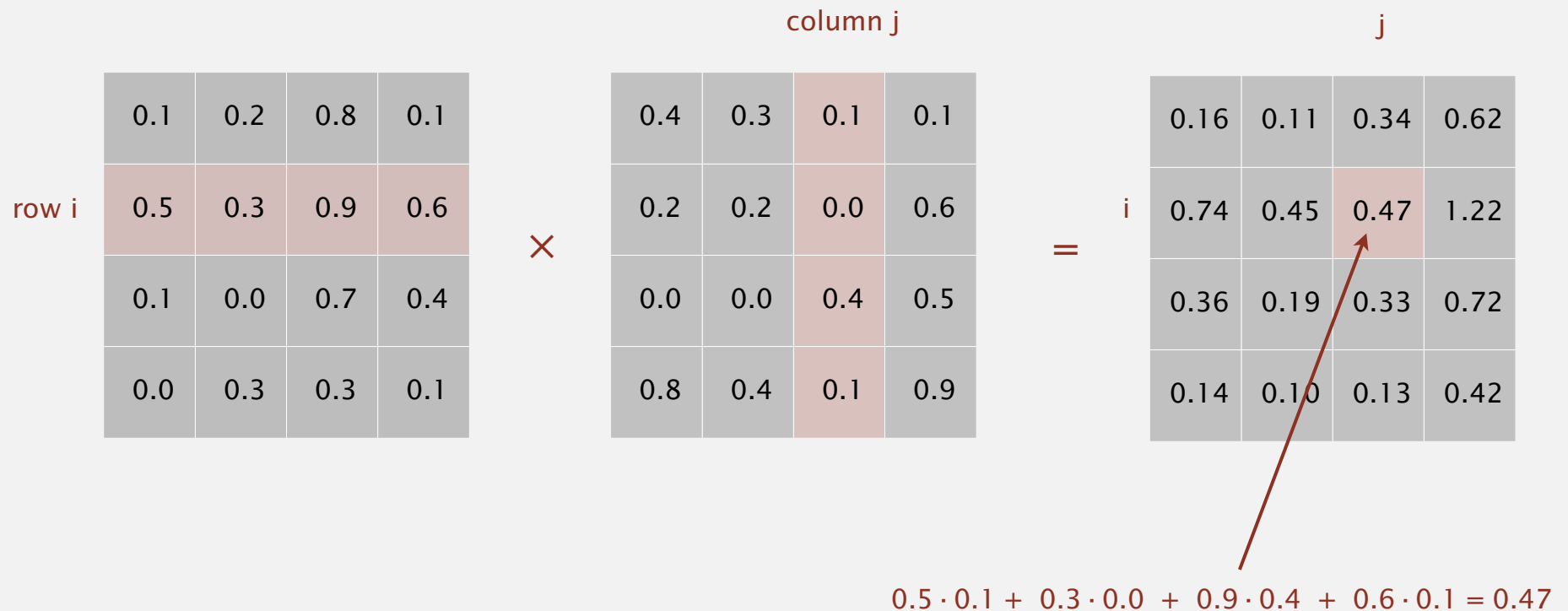
**Remark.** GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

**GMP**  
«Arithmetic without limitations»

# Linear algebra reductions

**Matrix multiplication.** Given two  $N$ -by- $N$  matrices, compute their product.

**Brute force.**  $N^3$  flops.





# Linear algebra reductions

---

**Matrix multiplication.** Given two  $N$ -by- $N$  matrices, compute their product.

**Brute force.**  $N^3$  flops.

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	MM(N)
matrix inversion	$A^{-1}$	MM(N)
determinant	$ A $	MM(N)
system of linear equations	$Ax = b$	MM(N)
LU decomposition	$A = LU$	MM(N)
least squares	$\min \ Ax - b\ _2$	MM(N)

numerical linear algebra problems with the same complexity as matrix multiplication

Q. Is brute-force algorithm optimal?

# History of complexity of matrix multiplication

---

year	algorithm	order of growth
?	brute force	$N^3$
1969	Strassen	$N^{2.808}$
1978	Pan	$N^{2.796}$
1979	Bini	$N^{2.780}$
1981	Schönhage	$N^{2.522}$
1982	Romani	$N^{2.517}$
1982	Coppersmith-Winograd	$N^{2.496}$
1986	Strassen	$N^{2.479}$
1989	Coppersmith-Winograd	$N^{2.376}$
2010	Strother	$N^{2.3737}$
2011	Williams	$N^{2.3727}$
?	?	$N^{2 + \epsilon}$

number of floating-point operations to multiply two N-by-N matrices

## Birds-eye view: review

---

**Desiderata.** Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	$N$	min, max, median, Burrows-Wheeler transform, ...
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, ...
quadratic	$N^2$	?
$\vdots$	$\vdots$	$\vdots$
exponential	$c^N$	?

**Frustrating news.** Huge number of problems have defied classification.

## Birds-eye view: revised

---

**Desiderata.** Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	$N$	min, max, median,
linearithmic	$N \log N$	sorting, convex hull,
$M(N)$	?	integer multiplication, division, square root, ...
$MM(N)$	?	matrix multiplication, $Ax = b$ , least square, determinant, ...
$\vdots$	$\vdots$	$\vdots$
NP-complete	probably not $N^b$	SAT, IND-SET, ILP, ...

↑  
STAY TUNED!

**Good news.** Can put many problems into equivalence classes.

**Complexity class.** Set of problems sharing some computational property.



37

# Summary

---

## Reductions are important in theory to:

- Design algorithms.
- Establish lower bounds.
- Classify problems according to their computational requirements.

## Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
  - stacks, queues, priority queues, symbol tables, sets, graphs
  - sorting, regular expressions, Delaunay triangulation
  - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.



## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*



<http://algs4.cs.princeton.edu>

## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*