



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

## 6.5 REDUCTIONS

---

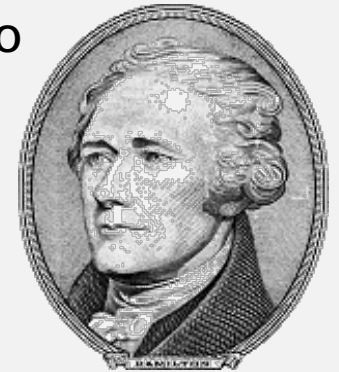
- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*
- ▶ *intractability*

## Exercise

---

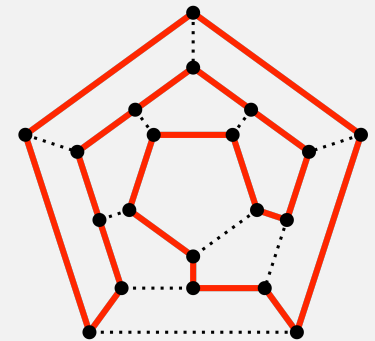
Imagine that founding father Alexander Hamilton\* has offered to find a **Hamiltonian cycle** in any given graph (if one exists).

Design an efficient algorithm to find a **Hamiltonian path** in a graph (if one exists) by making queries to Hamilton.



**Solution.** Given graph  $G = (V, E)$ :

- Query Hamilton with  $G$ .
  - If cycle found, remove last vertex and return rest.
- For every *nonexistent* edge  $e \notin E$ :
  - Query Hamilton with  $(V, E \cup \{e\})$
  - If cycle found, “rotate” it so that final two vertices are incident on  $e$ ; remove final vertex and return the rest.
- Return “no Hamiltonian path”.



The cycle *must* traverse  $e$ . Why?

Why is this correct?

\*Hamiltonian cycle/path are named after William Rowan Hamilton.

# Reductions: overview

---

## Main topics.

- Reduction: relationship between two problems.
- Algorithm design: paradigms for solving problems.

## 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!

## Reductions: practical tip

---

Reductions require ingenuity, but a few tricks recur. Practice them.



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

## 6.5 REDUCTIONS

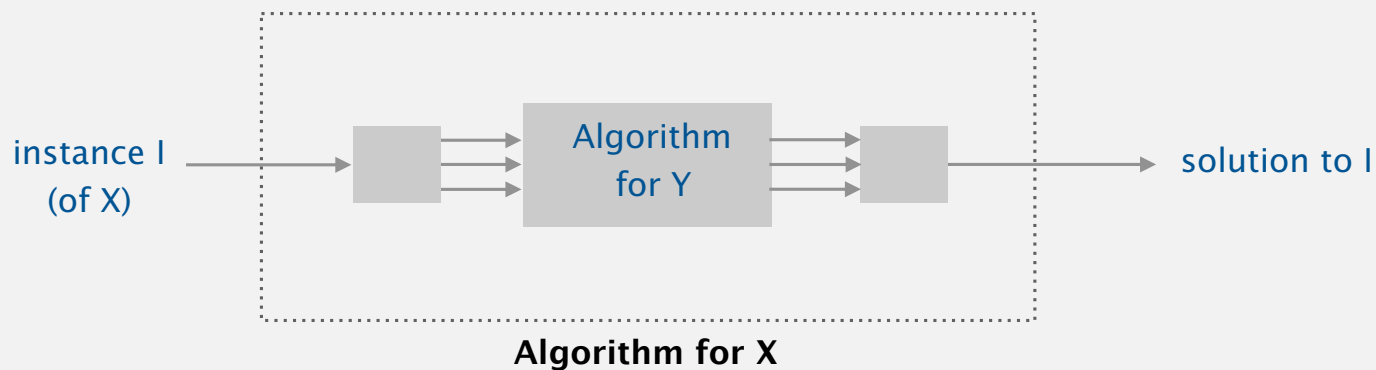
---

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

# 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.

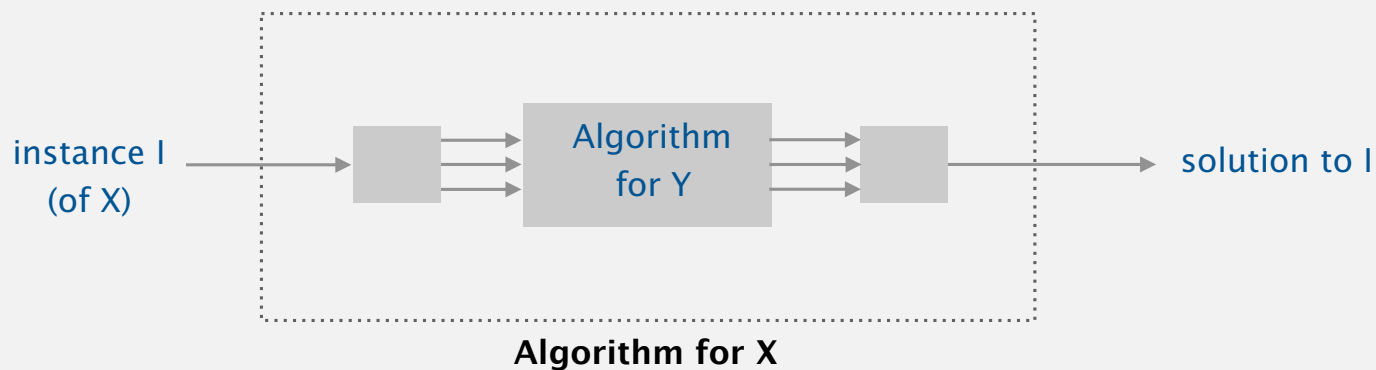
↑  
perhaps many calls to  $Y$   
on problems of different sizes  
(typically only 1 call)

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

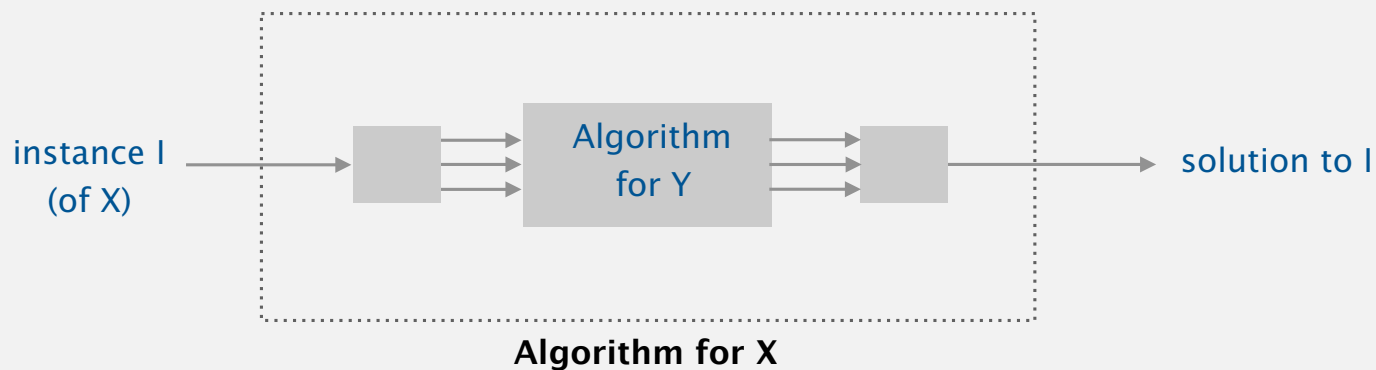
Cost of finding the median.  $N \log N + 1$ .

*cost of sorting* (with arrow pointing to  $N \log N$ )  
*cost of reduction* (with arrow pointing 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 the  $N$  items.
- Check adjacent pairs for equality.

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

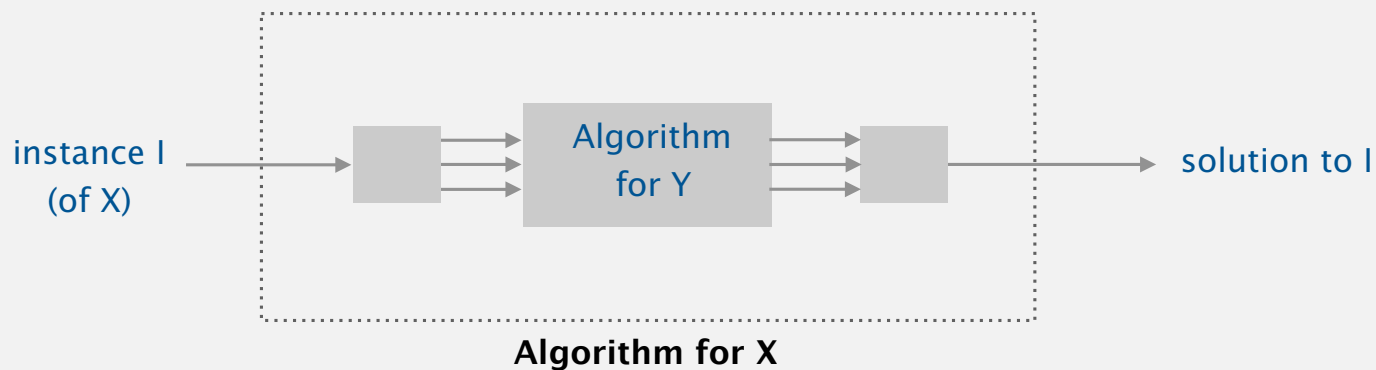
*cost of sorting* (with arrow pointing to  $N \log N$ )  
*cost of reduction* (with arrow pointing to  $N$ )



# Reduction

---

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



Confusing terminology. CS professors often slip up.

~~Novice error.~~ Confusing  $X$  reduces to  $Y$  with  $Y$  reduces to  $X$ .

# Reductions: quiz 1

---

Which of the following reductions have we encountered in this course?

- I. MAX-FLOW reduces to MIN-CUT.
  - II. MIN-CUT reduces to MAX-FLOW.
- need to find max st-flow and min st-cut  
(not simply compute the value)

- A. I only.
- B. II only.
- C. Both I and II.
- D. Neither I nor II.
- E. *I don't know.*



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

## 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 an algorithm for  $Y$ , can also solve  $X$ .

**More familiar reductions.**

- Mincut reduces to maxflow.
- Arbitrage reduces to negative cycles.
- Bipartite matching reduces to maxflow.
- Seam carving reduces to shortest paths in a DAG.
- Burrows-Wheeler transform reduces to suffix sort.
- ...

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



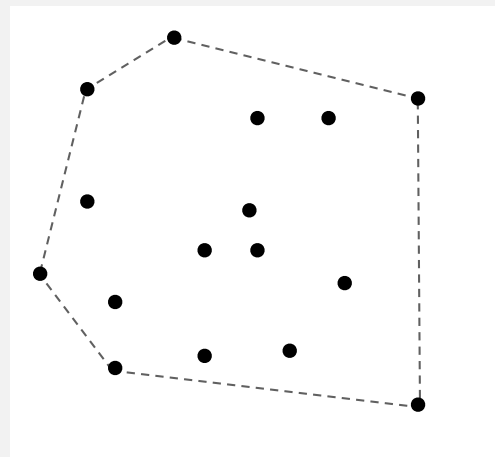
programmer's version: I have code for  $Y$ . Can I use it for  $X$ ?

# Convex hull reduces to sorting

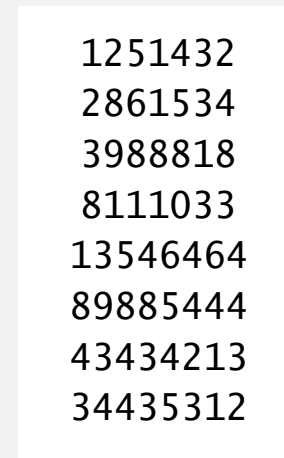
---

**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



sorting

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

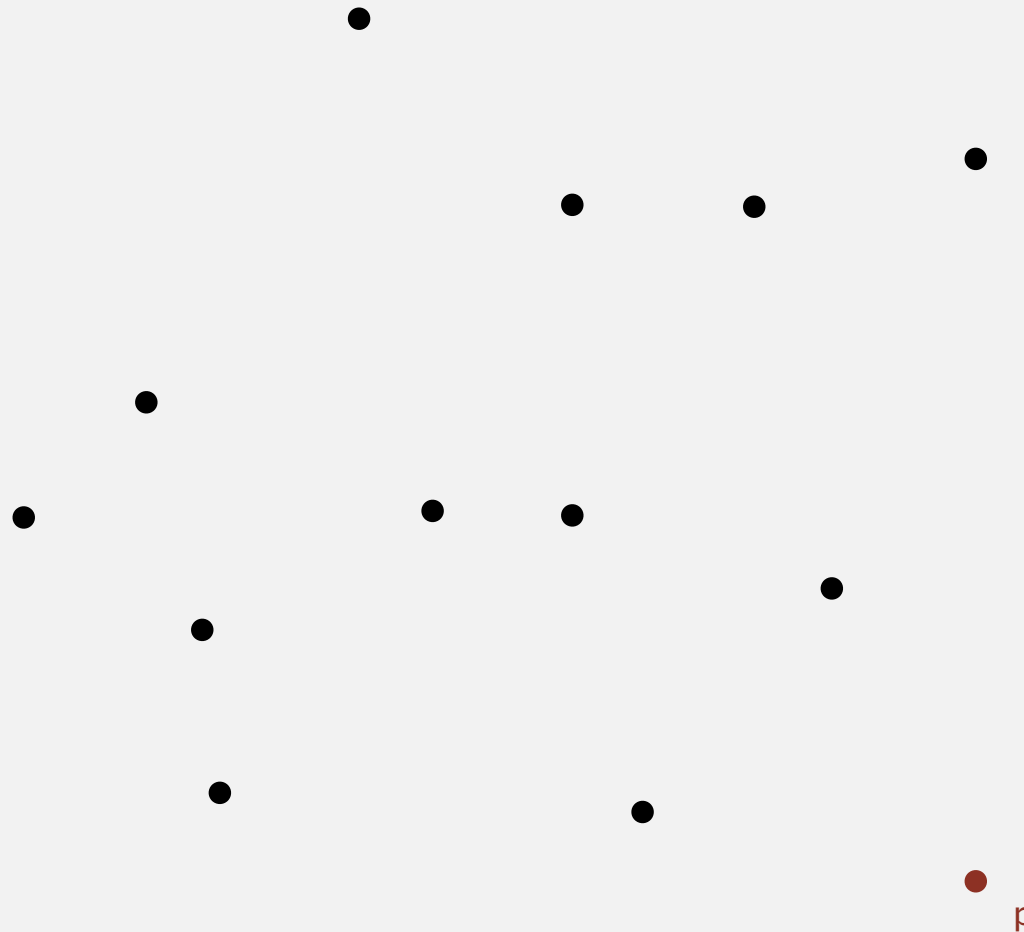
**Pf.** Graham scan algorithm.

**Cost of convex hull.**  $N \log N + N$ .  
← cost of sorting  
← cost of reduction

# Graham scan demo

---

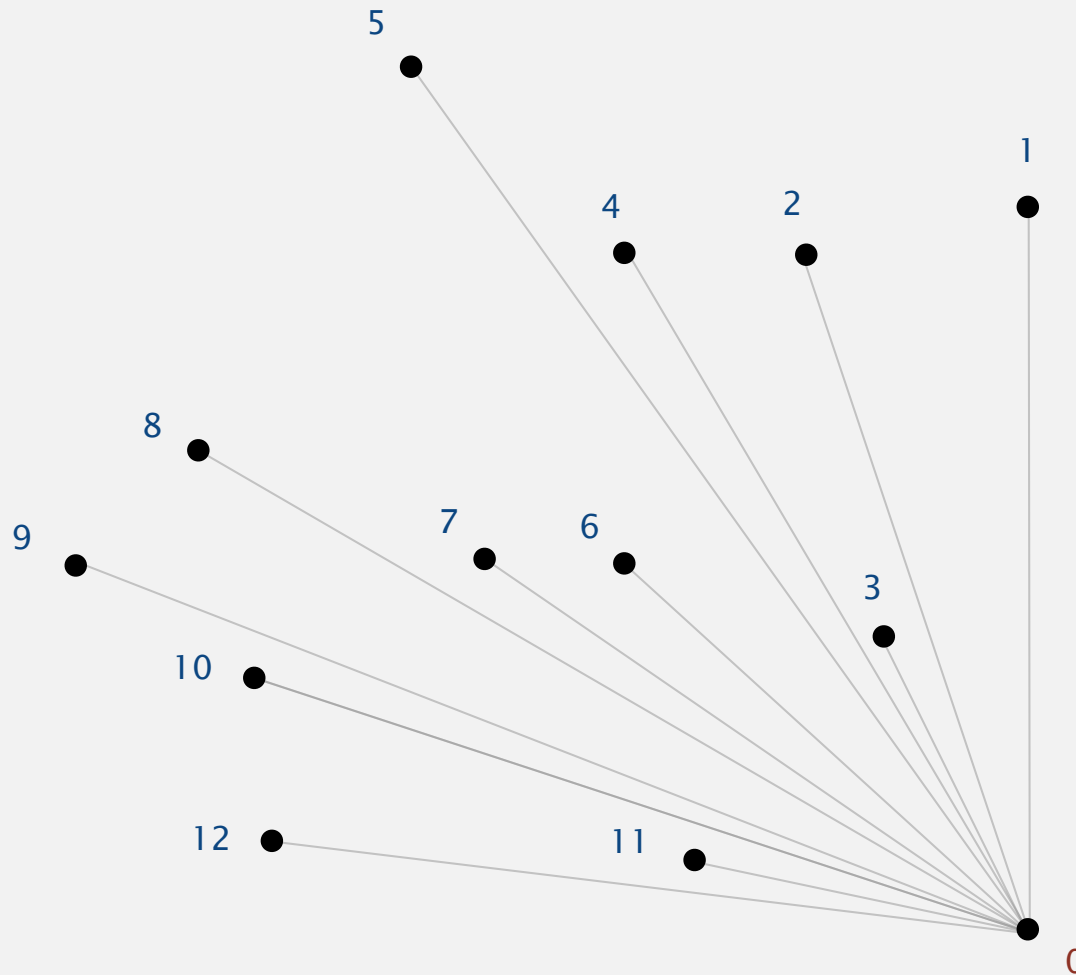
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

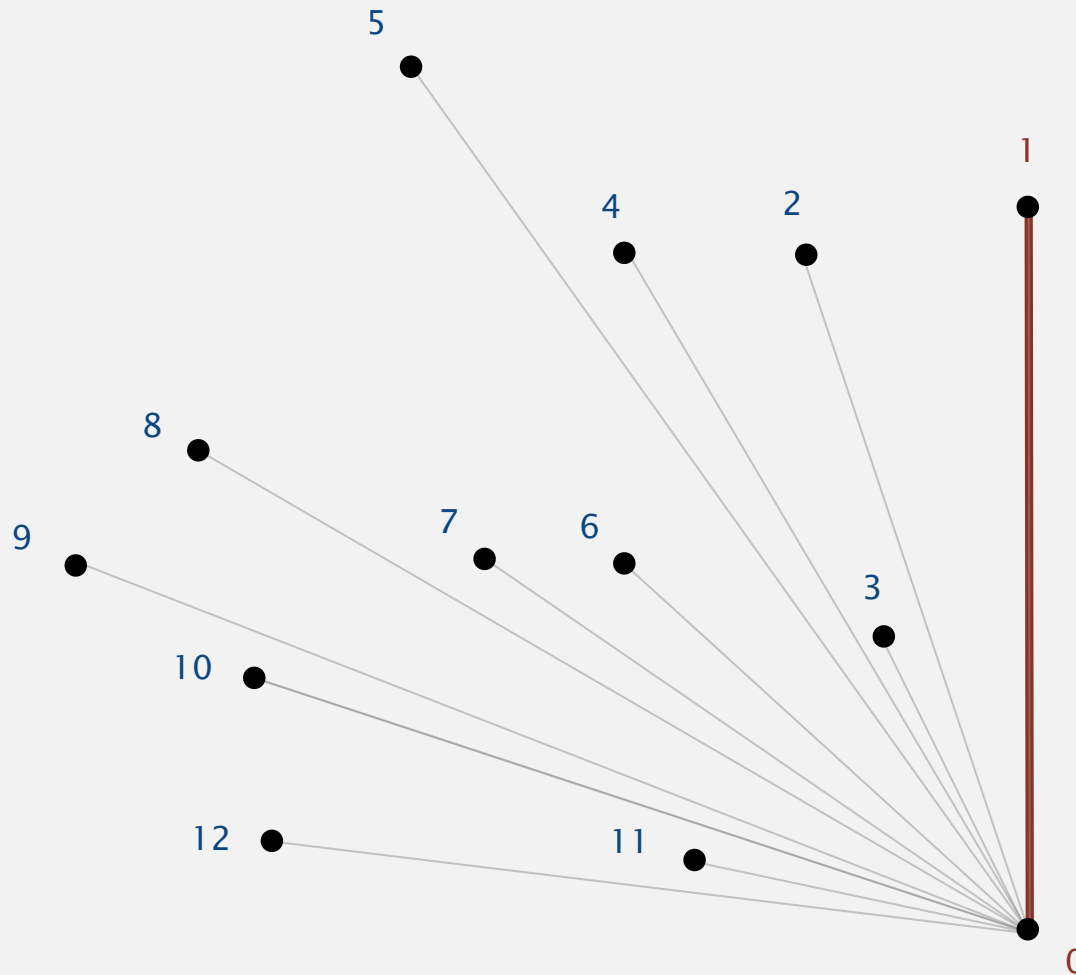
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.

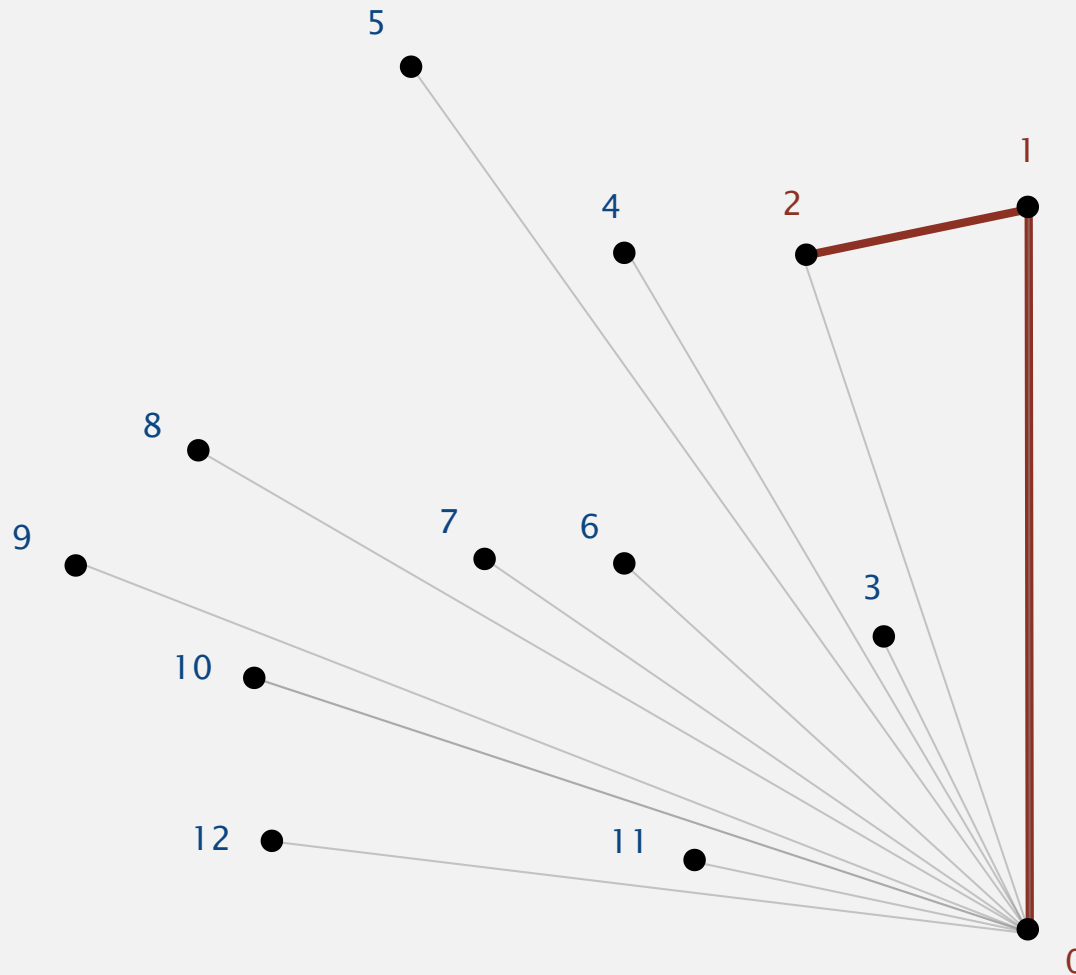




# Graham scan demo

---

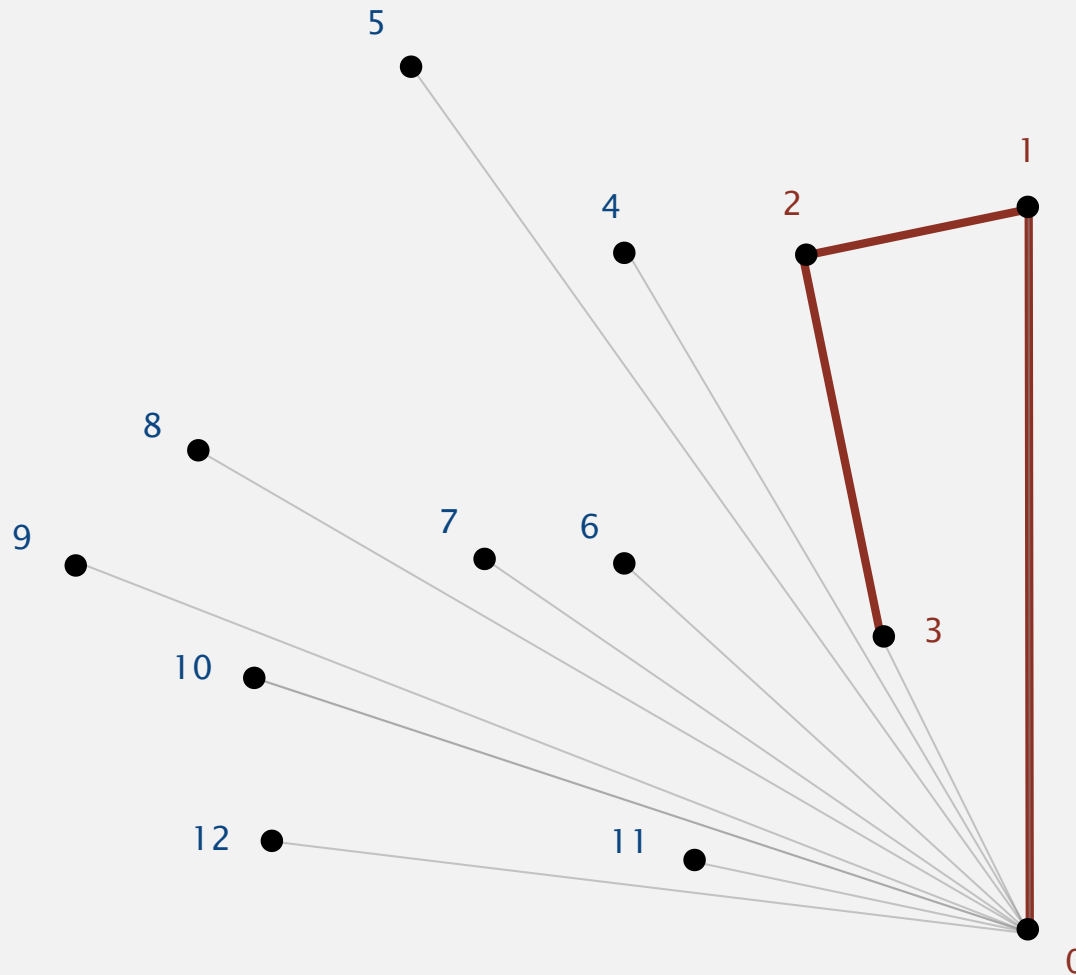
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

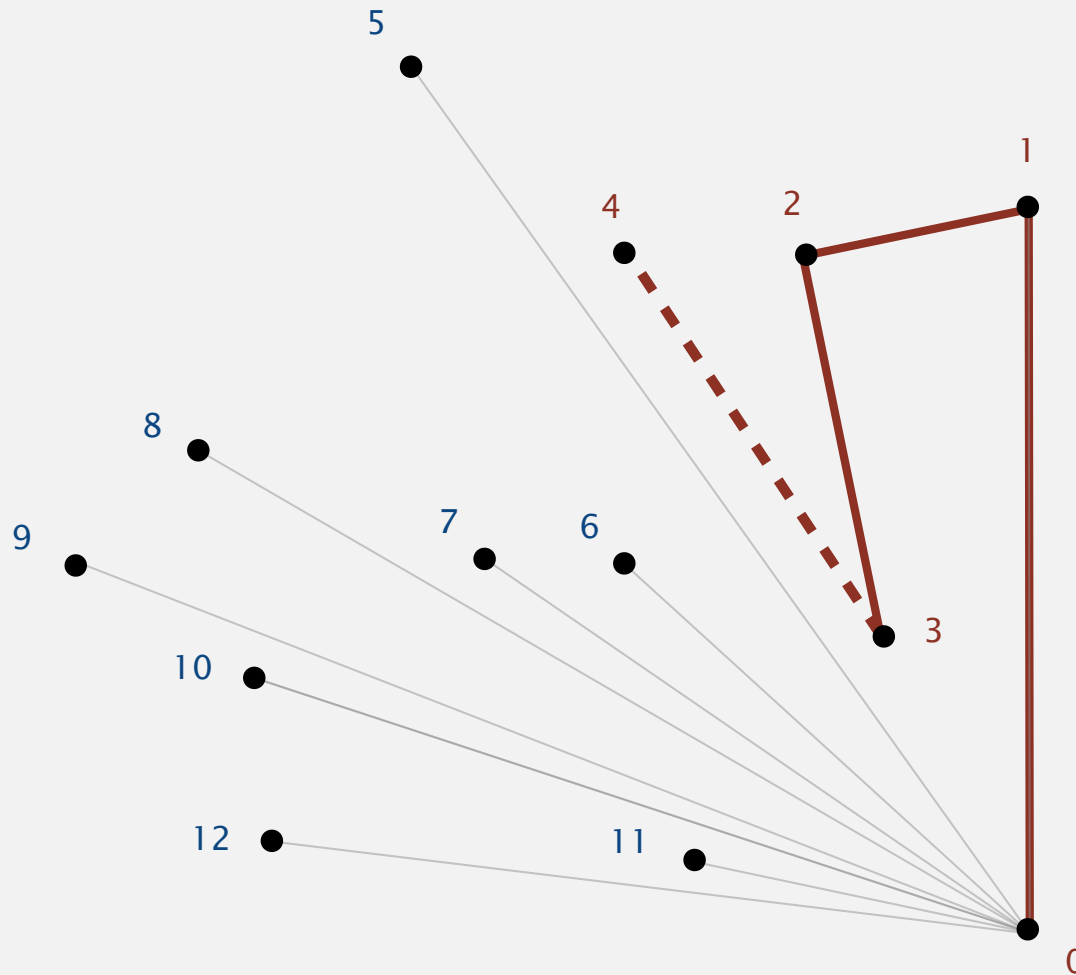
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

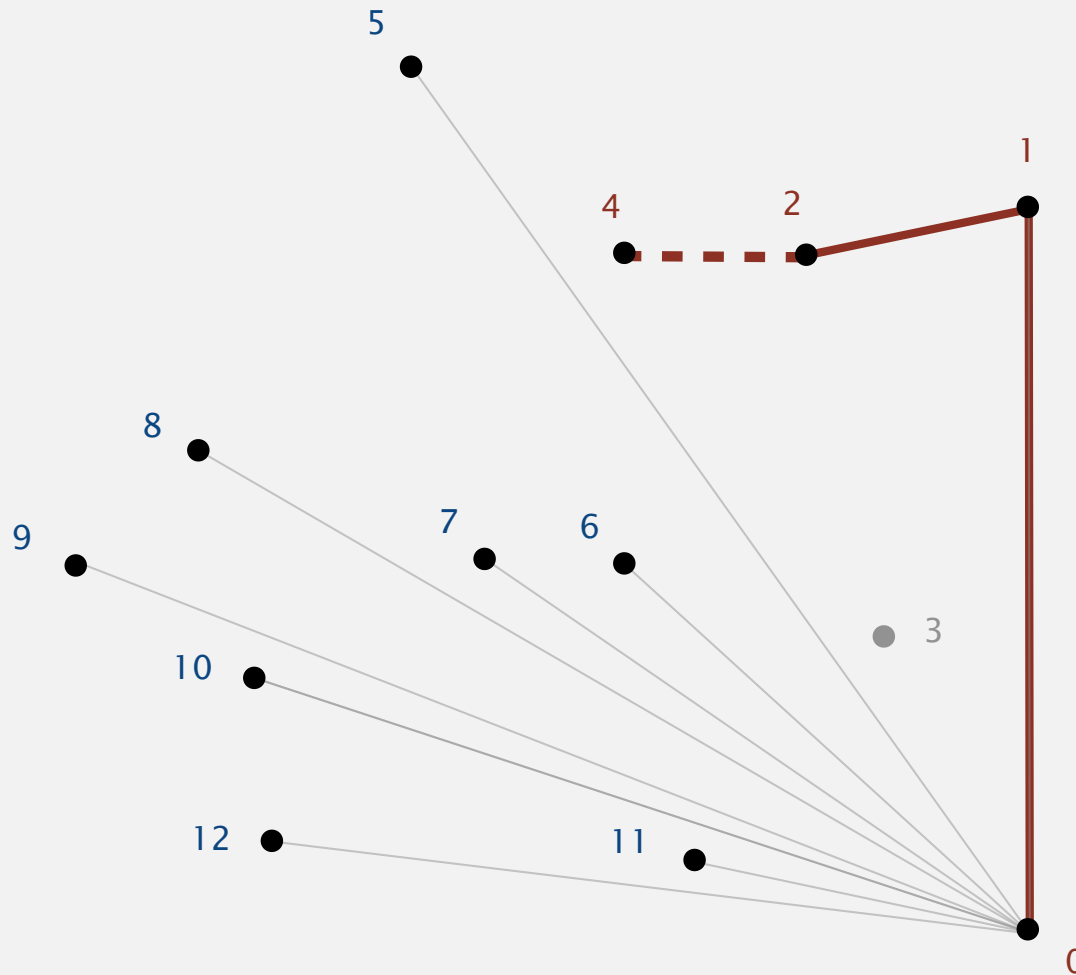
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

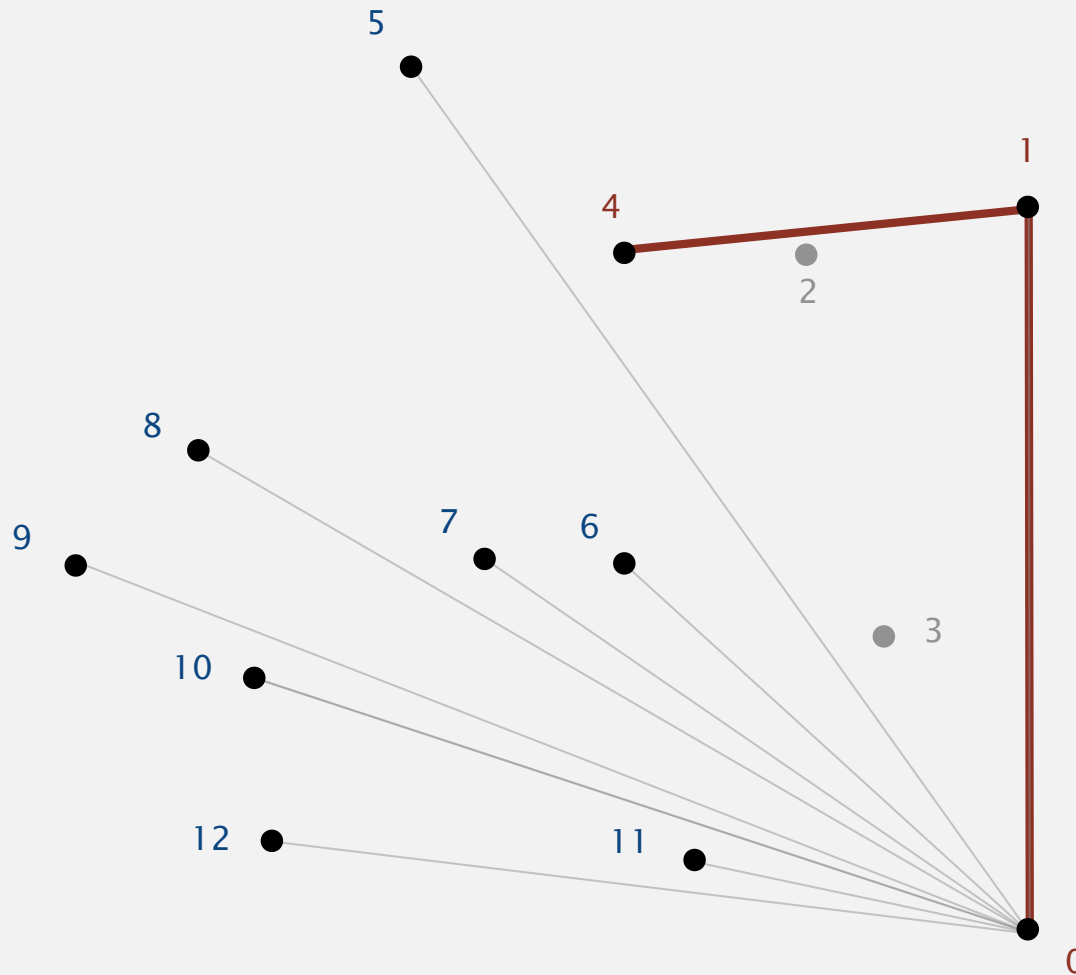
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

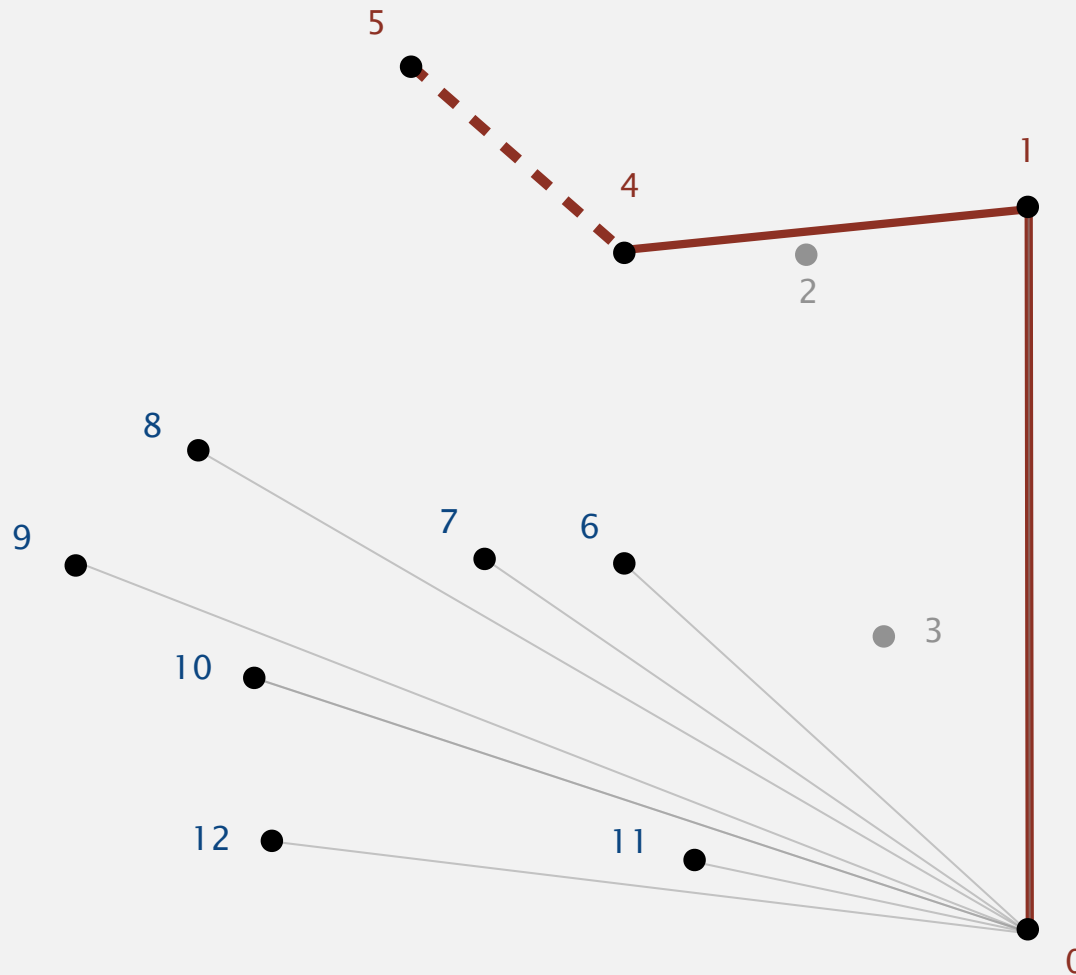
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

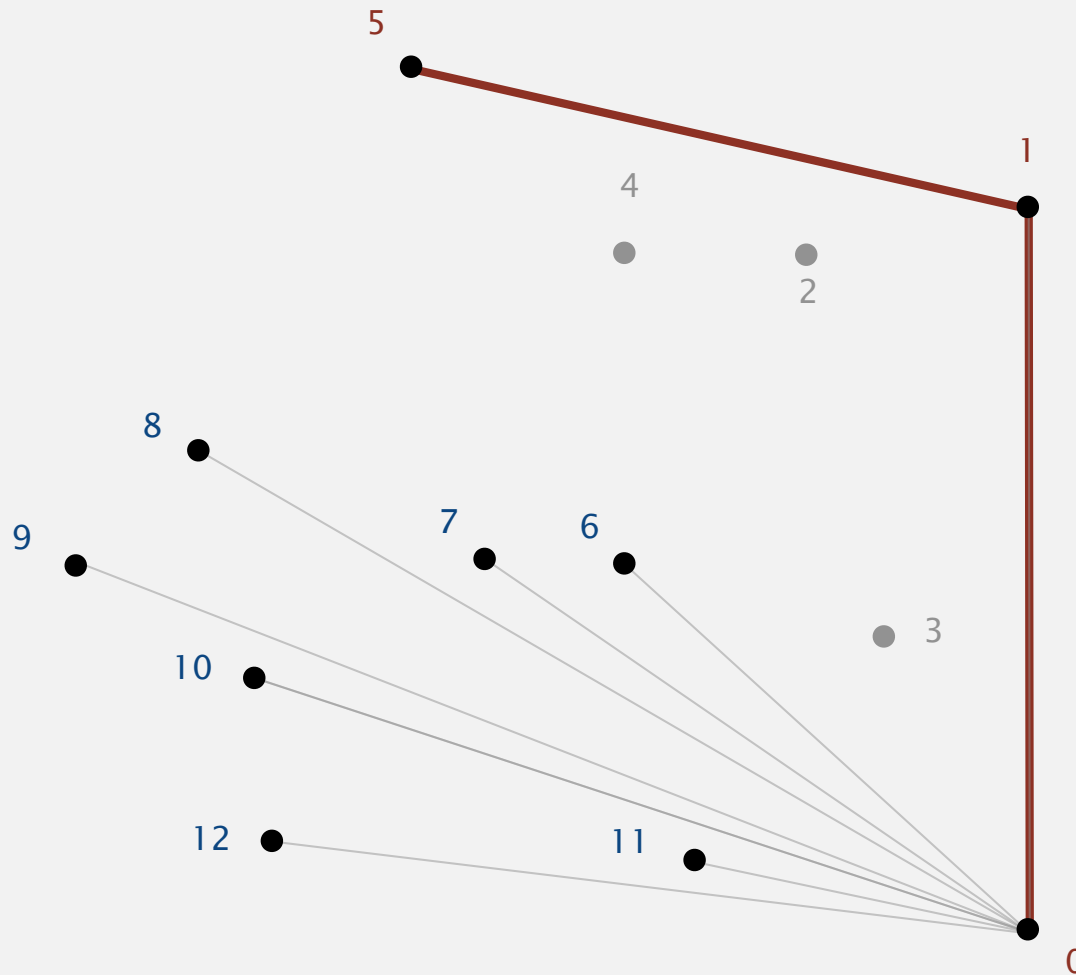
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

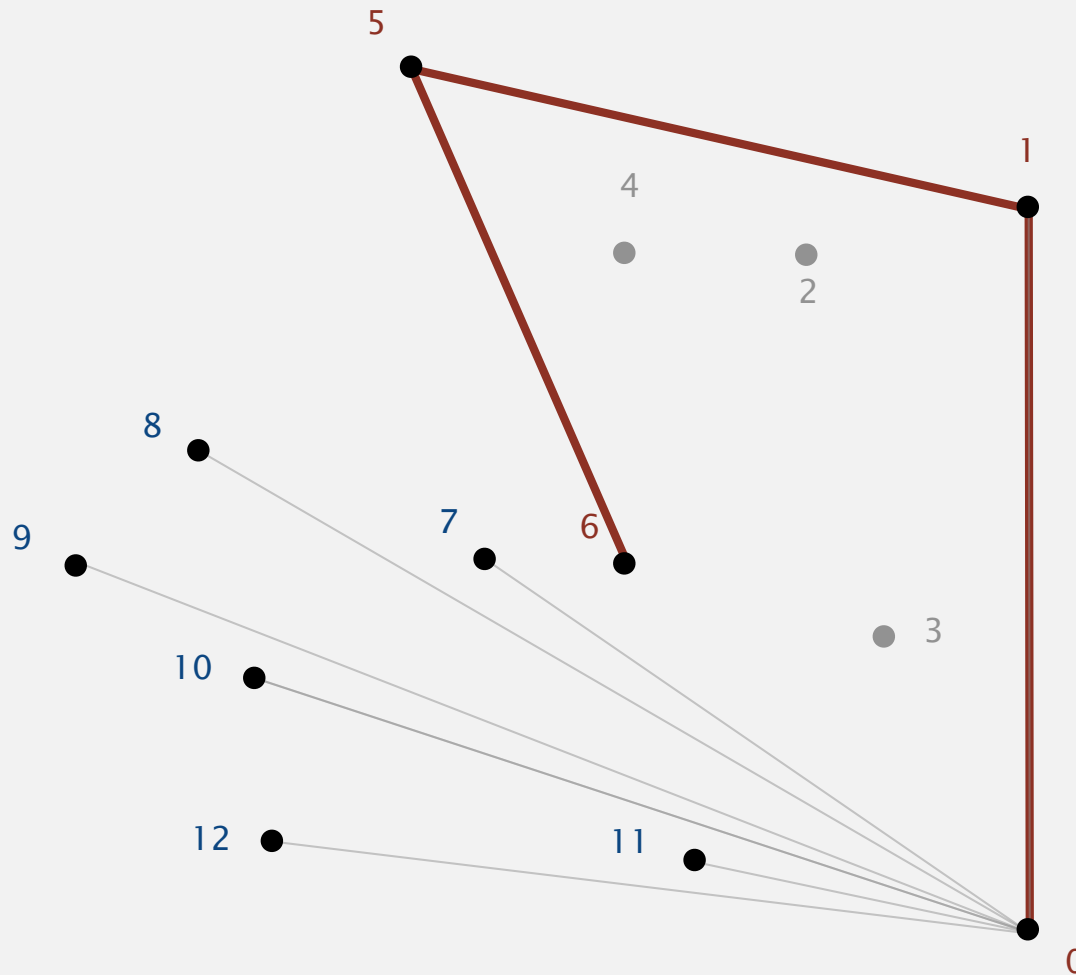
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.

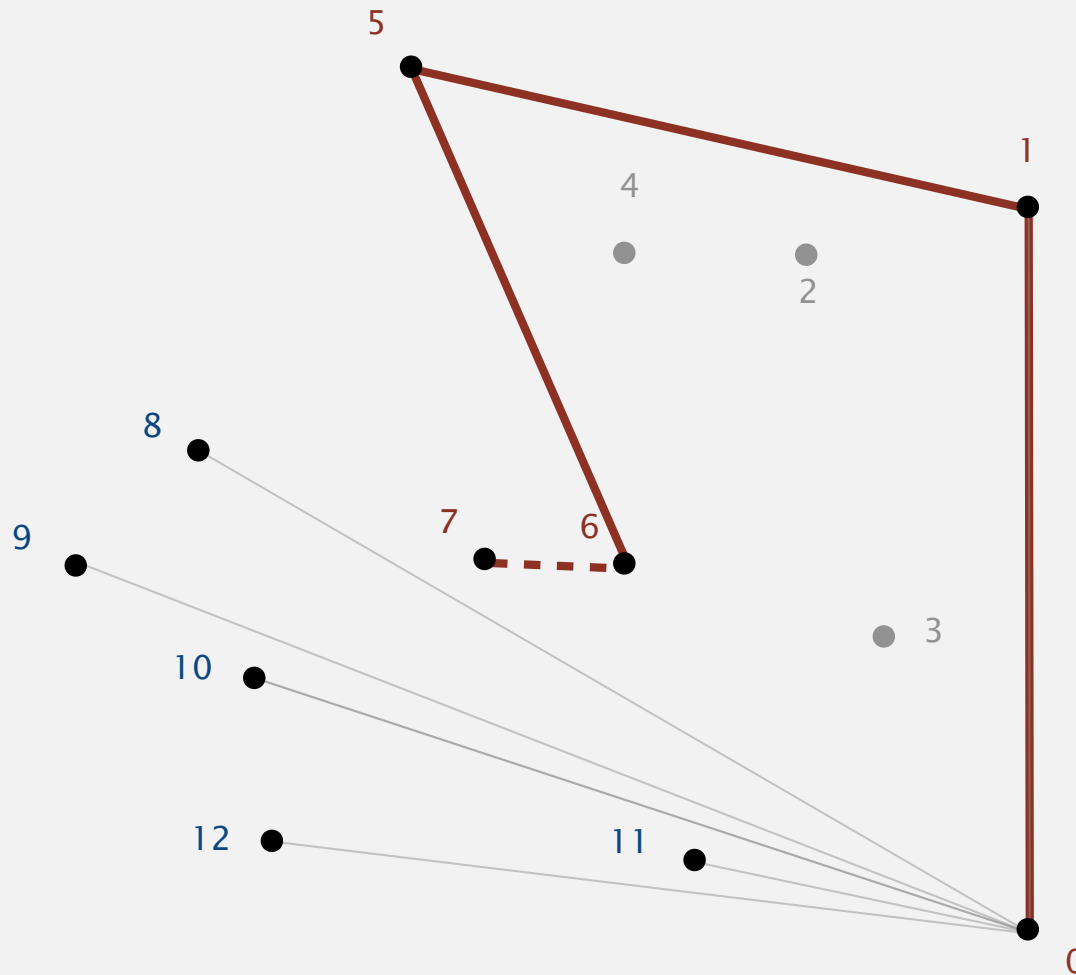




# Graham scan demo

---

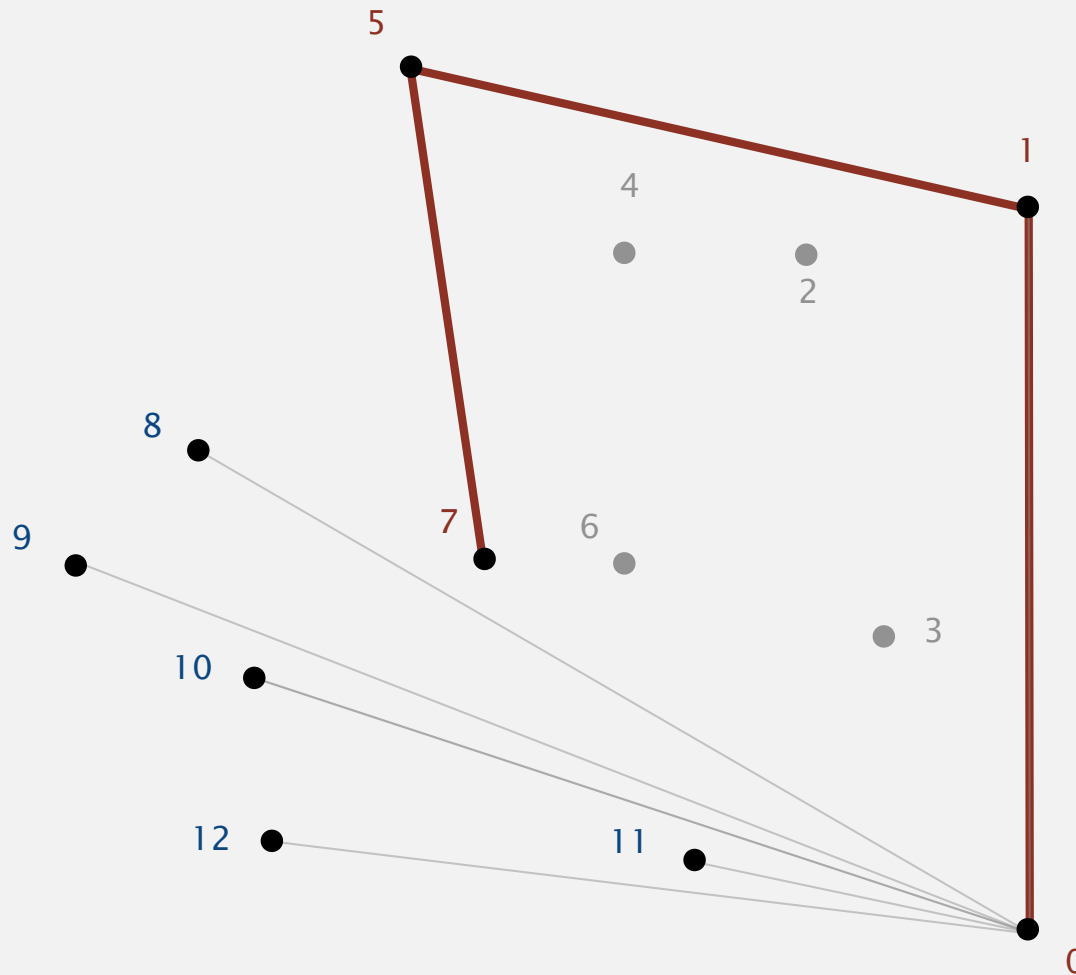
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

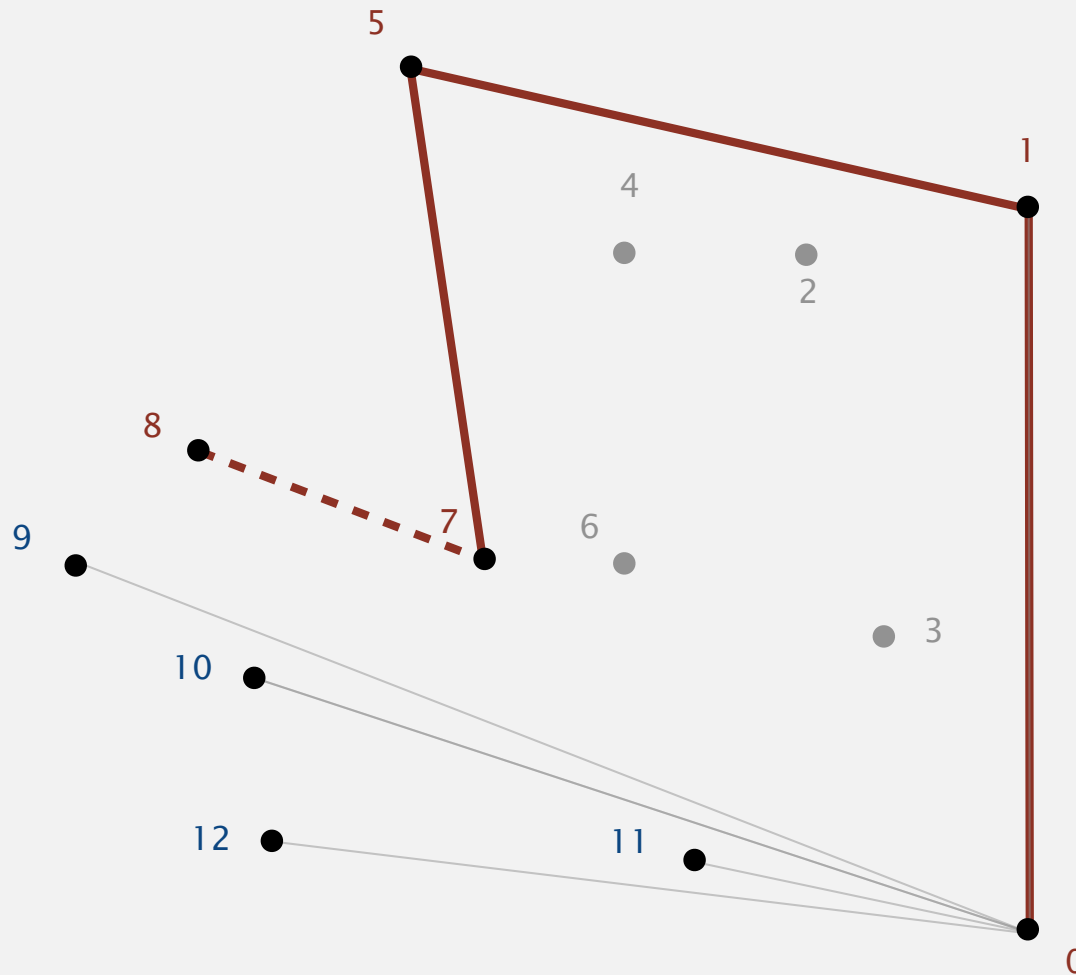
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

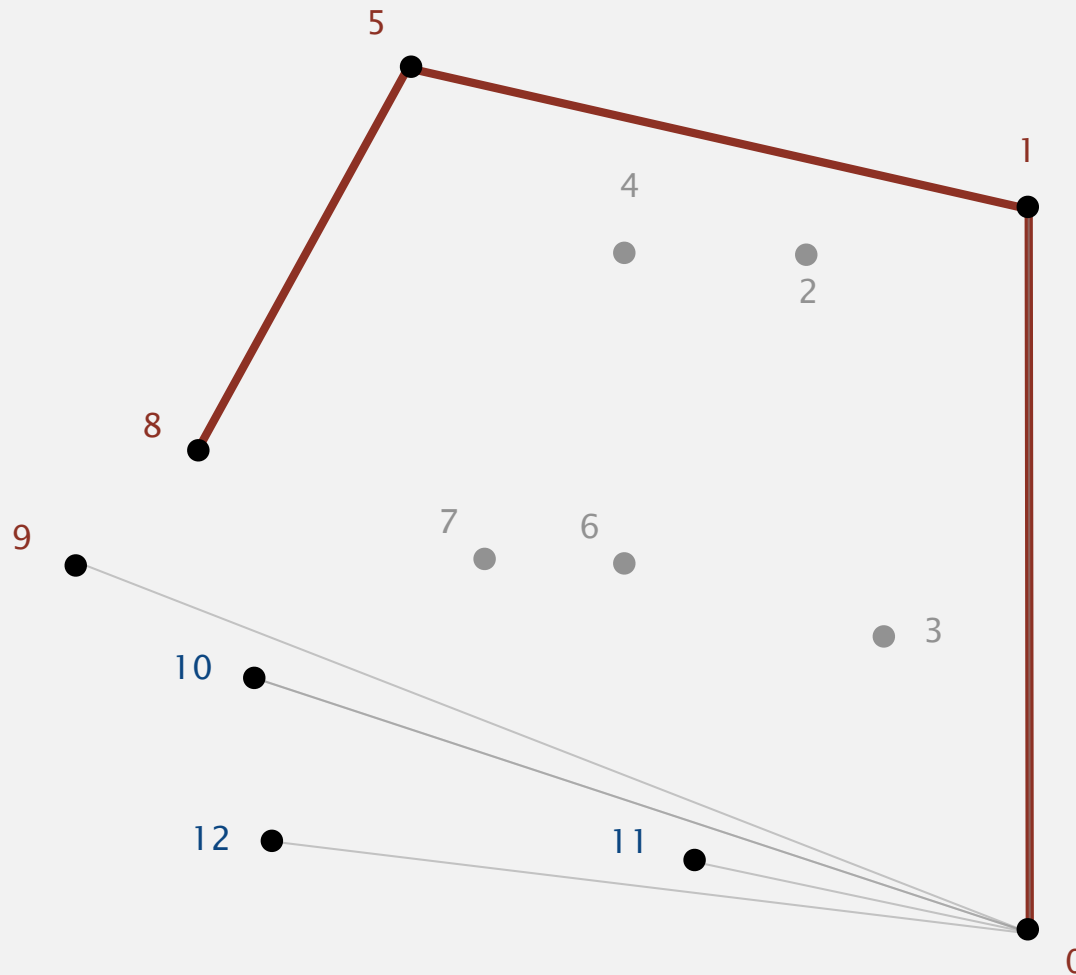
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

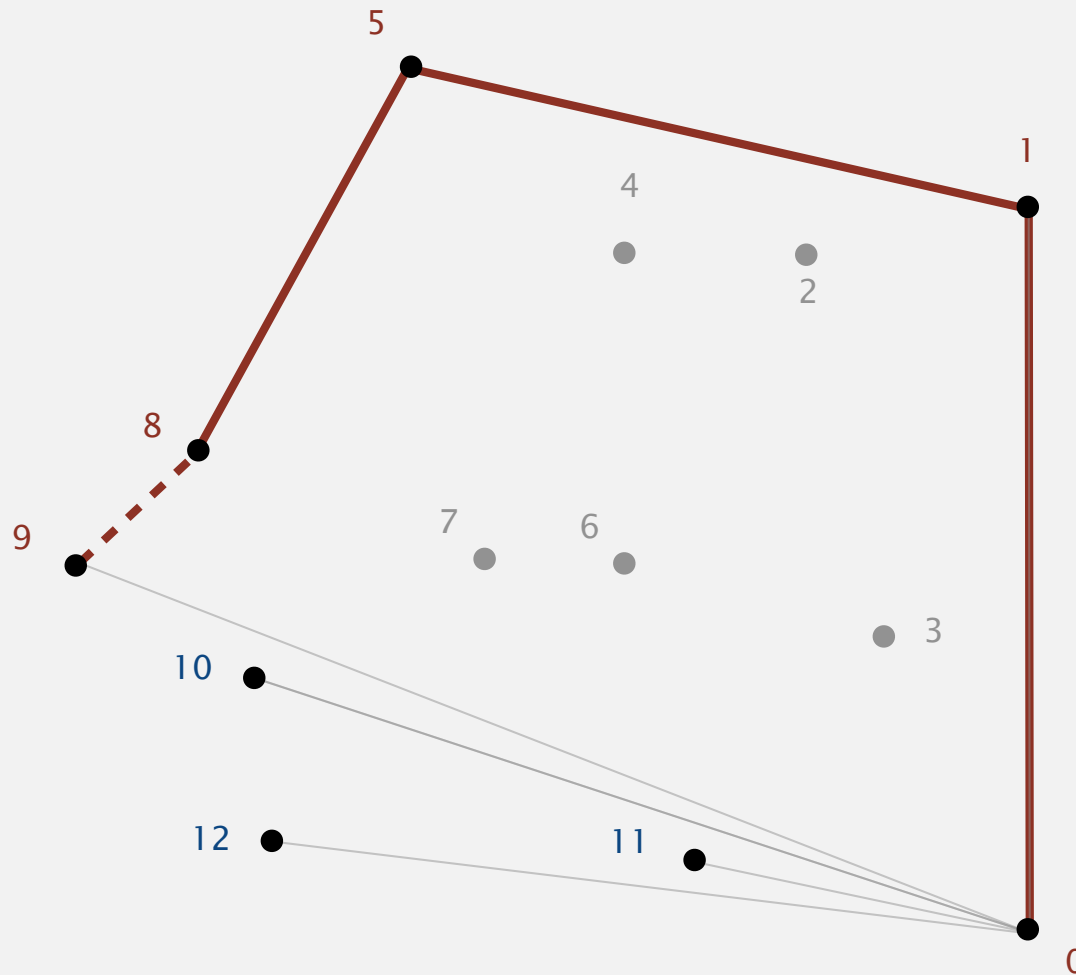
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

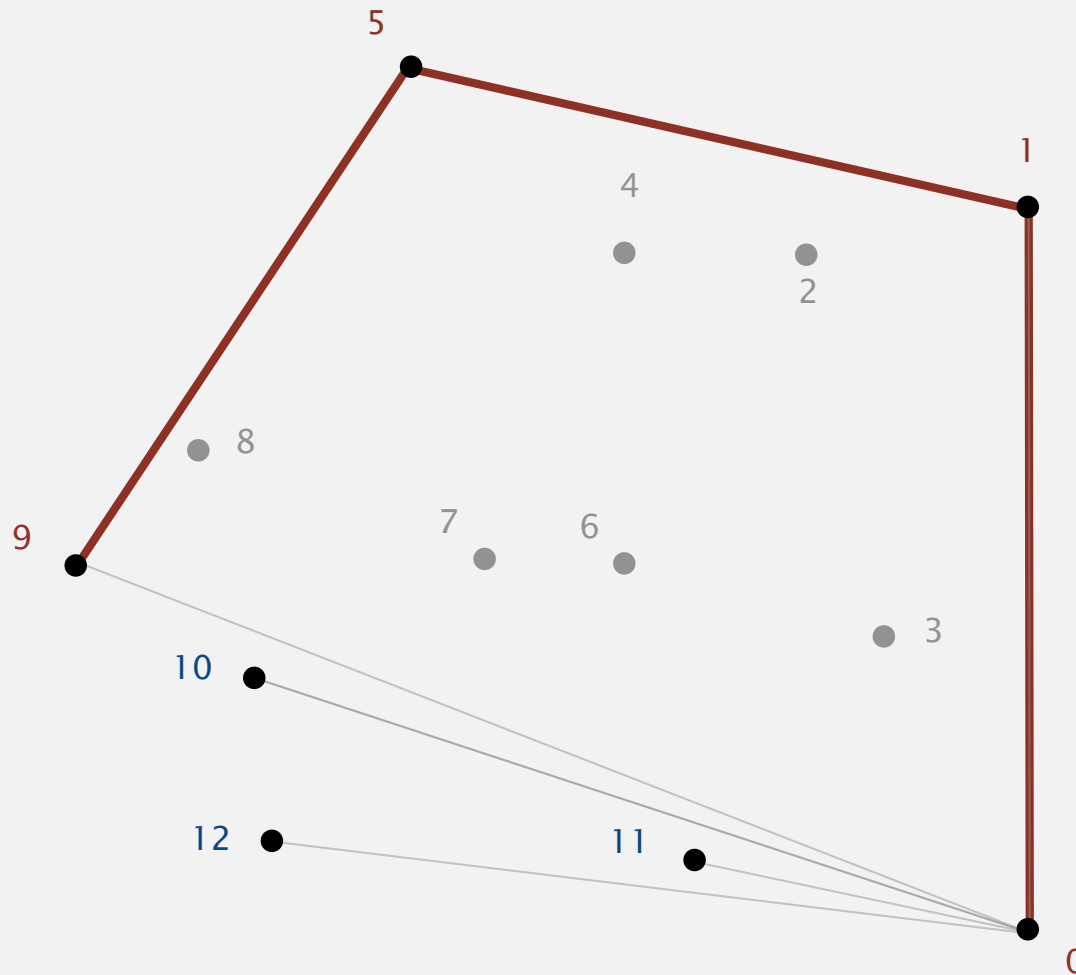
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

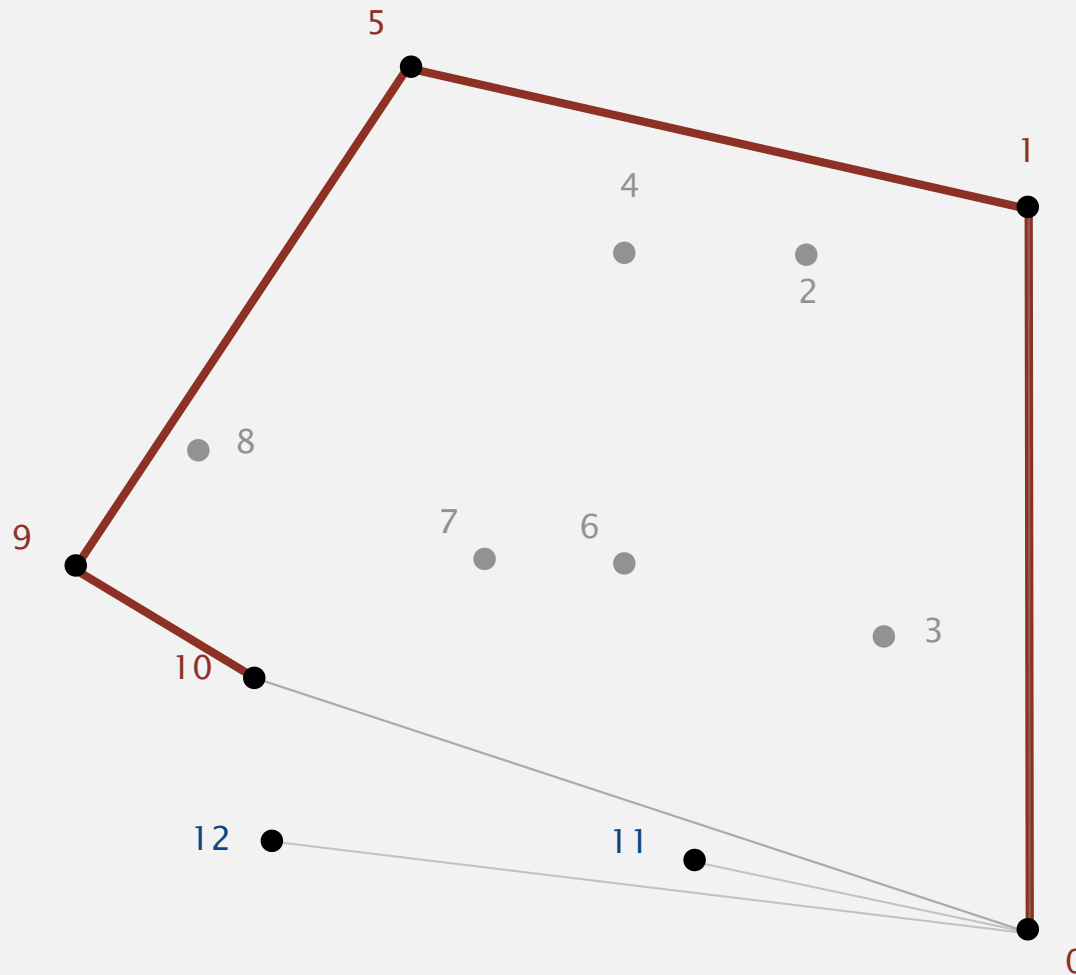
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

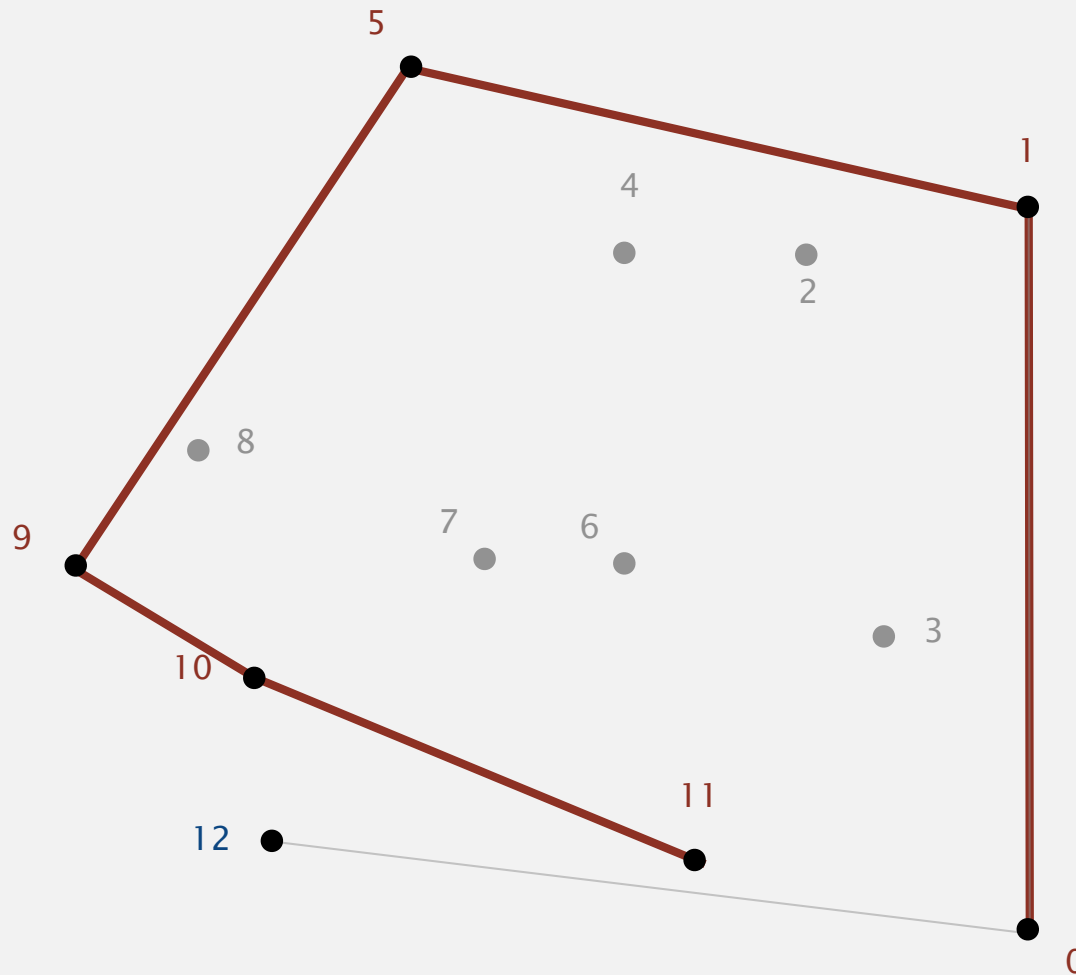
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.

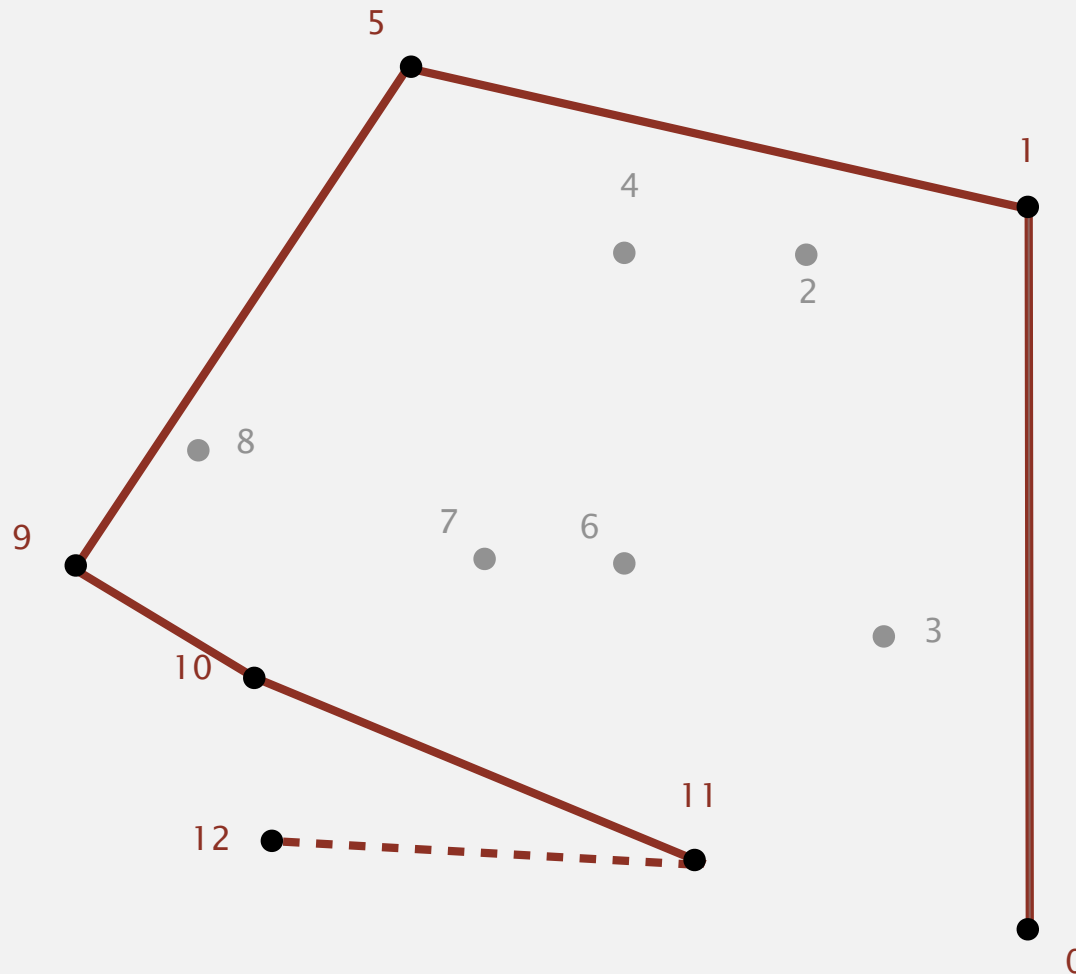




# Graham scan demo

---

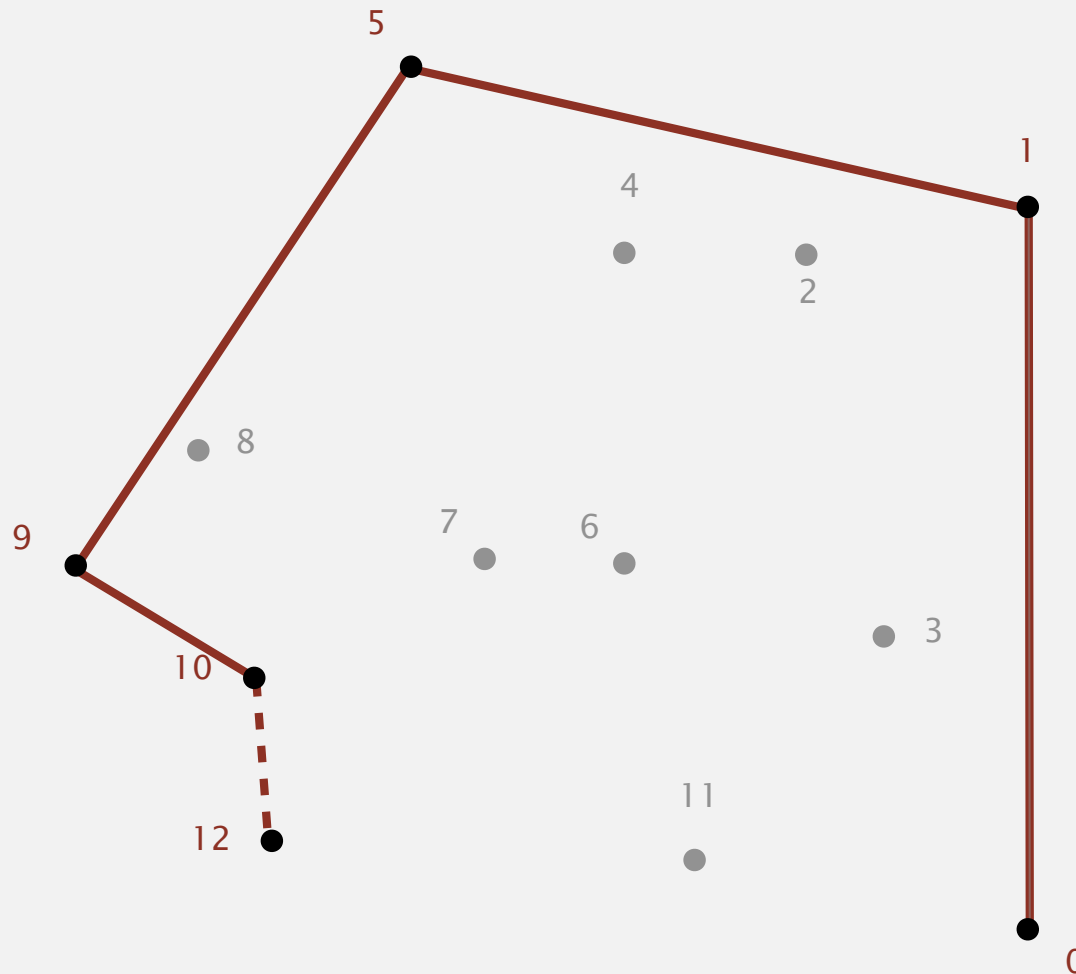
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

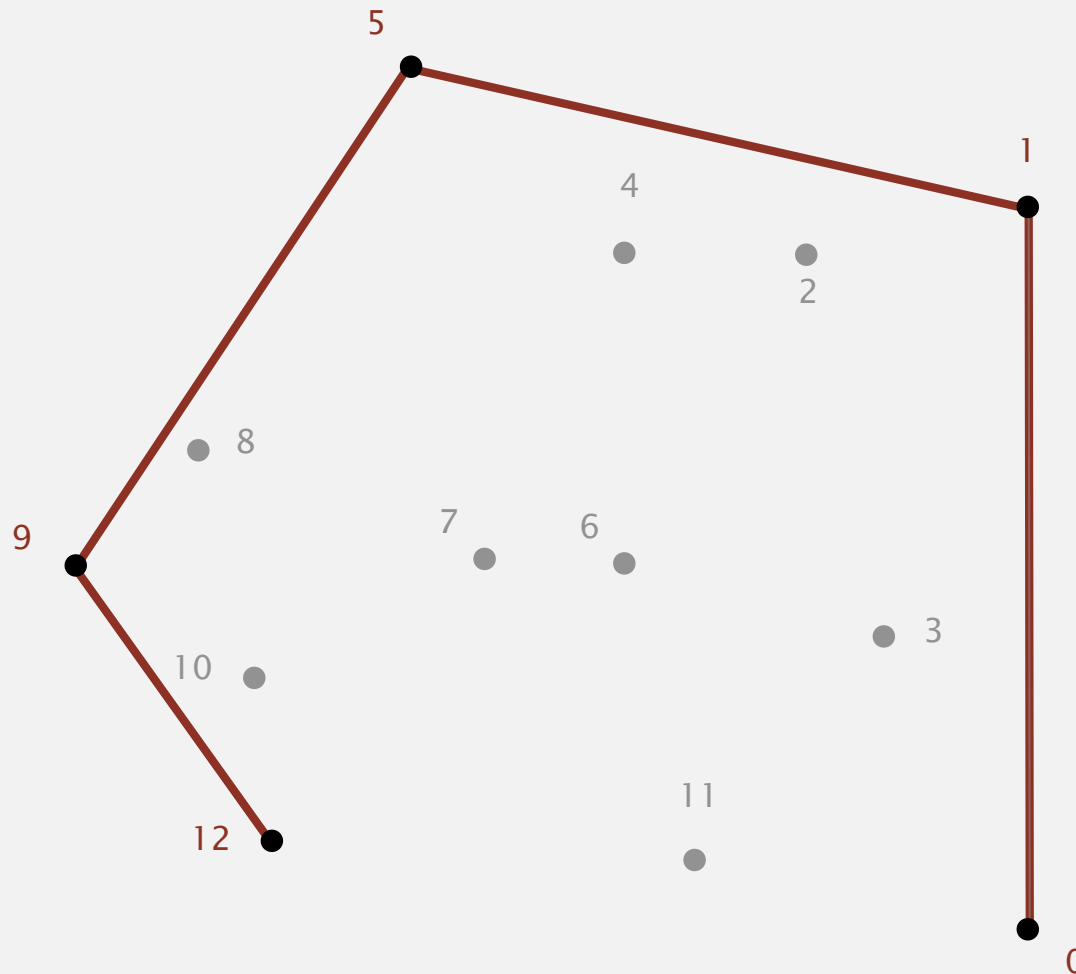
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

---

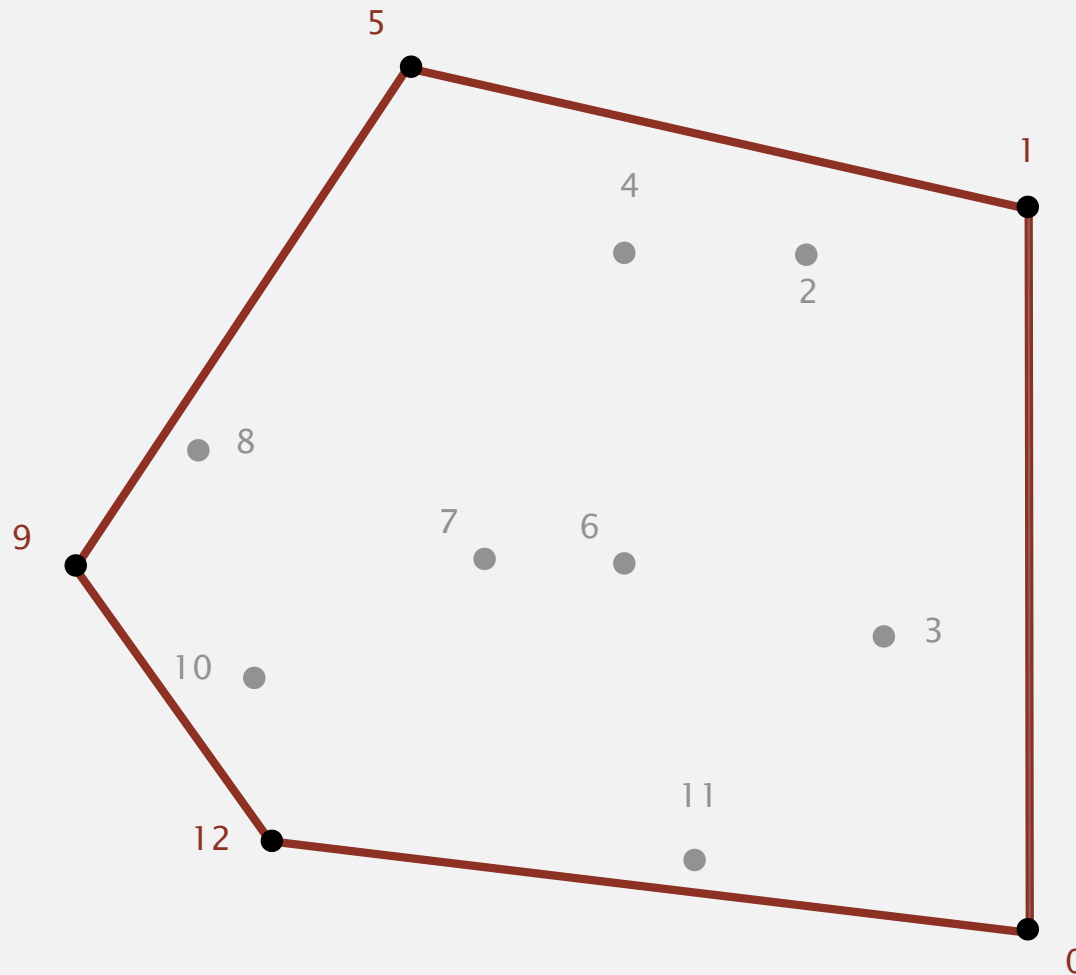
- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Graham scan demo

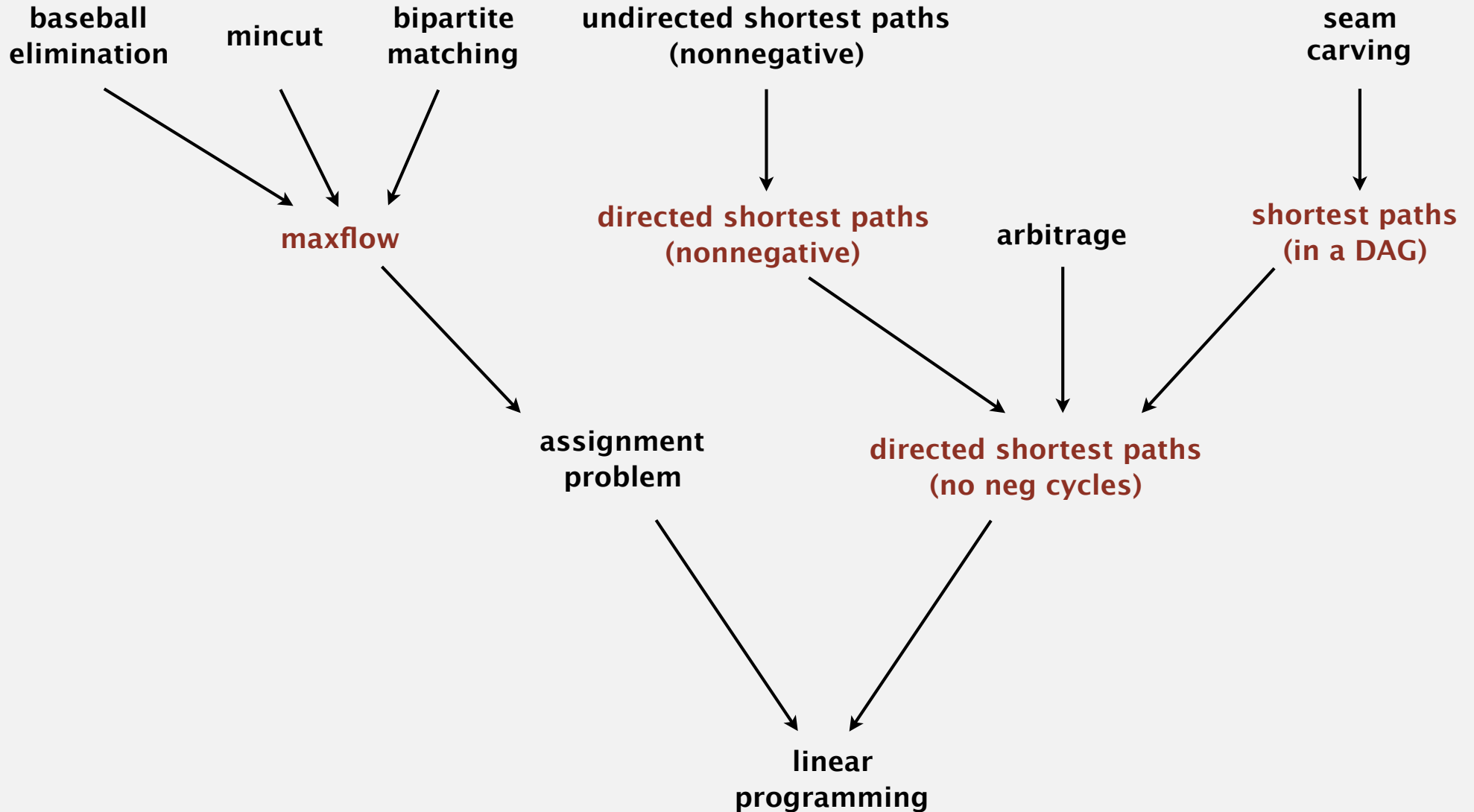
---

- Choose point  $p$  with smallest  $y$ -coordinate.
- Sort points by polar angle with  $p$ .
- Consider points in order; discard those that create clockwise turn.



# Some reductions in combinatorial optimization

---





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

## 6.5 REDUCTIONS

---

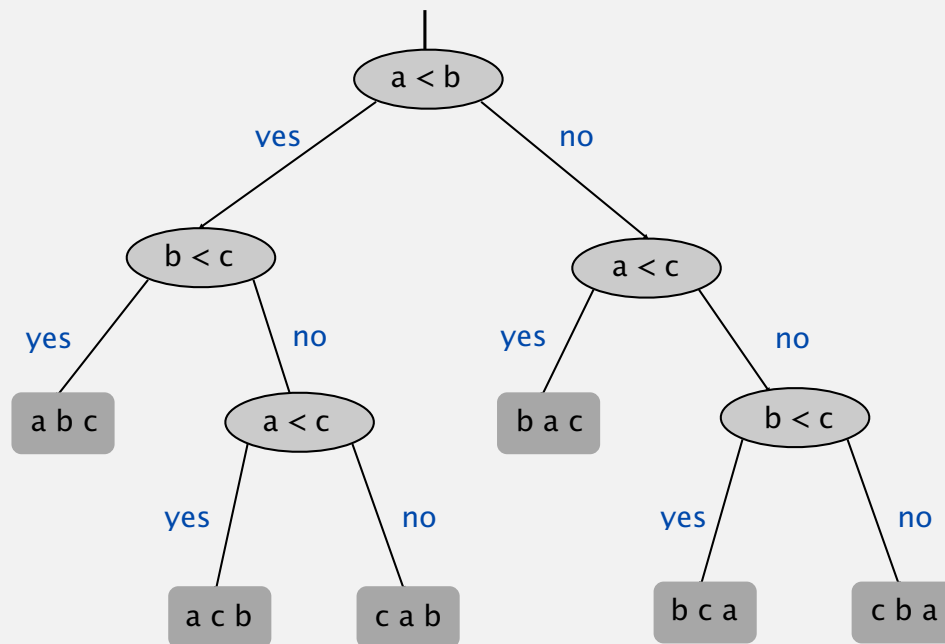
- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*
- ▶ *intractability*

# 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  $Y$ .

**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  $Y$ .
- Intuition:  $X$  = known problem;  $Y$  = new problem.

**Reasoning.**

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



## Reductions: quiz 2

---

Which of the following reductions is **not** a linear-time reduction?

- A. ELEMENT-DISTINCTNESS reduces to SORTING.
- B. MIN-CUT reduces to MAX-FLOW.
- C. HAMILTONIAN-PATH reduces to HAMILTONIAN-CYCLE.
- D. BURROWS-WHEELER-TRANSFORM reduces to SUFFIX-SORTING.
- E. *I don't know.*

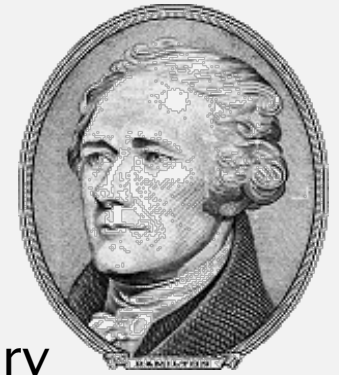
not the one we saw earlier,  
anyway



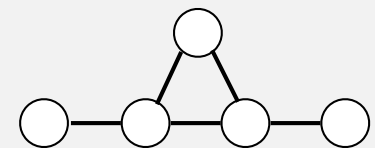
## Exercise: linear-time reduction

---

Imagine that founding father Alexander Hamilton has offered to find a **Hamiltonian cycle** in any given graph (if one exists).



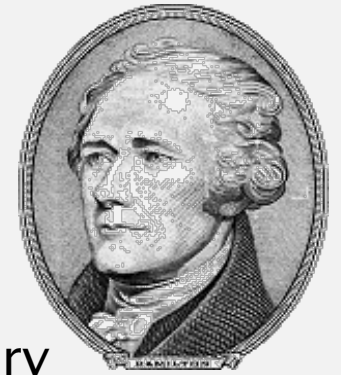
Design an efficient algorithm to find a **Hamiltonian path** in a graph (if one exists) by making queries to Hamilton. The Treasury Secretary's time is valuable, so you must minimize the number of queries.



## Exercise: linear-time reduction

---

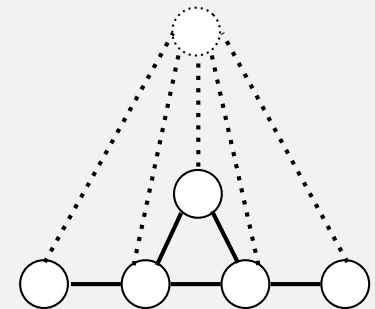
Imagine that founding father Alexander Hamilton has offered to find a **Hamiltonian cycle** in any given graph (if one exists).



Design an efficient algorithm to find a **Hamiltonian path** in a graph (if one exists) by making queries to Hamilton. The Treasury Secretary's time is valuable, so you must minimize the number of queries.

**Solution.** Given graph  $G = (V, E)$ :

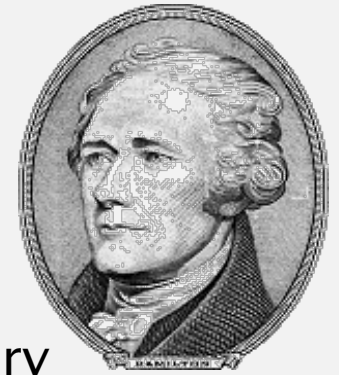
- Add a virtual vertex  $v$  and connect it to all vertices.
- Query Hamilton with the resulting graph:



## Exercise: linear-time reduction

---

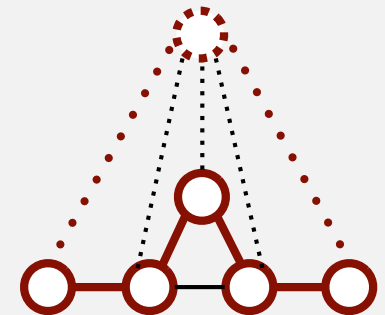
Imagine that founding father Alexander Hamilton has offered to find a **Hamiltonian cycle** in any given graph (if one exists).



Design an efficient algorithm to find a **Hamiltonian path** in a graph (if one exists) by making queries to Hamilton. The Treasury Secretary's time is valuable, so you must minimize the number of queries.

**Solution.** Given graph  $G = (V, E)$ :

- Add a virtual vertex  $v$  and connect it to all vertices.
- Query Hamilton with the resulting graph:
  - If cycle found, rotate so that  $v$  is first/last vertex; remove  $v$  and return the rest.
  - Else return “no Hamiltonian path”.



Why is this correct?

# Lower bound for convex hull

---

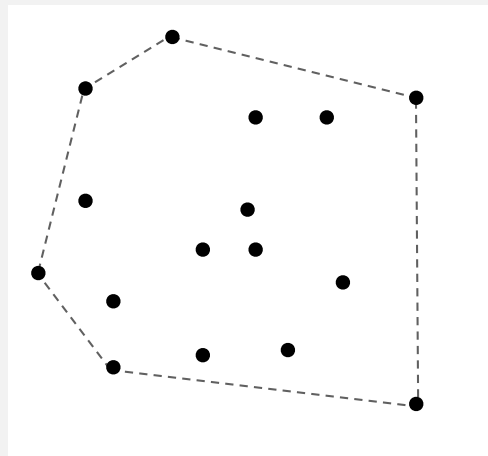
**Proposition.** Sorting linear-time reduces to convex hull.

**Pf.** [see next slide]

lower-bound reasoning:  
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



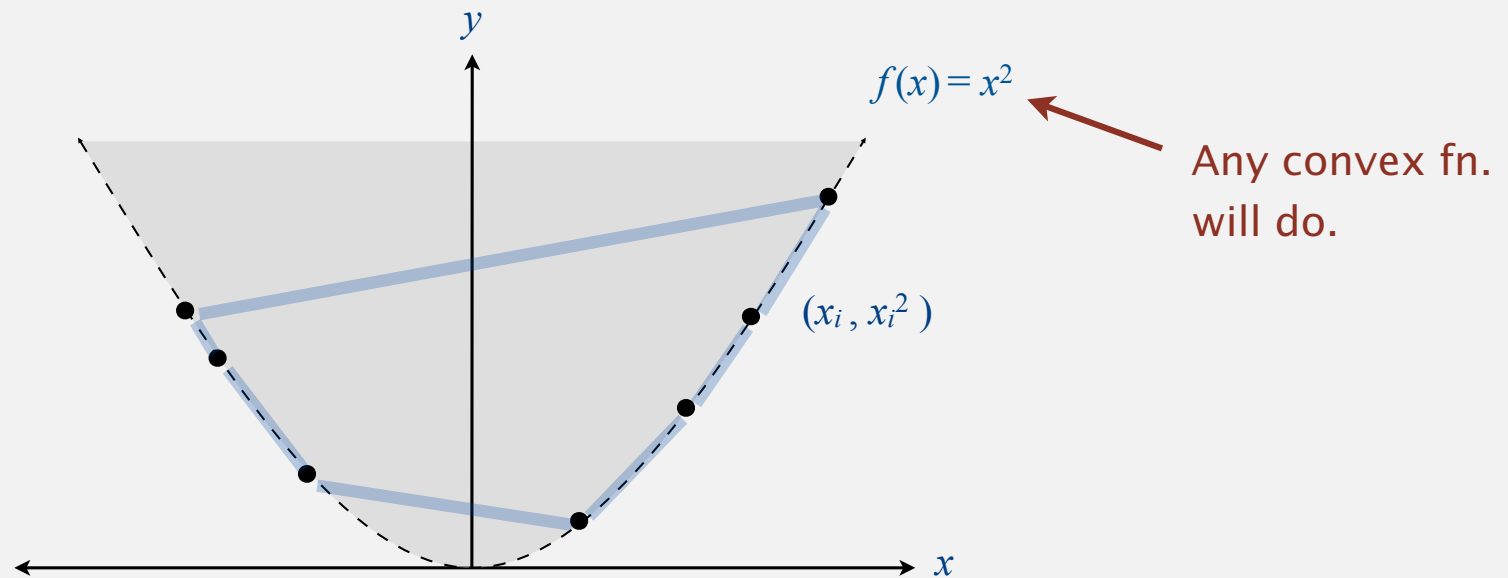
convex hull

**Implication.** Any 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  $\{ (x, y) : y \geq x^2 \}$  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.

## Reductions: quiz 3

---

Our lower bound proof strategy for CONVEX-HULL would work even if:

- A.** Our reduction invoked CONVEX-HULL  $\Theta(\log N)$  times instead of once.
- B.** Our pre-/post-processing was linearithmic instead of linear.
- C.** Both **A.** and **B.**
- D.** Neither **A.** nor **B.**
- E.** *I don't know.*

Cost of solving SORTING = total cost of CONVEX-HULL + cost of reduction.





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

## 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>	$N^2$	?
⋮	⋮	⋮
<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*

# Classifying problems: summary

---

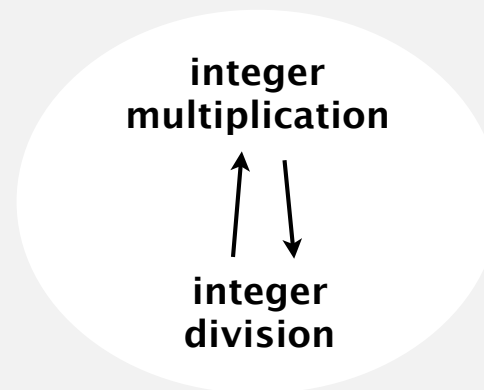
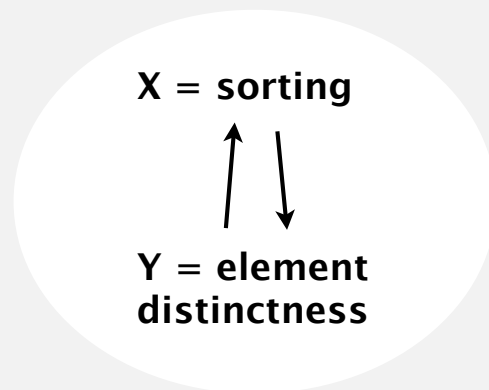
**Desiderata.** Problem with algorithm that matches lower bound.

**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  $T(N)$  iff  $Y$  has complexity  $T(N)$ .

even if we don't know what it is





# 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>	$N^{1+\epsilon}$
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.

**GMP**  
«Arithmetic without limitations»

# Lower bound for 3-COLLINEAR

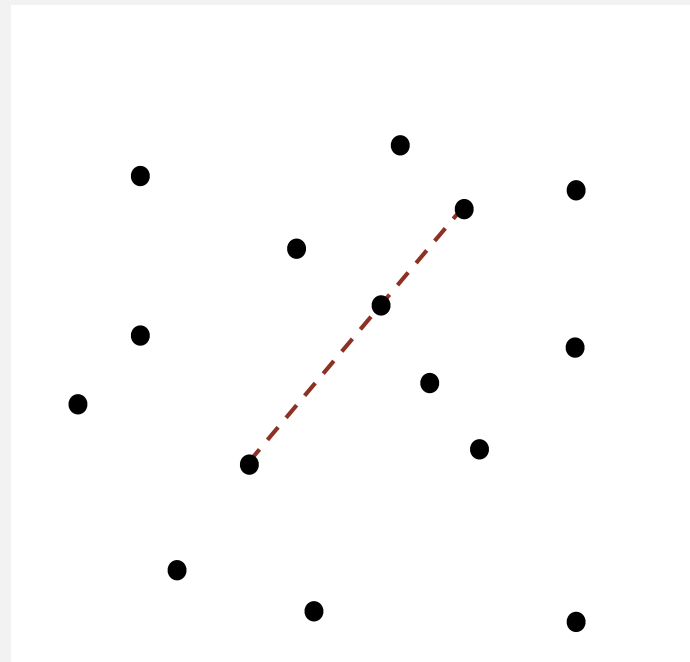
---

**3-SUM.** Given  $N$  distinct integers, are there three that sum to 0?

**3-COLLINEAR.** Given  $N$  distinct points in the plane, are there 3 (or more) that lie on the same line?

```
590584
-23439854
1251432
-2861534
3988818
-4190745
333255
13546464
89885444
-43434213
11998833
```

**3-sum**



**3-collinear**



# Lower bound for 3-COLLINEAR

---


**3-SUM.** Given  $N$  distinct integers, are there three that sum to 0?

**3-COLLINEAR.** Given  $N$  distinct points in the plane, are there 3 (or more) that lie on the same line?

**Proposition.** 3-SUM linear-time reduces to 3-COLLINEAR.

**Pf.** [next two slides]

lower-bound reasoning:  
if I can't solve 3-SUM in  $N^{1.99}$  time,  
I can't solve 3-COLLINEAR  
in  $N^{1.99}$  time either



**Conjecture.** No sub-quadratic algorithm for 3-SUM.

**Implication.** No sub-quadratic algorithm for 3-COLLINEAR likely.

our  $N^2 \log N$  algorithm was pretty good



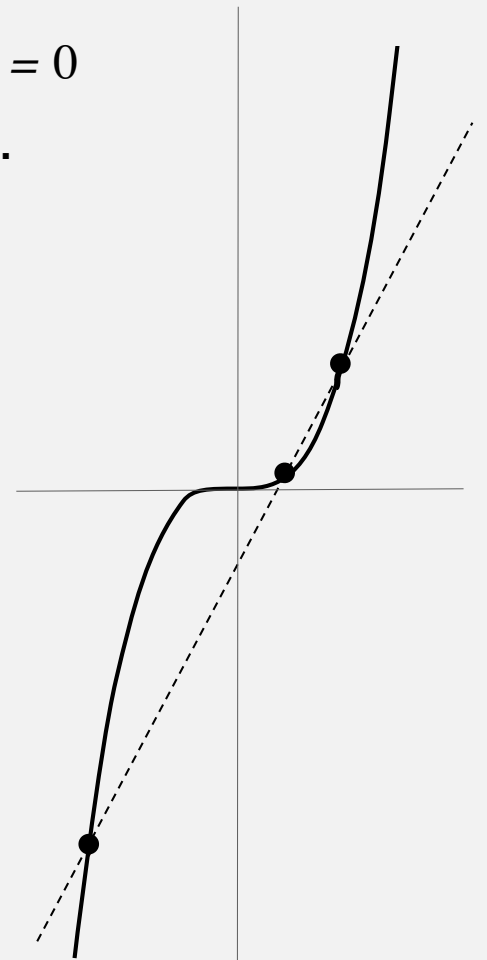
## 3-SUM linear-time reduces to 3-COLLINEAR

---

**Reduction.** 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance:  $x_1, x_2, \dots, x_N$ .
- 3-COLLINEAR instance:  $(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_N, f(x_N))$ .

**We hope to prove:** If  $a, b$ , and  $c$  are distinct, then  $a + b + c = 0$  if and only if  $(a_1, f(a_1)), (b_2, f(b_2)), \dots, (c_N, f(c_N))$  are collinear.



# 3-SUM linear-time reduces to 3-COLLINEAR

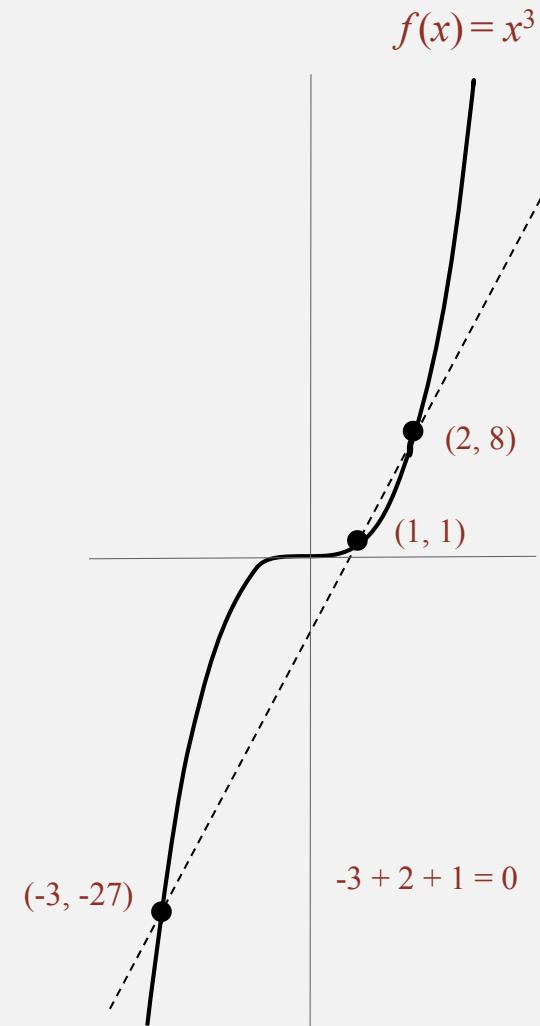
**Proposition.** 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance:  $x_1, x_2, \dots, x_N$ .
- 3-COLLINEAR instance:  $(x_1, x_1^3), (x_2, x_2^3), \dots, (x_N, x_N^3)$ .

**Lemma.** If  $a, b$ , and  $c$  are distinct, then  $a + b + c = 0$  if and only if  $(a, a^3), (b, b^3)$ , and  $(c, c^3)$  are collinear.

**Pf.** Three distinct points  $(a, a^3), (b, b^3), (c, c^3)$  are collinear iff:

$$\begin{aligned} 0 &= \begin{vmatrix} a & a^3 & 1 \\ b & b^3 & 1 \\ c & c^3 & 1 \end{vmatrix} \\ &= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3) \\ &= (a - b)(b - c)(c - a)(a + b + c) \end{aligned}$$



# Complexity of 3-SUM

---

Some recent (2014) evidence that the complexity might be  $N^{3/2}$ .

## Threesomes, Degenerates, and Love Triangles\*

Allan Grønlund  
MADALGO, Aarhus University

Seth Pettie  
University of Michigan

April 4, 2014

### Abstract

The 3SUM problem is to decide, given a set of  $n$  real numbers, whether any three sum to zero. We prove that the decision tree complexity of 3SUM is  $O(n^{3/2}\sqrt{\log n})$ , that there is a randomized 3SUM algorithm running in  $O(n^2(\log \log n)^2/\log n)$  time, and a deterministic algorithm running in  $O(n^2(\log \log n)^{5/3}/(\log n)^{2/3})$  time. These results refute the strongest version of the 3SUM conjecture, namely that its decision tree (and algorithmic) complexity is  $\Omega(n^2)$ .

# Reductions: summary

---

Reduction: relationship between two problems.

How to apply:

- Reduction to solved problem: paradigm for designing algorithms.
- Reduction **from** solved problem: technique for proving lower bounds.
- Putting the two together: classify **problems** into complexity classes.
  - Especially useful for proving NP-completeness.

Reductions require ingenuity, but a few tricks recur.



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

## 6.5 REDUCTIONS

---

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*
- ▶ *intractability (next lecture)*

# Bird's-eye view

---

**Def.** A problem is **intractable** if it can't be solved in polynomial time.

**Desiderata.** Prove that a problem is intractable.

Two problems that provably require exponential time.

- Given a constant-size program, does it halt in at most  $K$  steps?
- Given  $N$ -by- $N$  checkers board position, can the first player force a win?

input size =  $c + \lg K$



using forced capture rule



*Alan designed the perfect computer*



Frustrating news. Very few successes.

# A core problem: satisfiability

---

**SAT.** Given a system of boolean equations, find a solution.

**Ex.**

$$\begin{array}{l} \neg x_1 \quad \text{or} \quad x_2 \quad \text{or} \quad x_3 \quad = \quad \text{true} \\ x_1 \quad \text{or} \quad \neg x_2 \quad \text{or} \quad x_3 \quad = \quad \text{true} \\ \neg x_1 \quad \text{or} \quad \neg x_2 \quad \text{or} \quad \neg x_3 \quad = \quad \text{true} \\ \neg x_1 \quad \text{or} \quad \neg x_2 \quad \text{or} \quad \quad \quad \text{or} \quad x_4 \quad = \quad \text{true} \\ \quad \quad \quad \neg x_2 \quad \text{or} \quad x_3 \quad \text{or} \quad x_4 \quad = \quad \text{true} \end{array}$$

**instance I**

$$\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ T & T & F & T \end{array}$$

**solution S**

**3-SAT.** All equations of this form (with three variables per equation).

## Key applications.

- Automatic verification systems for software.
- Mean field diluted spin glass model in physics.
- Electronic design automation (EDA) for hardware.
- ...



# Satisfiability is conjectured to be intractable

---

Q. How to solve an instance of 3-SAT with  $N$  variables?

A. Exhaustive search: try all  $2^N$  truth assignments.



Q. Can we do anything substantially more clever?

Conjecture (**P**  $\neq$  **NP**). 3-SAT is intractable (no poly-time algorithm).

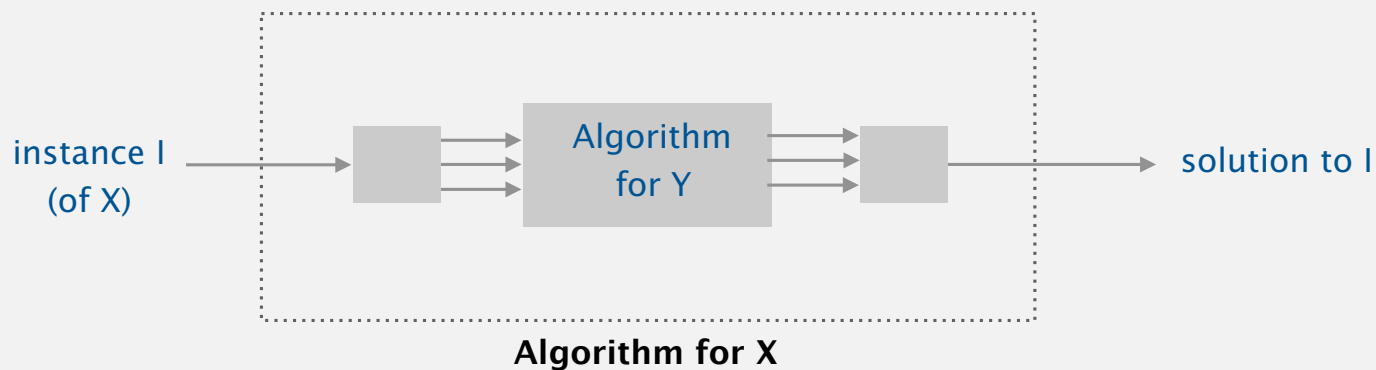
↖  
consensus opinion

# Polynomial-time reductions

---

Problem  $X$  **poly-time (Cook) reduces** to problem  $Y$  if  $X$  can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to  $Y$ .



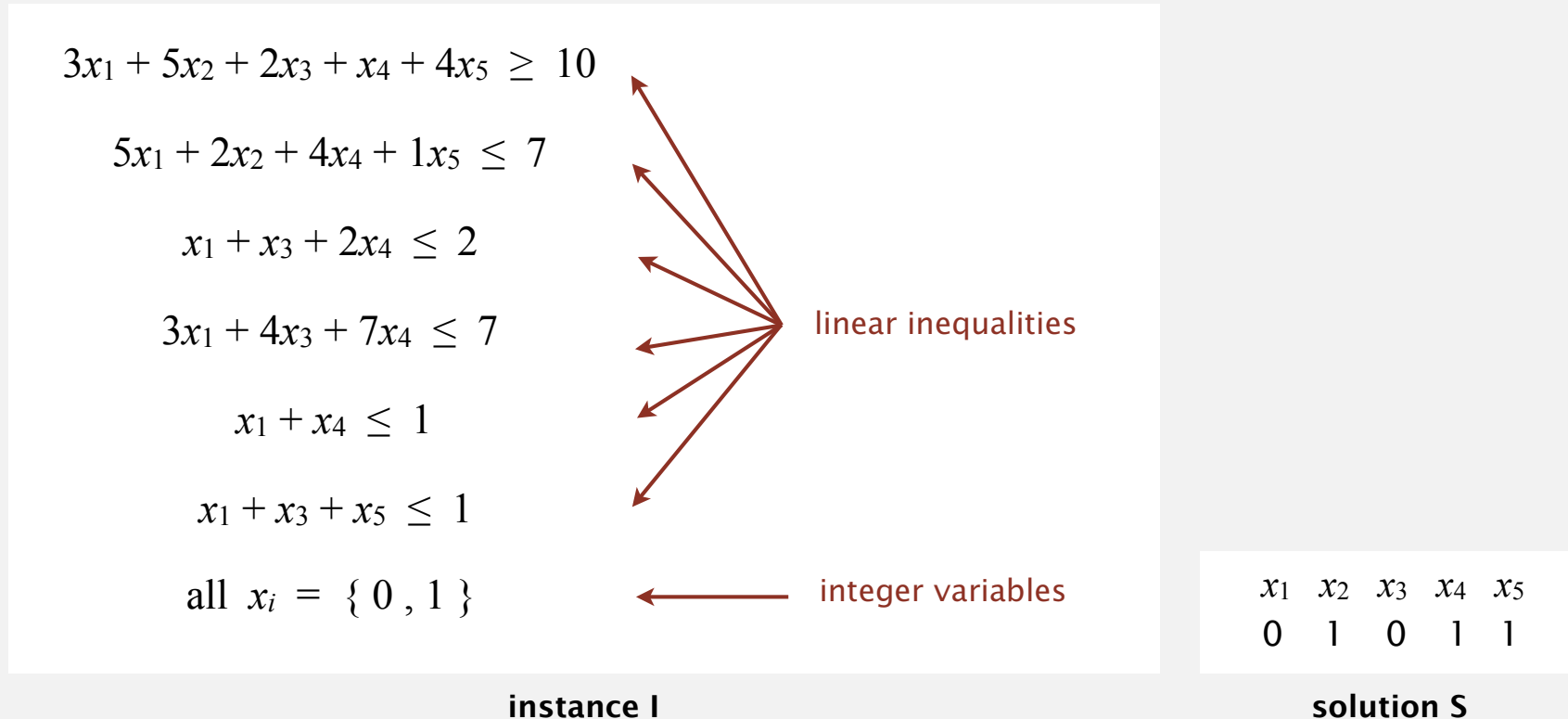
**Establish intractability.** If 3-SAT poly-time reduces to  $Y$ , then  $Y$  is intractable. (assuming 3-SAT is intractable)

**Reasoning.**

- If I could solve  $Y$  in poly-time, then I could also solve 3-SAT in poly-time.
- 3-SAT is believed to be intractable.
- Therefore, so is  $Y$ .

# Integer linear programming

**ILP.** Given a system of linear inequalities, find an **integral** solution.



**Context.** Cornerstone problem in operations research.

**Remark.** Finding a real-valued solution is tractable (linear programming).

## 3-SAT poly-time reduces to ILP

---

**3-SAT.** Given a system of boolean equations, find a solution.

$$\begin{array}{l} \neg x_1 \text{ or } x_2 \text{ or } x_3 = \text{true} \\ x_1 \text{ or } \neg x_2 \text{ or } x_3 = \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } \neg x_3 = \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } x_4 = \text{true} \\ \neg x_2 \text{ or } x_3 \text{ or } x_4 = \text{true} \end{array}$$

**ILP.** Given a system of linear inequalities, find a 0-1 solution.

$$\begin{array}{l} (1 - x_1) + x_2 + x_3 \geq 1 \\ x_1 + (1 - x_2) + x_3 \geq 1 \\ (1 - x_1) + (1 - x_2) + (1 - x_3) \geq 1 \\ (1 - x_1) + (1 - x_2) + x_4 \geq 1 \\ (1 - x_2) + x_3 + x_4 \geq 1 \end{array}$$

**solution to this ILP instance gives solution to original 3-SAT instance**

## Reductions: quiz 3

---

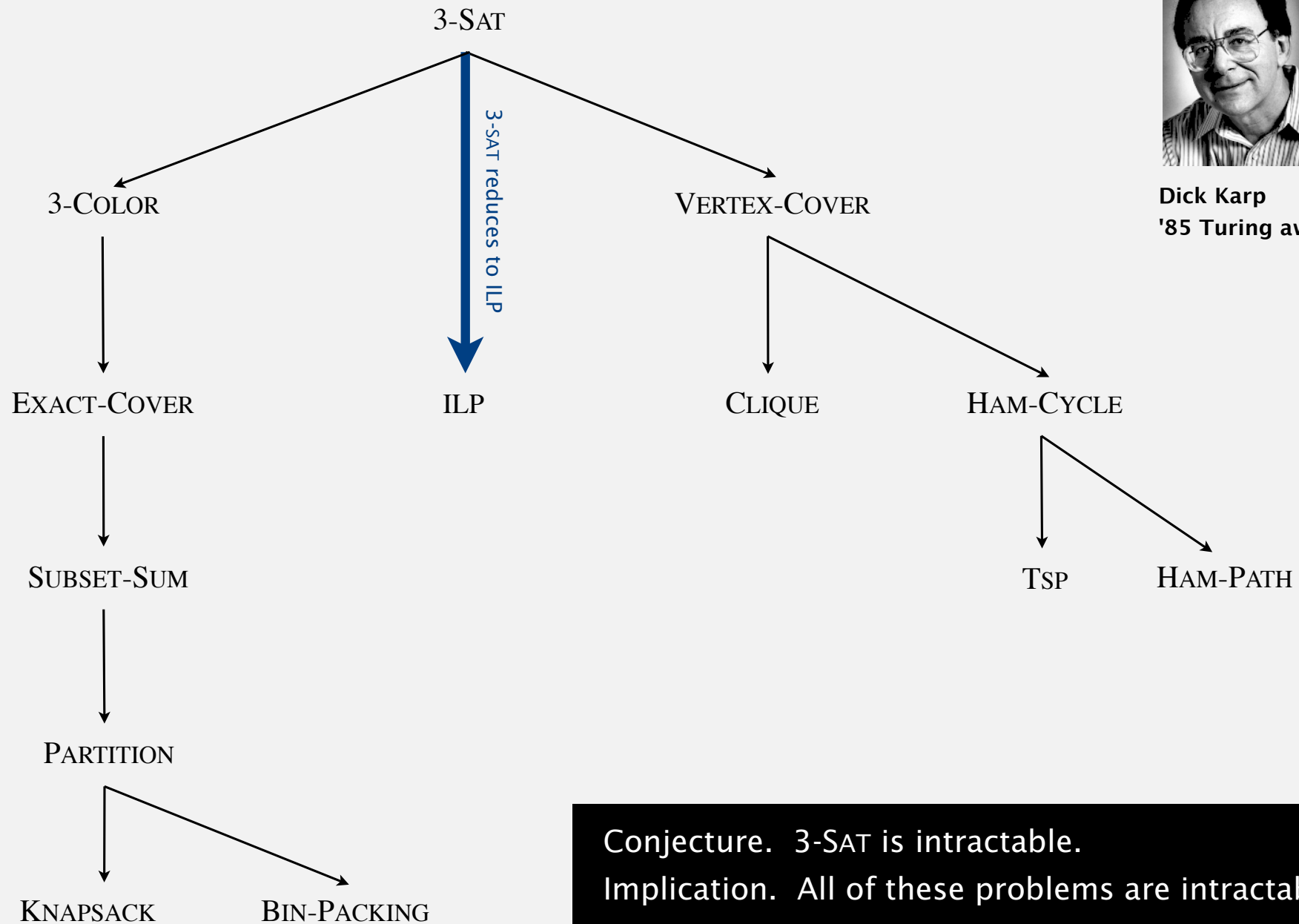
Suppose that Problem  $X$  poly-time reduces to Problem  $Y$ . Which of the following can you infer?

- A.** If  $X$  can be solved in poly-time, then so can  $Y$ .
- B.** If  $X$  cannot be solved in cubic time,  $Y$  cannot be solved in poly-time.
- C.** If  $Y$  can be solved in cubic time, then  $X$  can be solved in poly-time.
- D.** If  $Y$  cannot be solved in poly-time, then neither can  $X$ .
- E.** *I don't know.*

# More poly-time reductions from 3-satisfiability



Dick Karp  
'85 Turing award



Conjecture. 3-SAT is intractable.  
Implication. All of these problems are intractable.

# Implications of poly-time reductions from 3-satisfiability

---

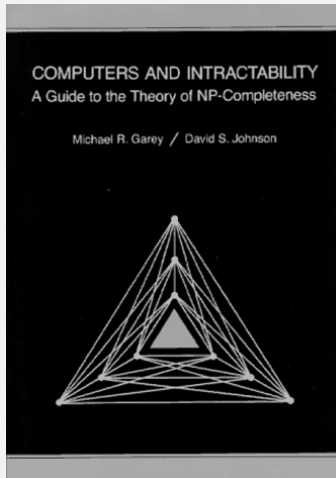
Establishing intractability through poly-time reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself that a new problem is (probably) intractable?

A1. [hard way] Long futile search for an efficient algorithm (as for 3-SAT).

A2. [easy way] Reduction from 3-SAT.

Caveat. Intricate reductions are common.



# Search problems

---

**Search problem.** Problem where you can check a solution in poly-time.

**Ex 1.** 3-SAT.

$$\begin{array}{l} \neg x_1 \text{ or } x_2 \text{ or } x_3 = \text{true} \\ x_1 \text{ or } \neg x_2 \text{ or } x_3 = \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } \neg x_3 = \text{true} \\ \neg x_1 \text{ or } \neg x_2 \text{ or } \text{ or } x_4 = \text{true} \\ \neg x_2 \text{ or } x_3 \text{ or } x_4 = \text{true} \end{array}$$

**instance I**

$x_1$	$x_2$	$x_3$	$x_4$
T	T	F	T

**solution S**

**Ex 2.** FACTOR. Given an  $N$ -bit integer  $x$ , find a nontrivial factor.

147573952589676412927

**instance I**

193707721

**solution S**



# P vs. NP

---

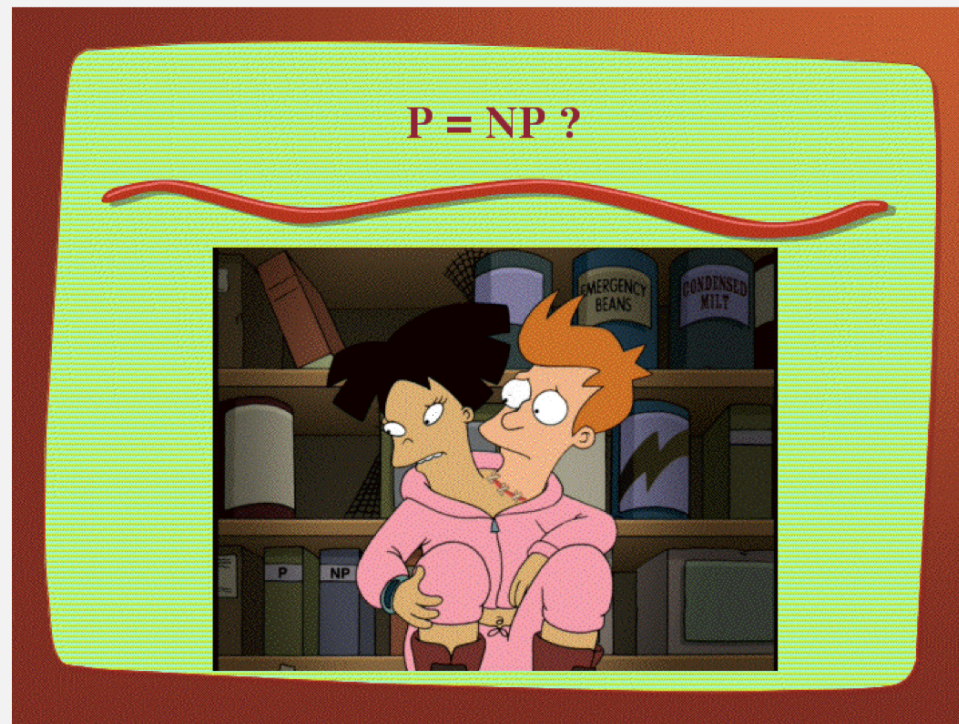
**P.** Set of search problems solvable in poly-time.

**Importance.** What scientists and engineers can compute feasibly.

**NP.** Set of search problems (checkable in poly-time).

**Importance.** What scientists and engineers aspire to compute feasibly.

Fundamental question.



Consensus opinion. No.

# Cook-Levin theorem

---

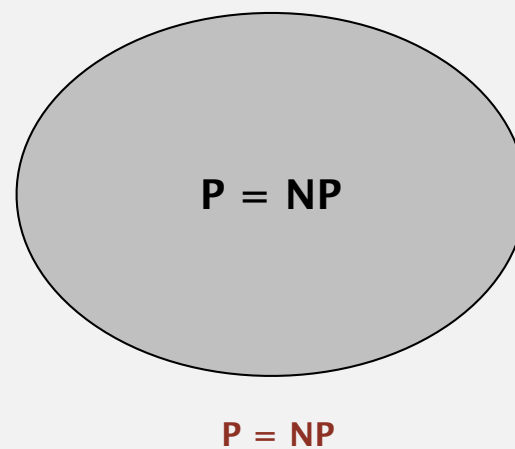
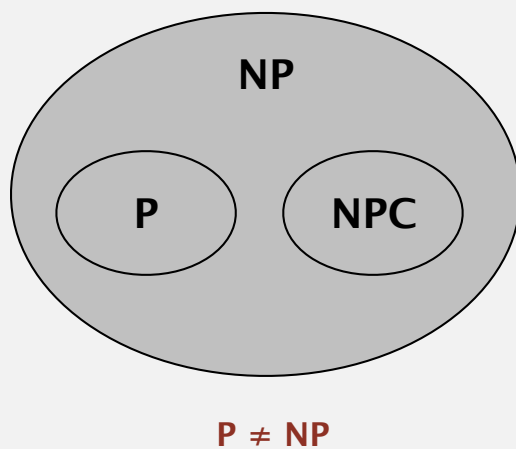
A problem is **NP-COMPLETE** if

- It is in **NP**.
- All problems in **NP** poly-time to reduce to it.

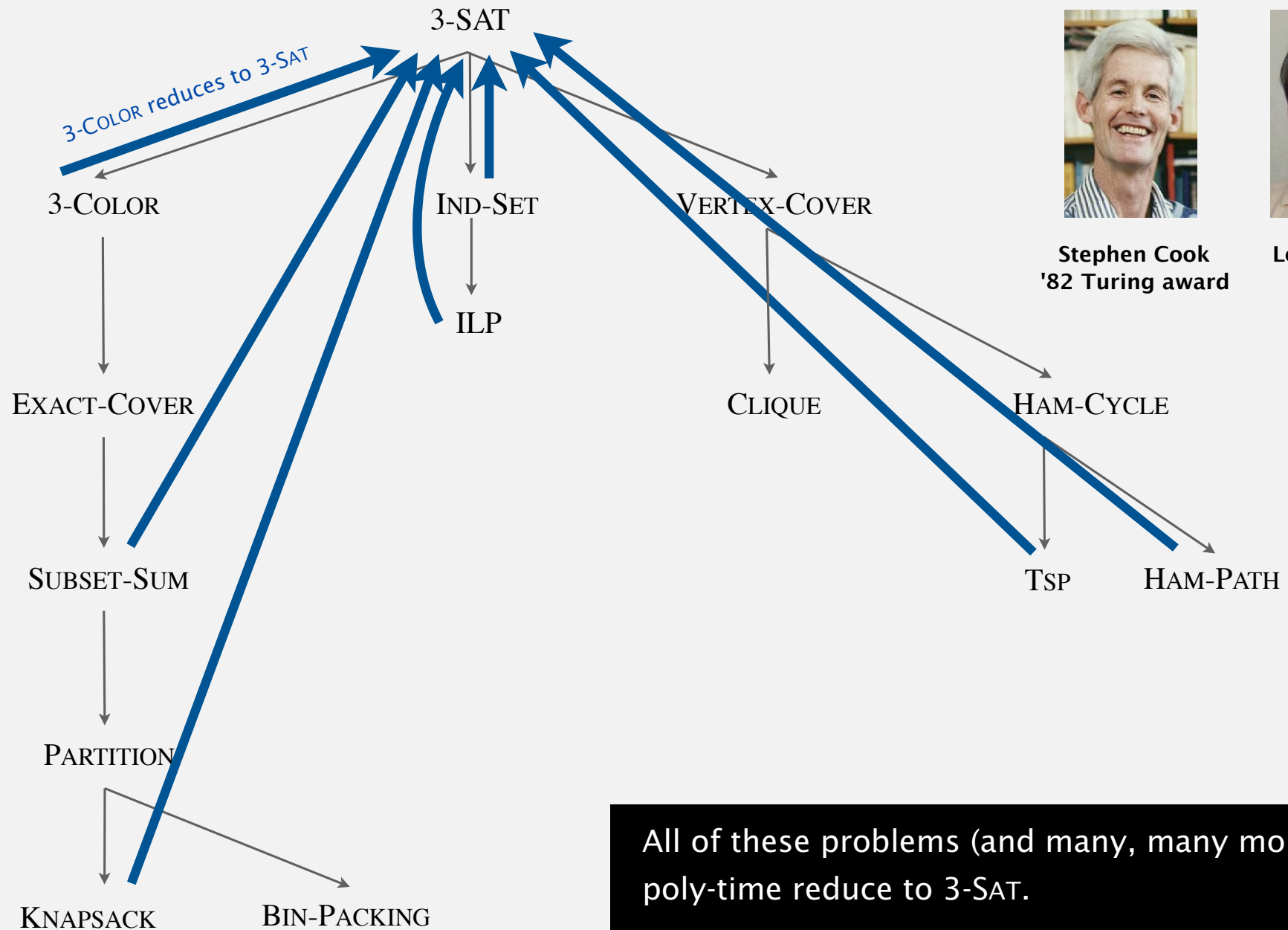
**Cook-Levin theorem.** 3-SAT is **NP-COMPLETE**.

**Corollary.** 3-SAT is tractable if and only if **P = NP**.

Two worlds.

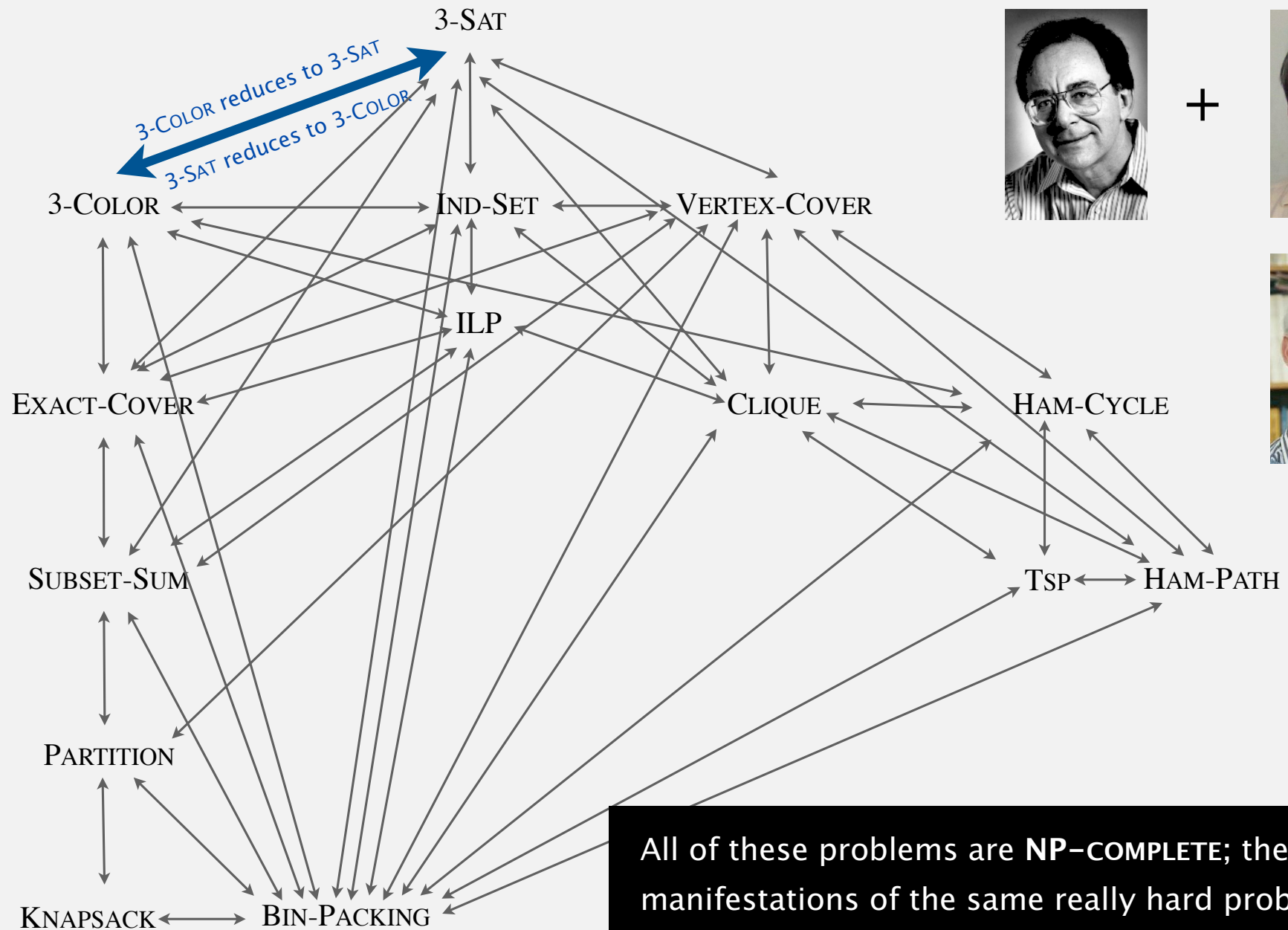


# Implications of Cook-Levin theorem



All of these problems (and many, many more) poly-time reduce to 3-SAT.

# Implications of Karp + Cook-Levin



## Reductions: quiz 4

---

Suppose that  $X$  is **NP-COMPLETE**,  $Y$  is in **NP**, and  $X$  poly-time reduces to  $Y$ . Which of the following statements can you infer?

- I.  $Y$  is NP-COMPLETE.
- II. If  $Y$  cannot be solved in poly-time, then  $P \neq NP$ .
- III. If  $P \neq NP$ , then neither  $X$  nor  $Y$  can be solved in poly-time.

- A. I only.
- B. II only.
- C. I and II only.
- D. I, II, and III.
- E. *I don't know.*

## Birds-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, ...</i>
<b>quadratic</b>	$N^2$	?
⋮	⋮	⋮
<b>exponential</b>	$c^N$	?

**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
<b>linear</b>	$N$	<i>min, max, median, Burrows-Wheeler transform, ...</i>
<b>linearithmic</b>	$N \log N$	<i>sorting, element distinctness, ...</i>
<b>M(N)</b>	?	<i>integer multiplication, division, square root, ...</i>
<b>MM(N)</b>	?	<i>matrix multiplication, <math>Ax = b</math>, least square, determinant, ...</i>
⋮	⋮	⋮
<b>NP-complete</b>	<i>probably not <math>N^b</math></i>	3-SAT, IND-SET, ILP, ...

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





# 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, suffix arrays
  - MST, shortest paths, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.

