



## 6.5 REDUCTIONS

---

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

# Overview: introduction to advanced topics

---

## Main topics. [next 3 lectures] {NOT for COMP 582}

- 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 implementation details to conceptual frameworks.



## Goals.

- Place algorithms and techniques 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*
- ▶ *intractability*

# Bird's-eye view

---

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

complexity	order of growth	examples
<b>linear</b>	$N$	<i>min, max, median, Burrows-Wheeler transform, ...</i>
<b>linearithmic</b>	$N \log N$	<i>sorting, element distinctness, closest pair, Euclidean MST, ...</i>
<b>quadratic</b>	$N^2$	?
$\vdots$	$\vdots$	$\vdots$
<b>exponential</b>	$c^N$	?

**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?

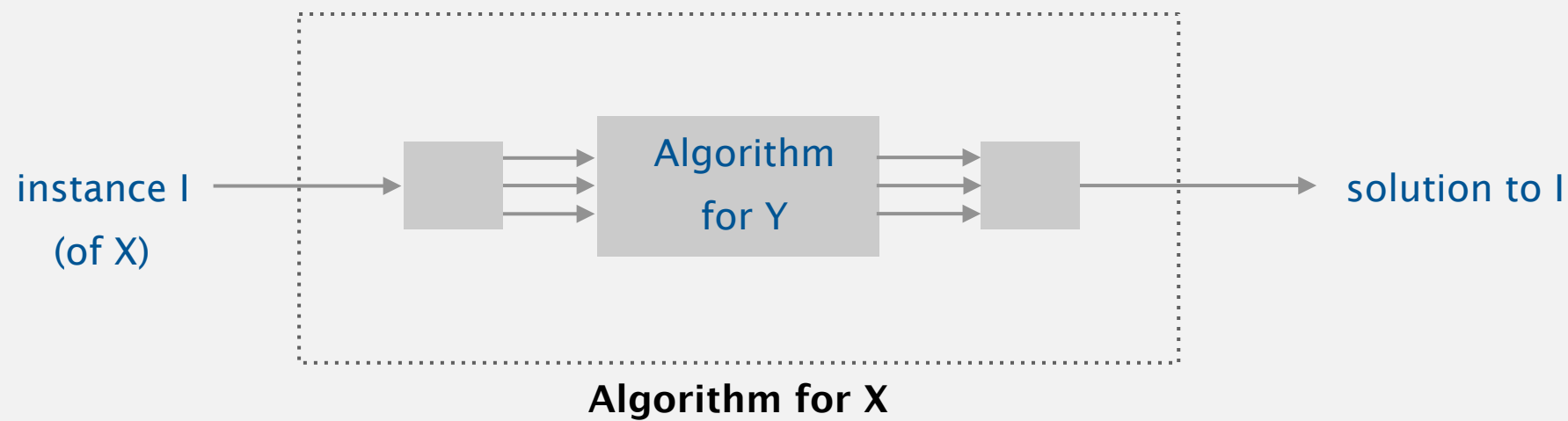


*“ Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. ” — Archimedes*

# 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.

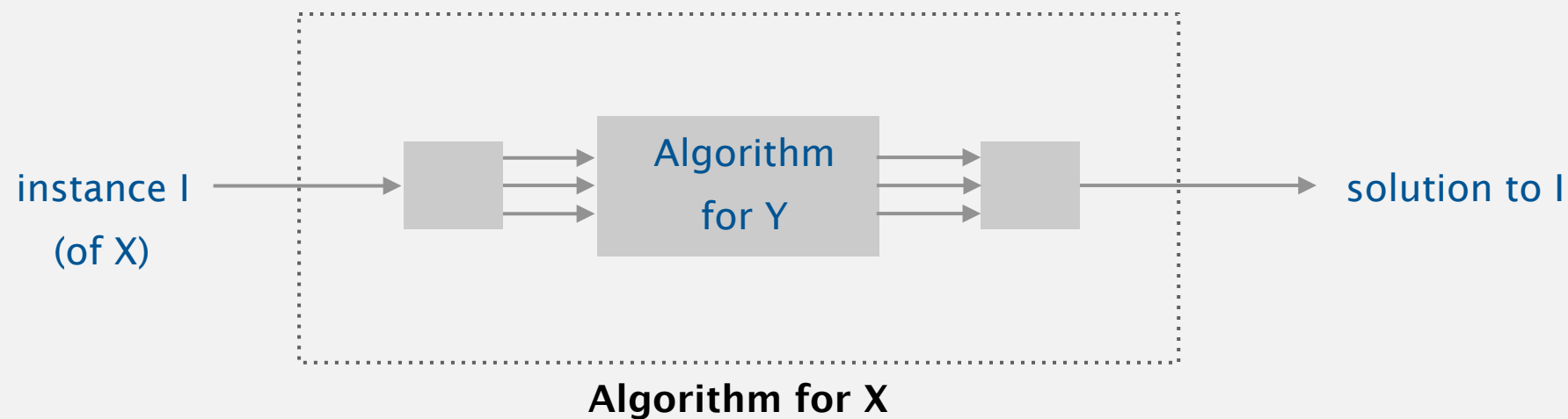
↑  
perhaps many calls to  $Y$   
on problems of different sizes  
(though, typically only one call)

↑  
preprocessing and postprocessing  
(typically less than cost of solving  $Y$ )

# 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.

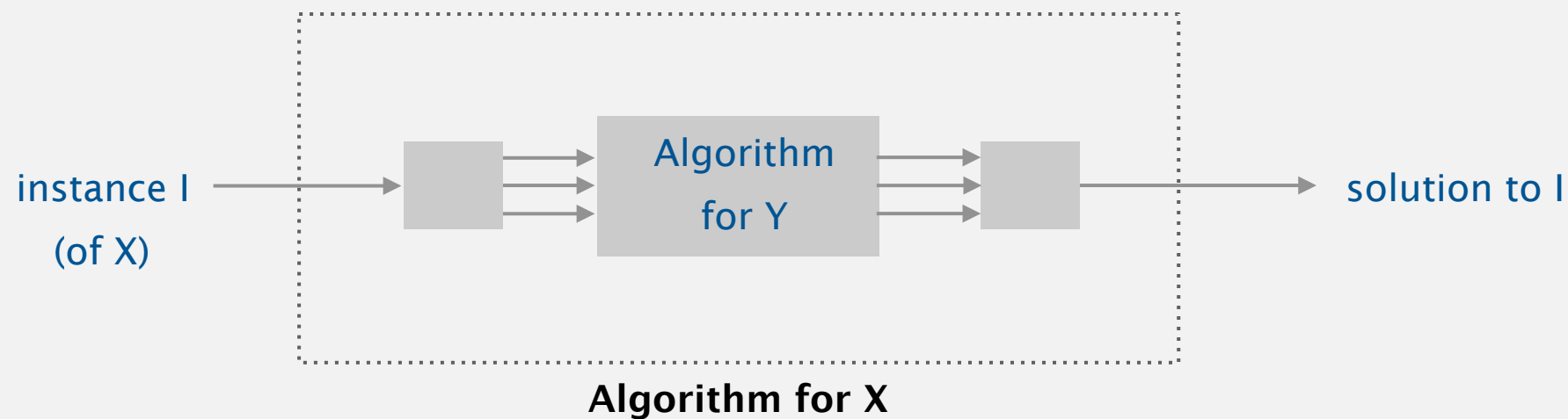
Cost of solving finding the median.  $N \log N + 1$ .

Two red arrows point to the terms in the formula: one from "cost of sorting" to  $N \log N$ , and another from "cost of reduction" to  $+ 1$ .

# 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* (arrow pointing to  $N \log N$ )      *cost of reduction* (arrow pointing to  $N$ )





## 6.5 REDUCTIONS

---

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

# 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  $X$ .

**More familiar reductions.**

- 3-collinear reduces to sorting
- Finding the median reduces to sorting
- Element distinctness reduces to sorting
- CPM reduces to topological sort. [shortest paths lecture]
- Arbitrage reduces to negative cycles. [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  $X$ ?



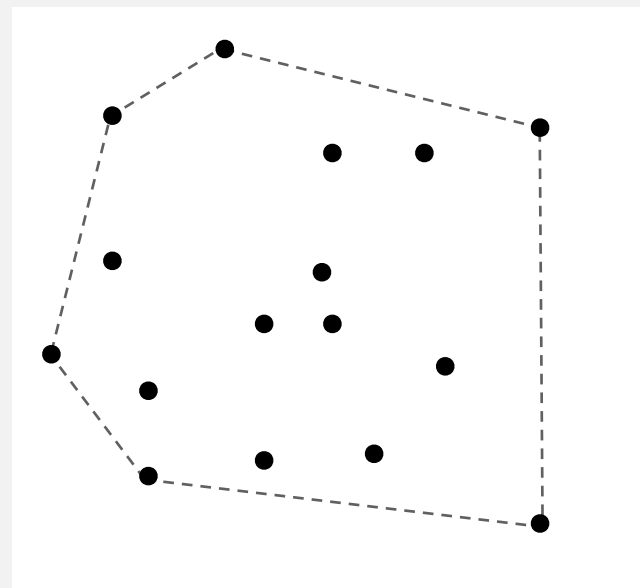
programmer's version: I have code for  $Y$ . Can I use it for  $X$ ?

# 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
8111033
13546464
89885444
43434213
34435312
```

sorting

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

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

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

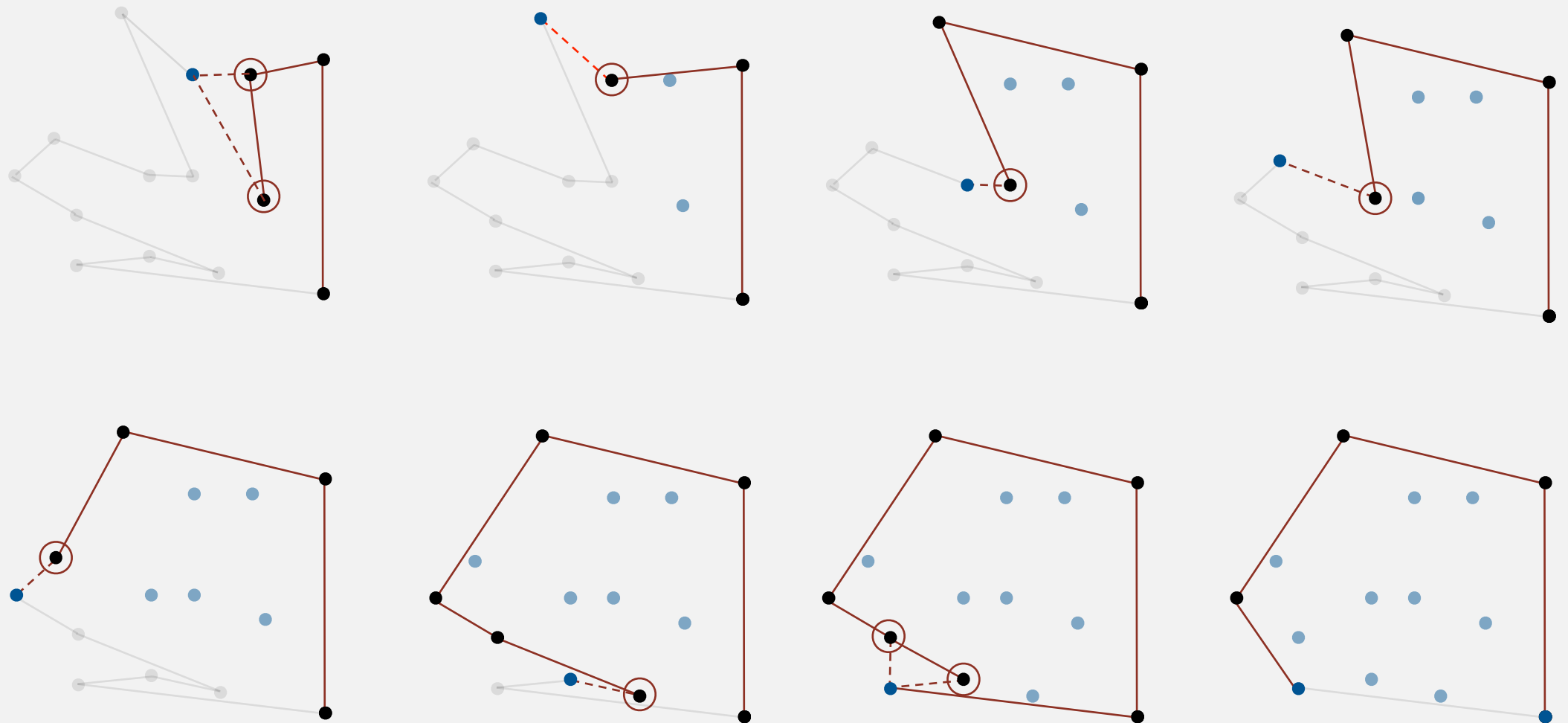
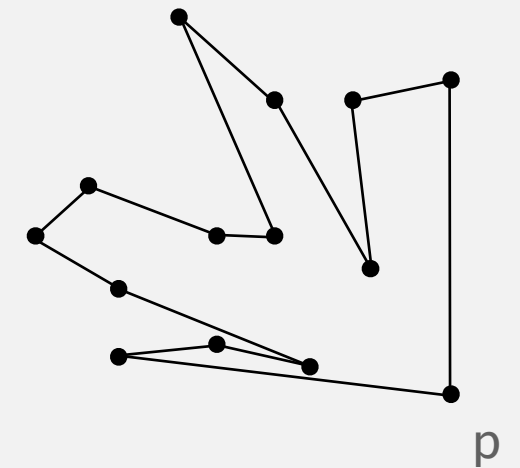
← cost of sorting

← cost of reduction

# Graham scan algorithm

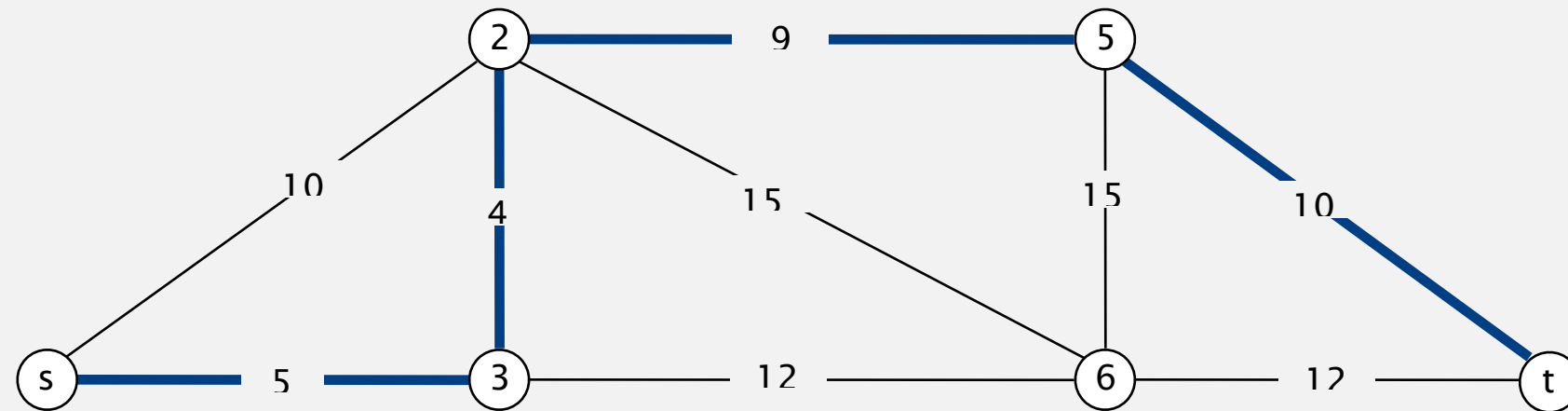
## Graham scan.

- 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.

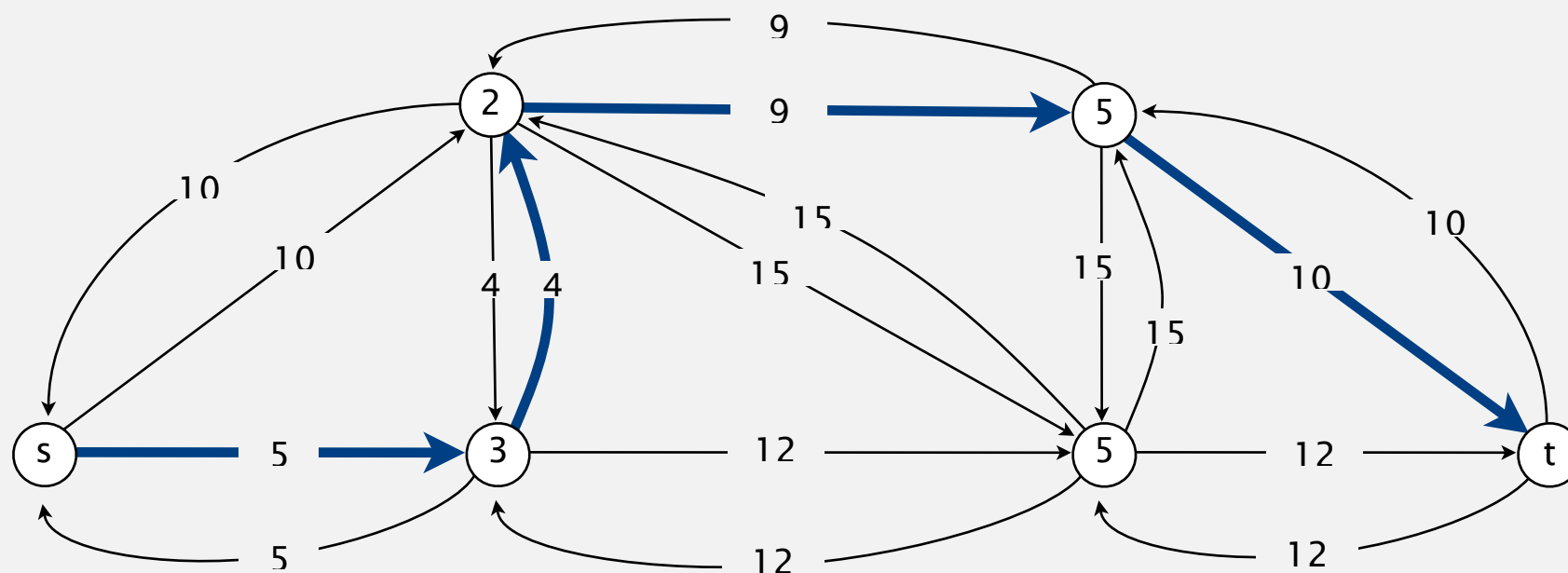


# Shortest paths on edge-weighted graphs and digraphs

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

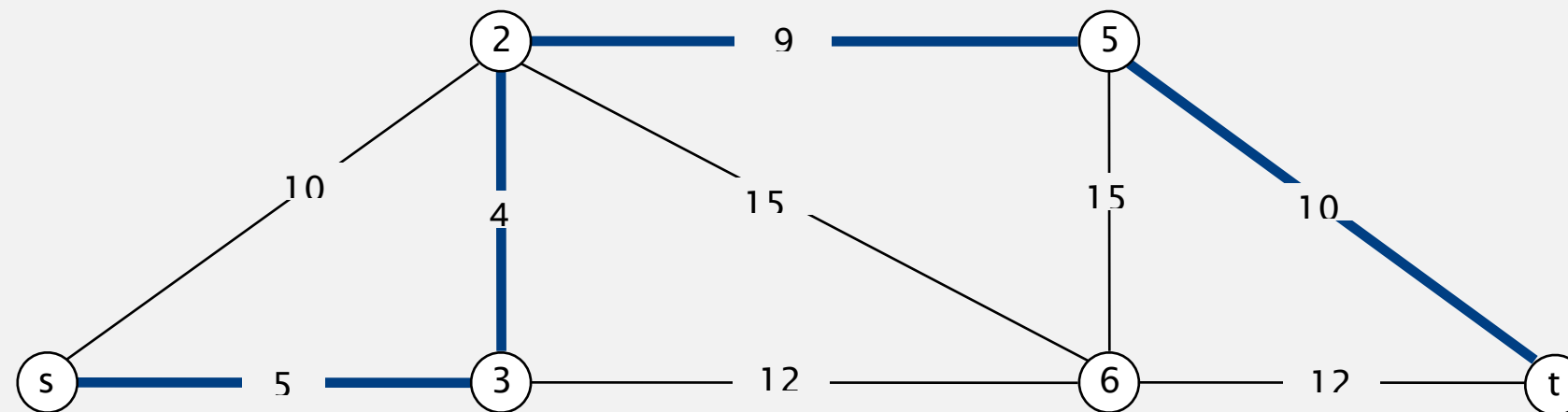


**Pf.** Replace each undirected edge by two directed edges.



# Shortest paths on edge-weighted graphs and digraphs

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



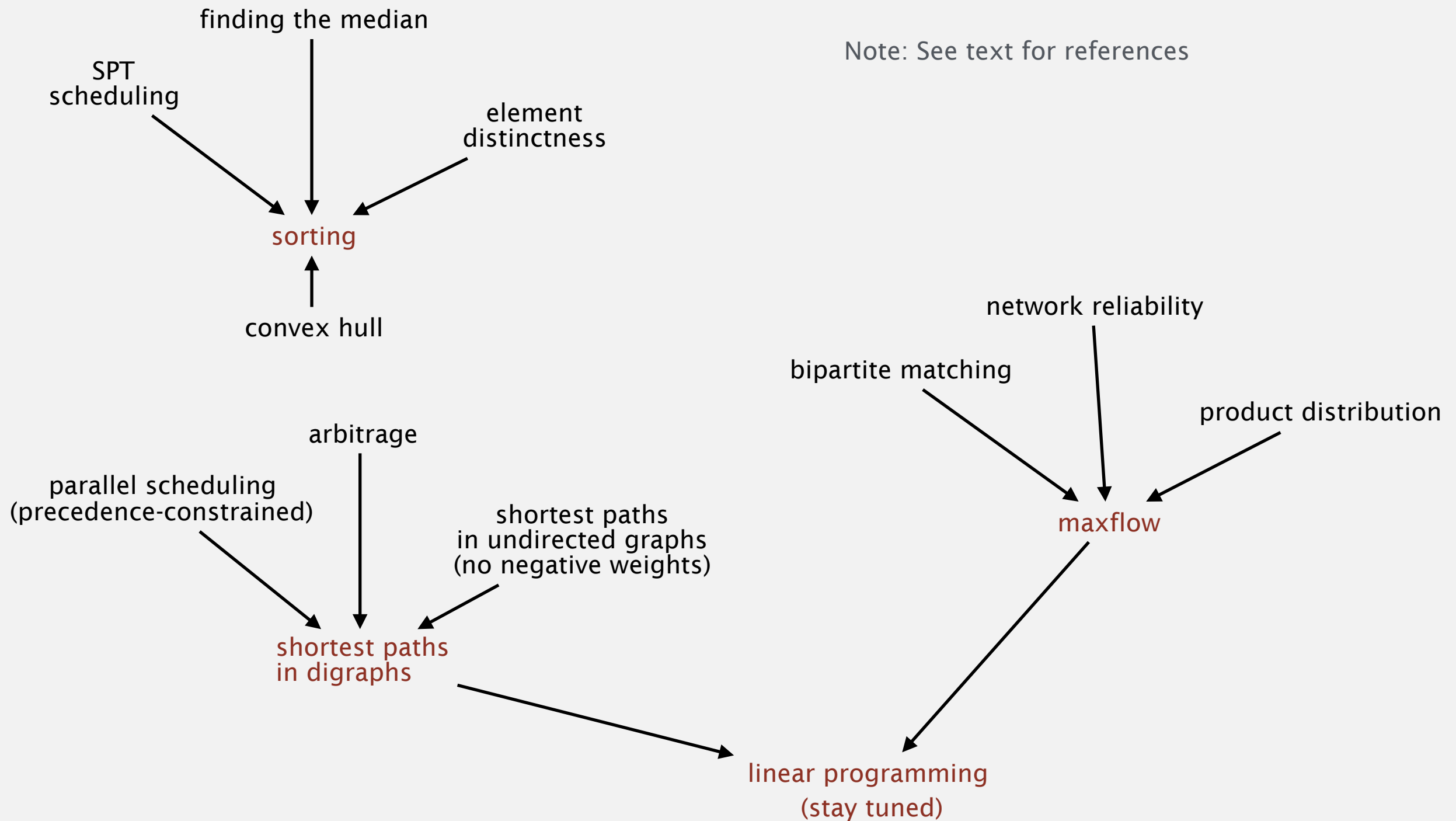
cost of shortest  
paths in digraph

cost of reduction

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

# Linear-time reductions involving familiar problems

---





## 6.5 REDUCTIONS

---

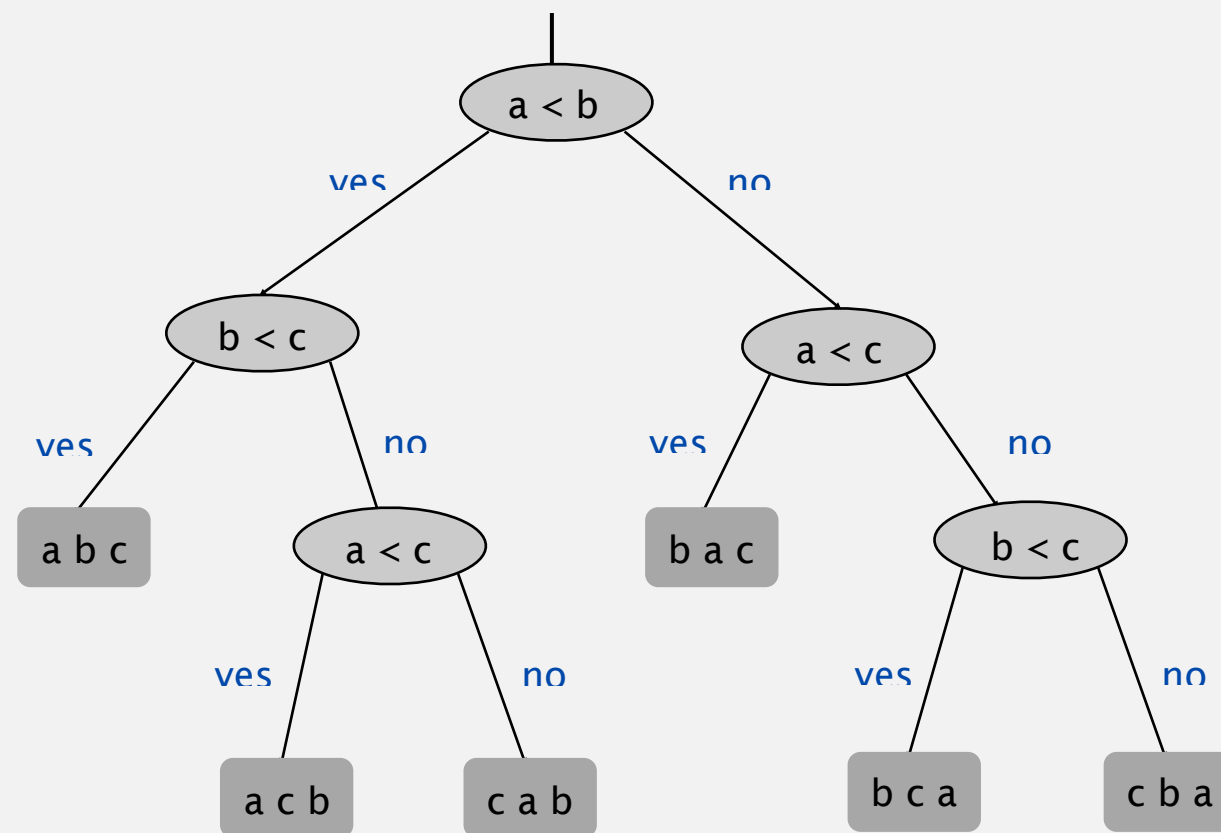
- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*
- ▶ *intractability*



# 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:

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

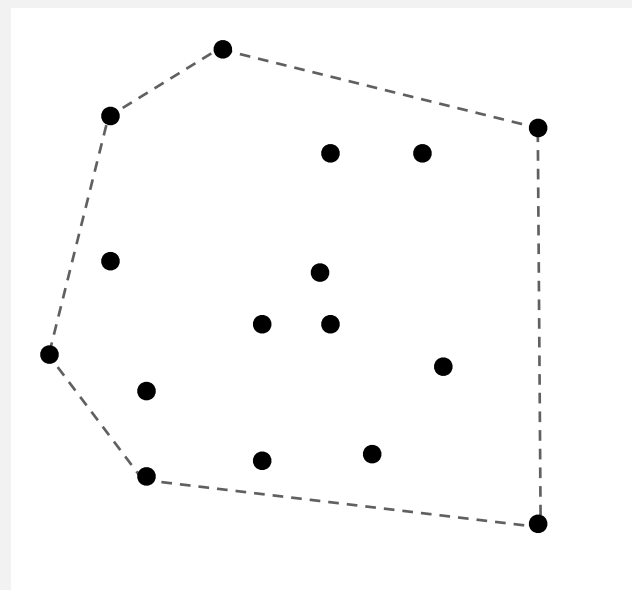
**Proposition.** Sorting linear-time reduces to convex hull.

**Pf.** [see next slide]

lower-bound mentality:  
I can't sort in linear time,  
so I can't solve convex hull  
in linear time either

1251432  
2861534  
3988818  
4190745  
8111033  
13546464  
89885444  
43434213  
34435312

sorting



convex hull

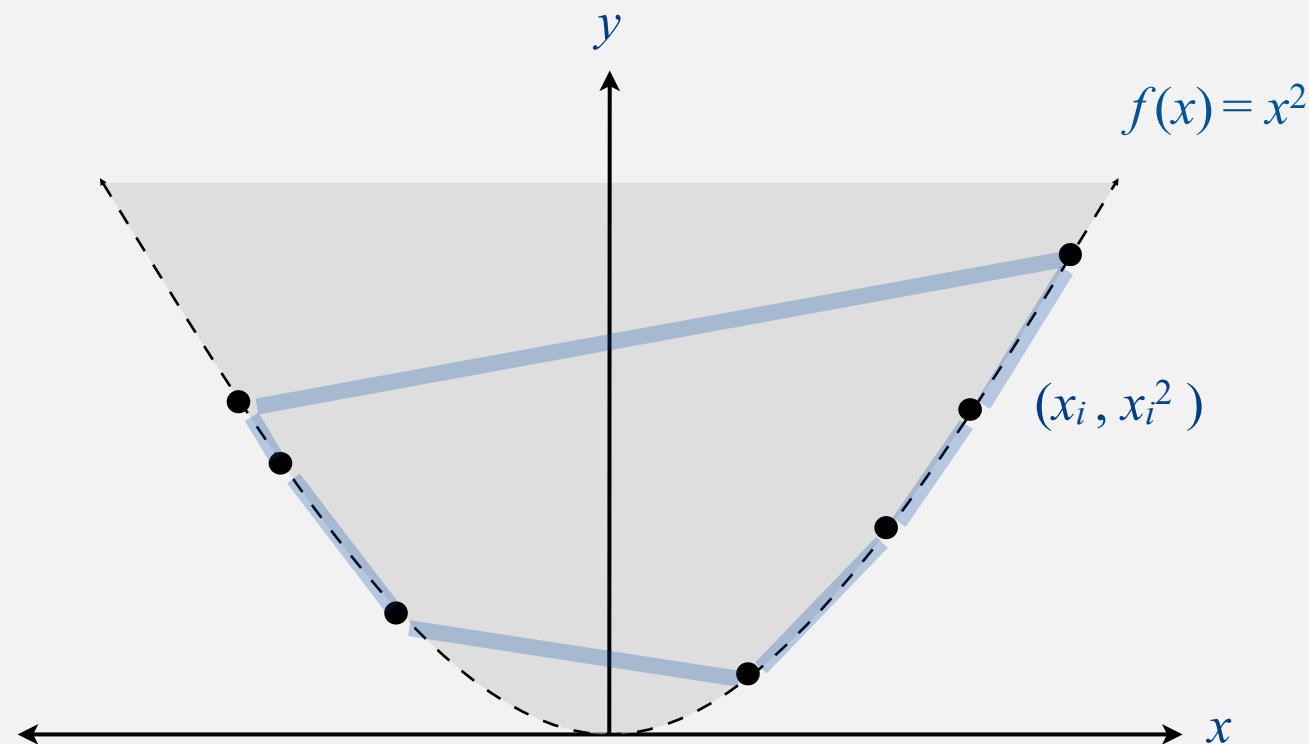
linear or  
quadratic tests

**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:  $(x_1, x_1^2), (x_2, x_2^2), \dots, (x_N, x_N^2)$ .



**Pf.**

- Region  $\{x : x^2 \geq x\}$  is convex  $\Rightarrow$  all  $N$  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

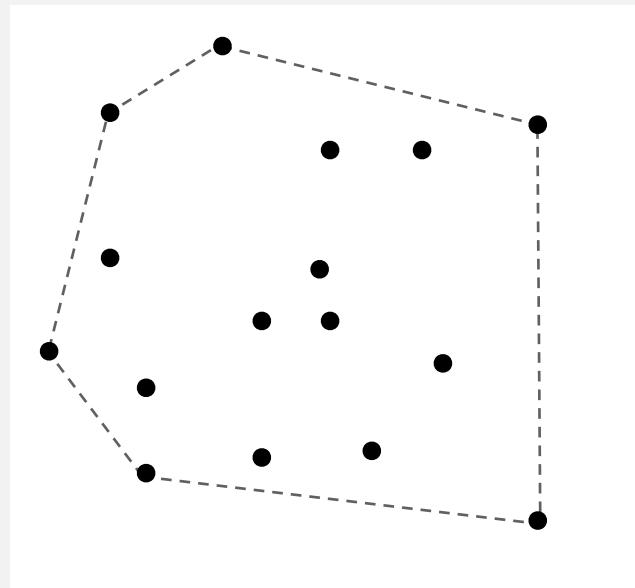
---

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*
- ▶ *intractability*

# Classifying problems: summary

---

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

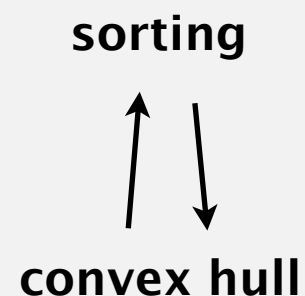
**Ex.** Sorting and element distinctness have complexity  $N \log N$ .

**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  $X$  has complexity  $N^b$  iff  $Y$  has complexity  $N^b$  for  $b \geq 1$ .

even if we don't know what it is



# Caveat

---

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

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

**Proposition.** INSERTION SORT linear-time reduces to CONVEX HULL.


**Proposition.** CONVEX HULL linear-time reduces to INSERTION SORT.

**Conclusion.** INSERTION 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





25

# Integer arithmetic reductions

---

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

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

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

**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
?	<b>brute force</b>	$N^2$
1962	<b>Karatsuba</b>	$N^{1.585}$
1963	<b>Toom-3, Toom-4</b>	$N^{1.465}$ , $N^{1.404}$
1966	<b>Toom-Cook</b>	$N^{1+\varepsilon}$
1971	<b>Schönhage-Strassen</b>	$N \log N \log \log N$
2007	<b>Fürer</b>	$N \log N 2^{\log^* N}$
?	?	$N$

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.



# 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
<b>matrix multiplication</b>	$A \times B$	$MM(N)$
<b>matrix inversion</b>	$A^{-1}$	$MM(N)$
<b>determinant</b>	$ A $	$MM(N)$
<b>system of linear equations</b>	$Ax = b$	$MM(N)$
<b>LU decomposition</b>	$A = LU$	$MM(N)$
<b>least squares</b>	$\min \ Ax - b\ _2$	$MM(N)$

**numerical linear algebra problems with the same complexity as matrix multiplication**

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

# Complexity of matrix multiplication

---

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

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

# Bird's-eye view:review

---

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

complexity	order of growth	examples
<b>linear</b>	$N$	<i>min, max, median, Burrows-Wheeler transform, ...</i>
<b>linearithmic</b>	$N \log N$	<i>sorting, element distinctness, closest pair, Euclidean MST, ...</i>
<b>quadratic</b>	$N^2$	?
$\vdots$	$\vdots$	$\vdots$
<b>exponential</b>	$c^N$	?

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

# Bird's-eye view: revised

---

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

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



<https://complexityzoo.uwaterloo.ca>

**Bad news.** Lots of complexity classes (496 animals in zoo).

# 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
  - use exact algorithm for tractable problems
  - use heuristics or intractable problems