Suppose...

An alien species is traveling towards Earth and wishes to avoid bloodshed before they arrive.

They want to send a light speed transmission of a proof of their scientific and technological superiority:

- They can only send binary data.
- They do not know our language.

What sequence of bits would prove their superiority?

http://algs4.cs.princeton.edu

REDUCTIONS AND TRACTABILITY

- *linear reductions*
- *theoretical uses of linear reductions*
- *tractability, P, and NP*

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

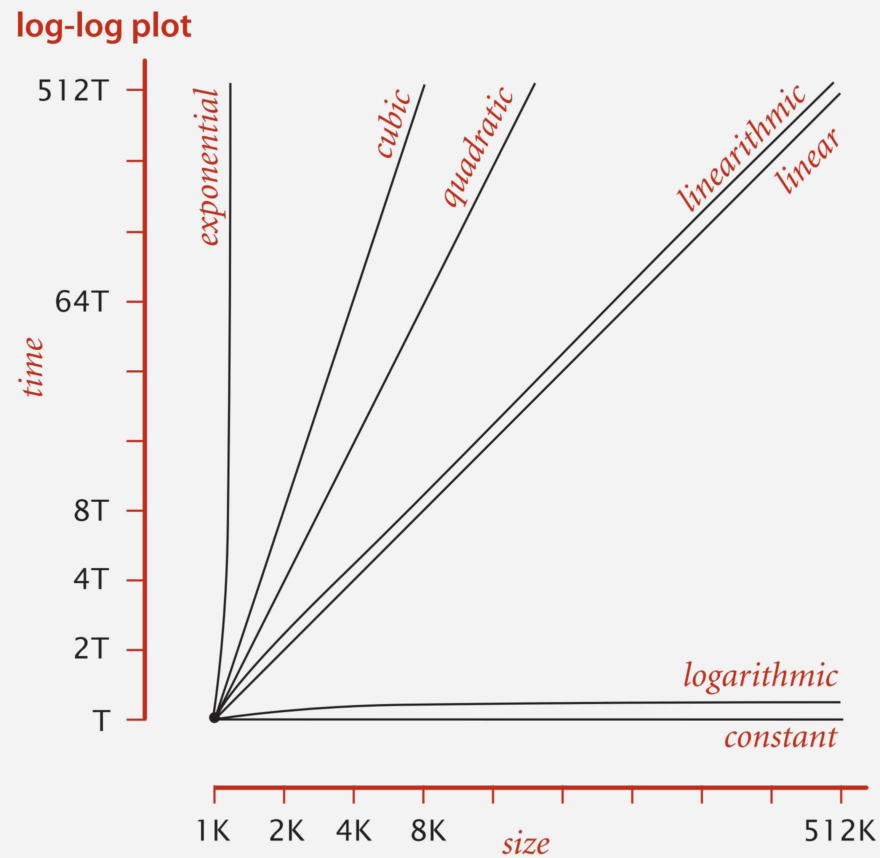
Reductions and Tractability

- *linear reductions*
- *theoretical uses of linear reductions*
- *tractability, P, and NP*

Overview: introduction to advanced topics

Main topics.

- Most of our problems so far have been easy.
  - Sorting, symbol table operations (array, LLRB, hash table, tries), graph search, MSTs, SPTs, substring matching, regex simulation, etc.
- Some have been hard.
  - 8puzzle.
  - Hamilton path.



Typical orders of growth

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, element distinctness, convex hull, closest pair, ...
quadratic	N^2	?
⋮	⋮	⋮
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.

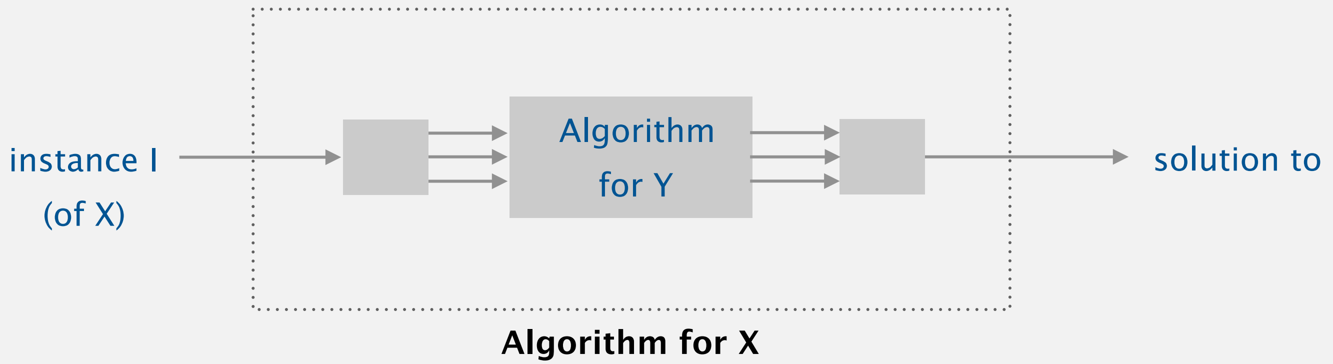
What else could (could not) we solve efficiently?

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

Reduction

Can also think of as "Y solves X"

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.

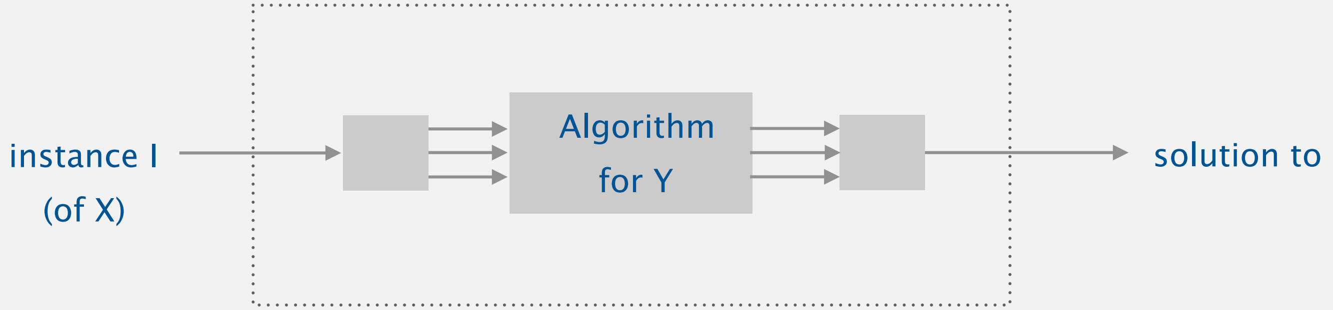
perhaps many calls to Y
on problems of different sizes
(though, typically only one call)

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

Reduction

Can also think of as "Y solves X"

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



Algorithm for X

Ex 1. [finding the median reduces to sorting]

To find the median of N items:

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

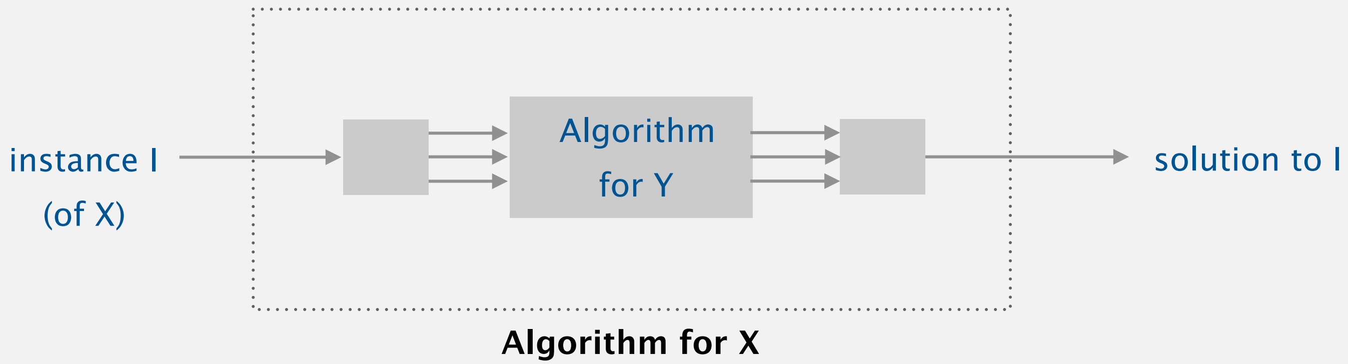
cost of sorting

cost of reduction

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

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.

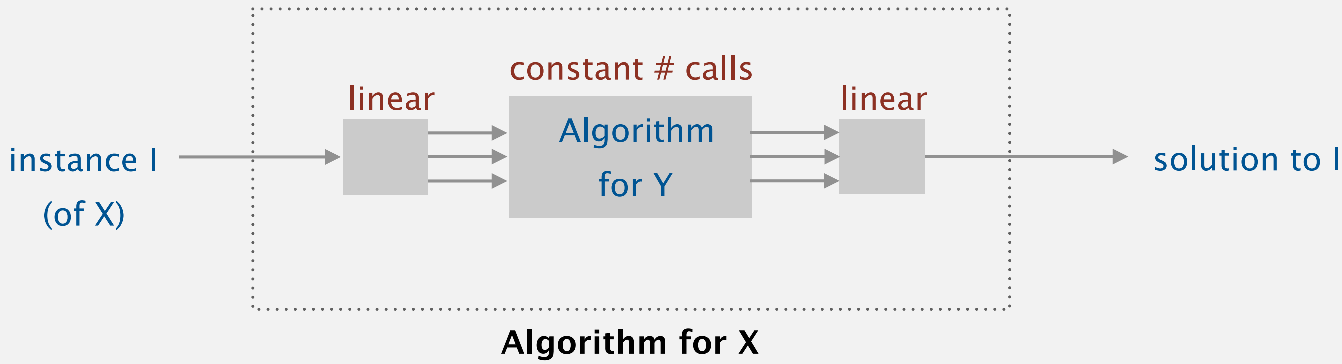
cost of sorting

cost of reduction

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

Reduction

Def. Problem X linear-time reduces to problem Y if X reduces to Y with linear reduction cost and constant number of calls to Y.



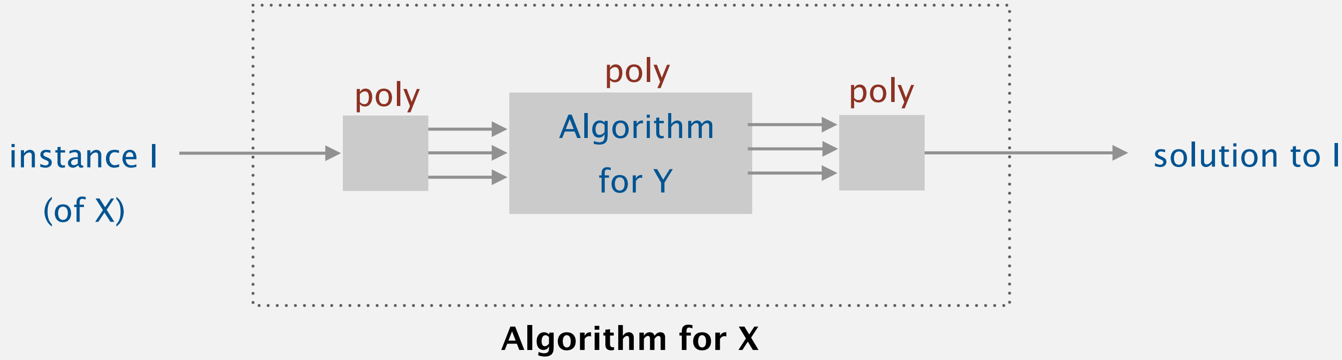
Ex. Almost all of the reductions we've seen so far. [Which ones weren't?]

Also common: polynomial-time reduction.

Polynomial-time reductions

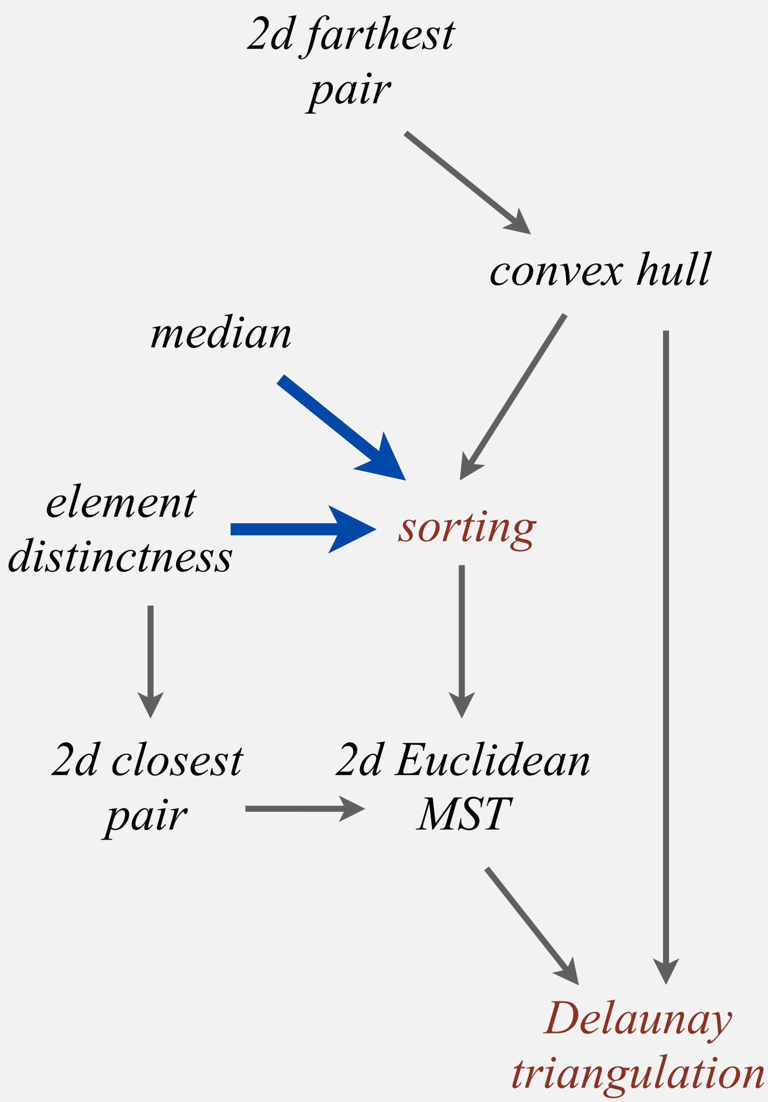
Problem X poly-time (Cook) reduces to problem Y if X can be solved with:

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

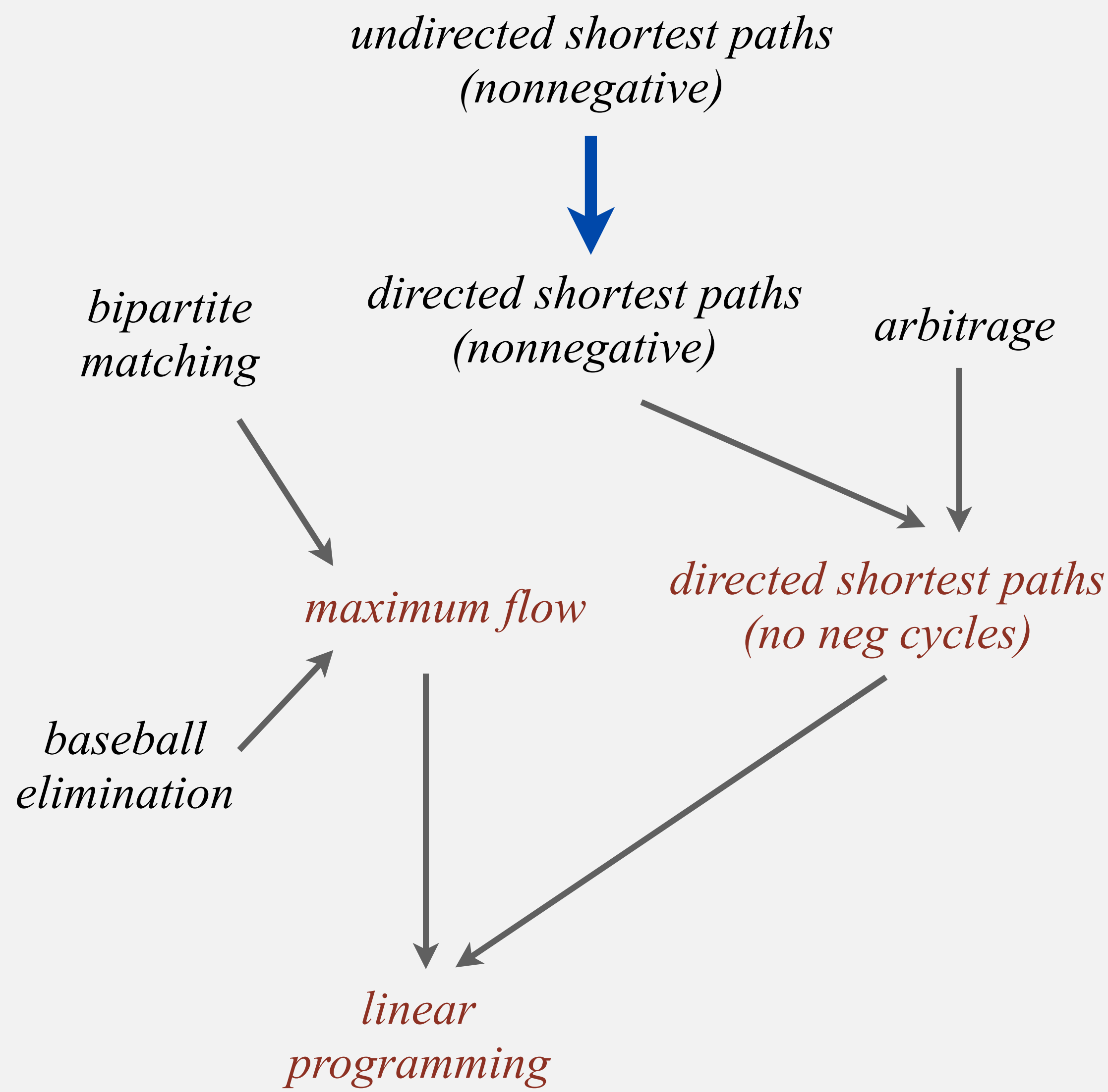


Some reductions involving familiar problems

computational geometry



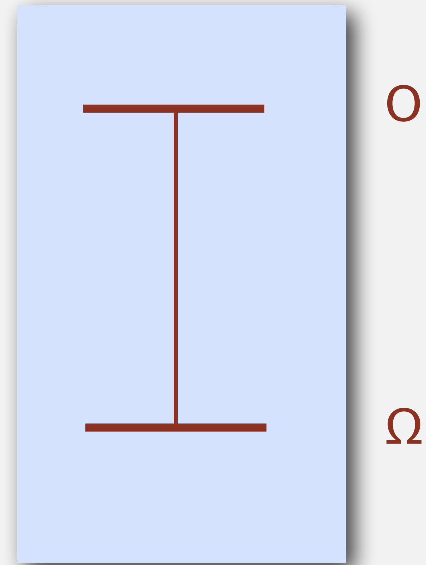
combinatorial optimization



Big O and Big Omega reminders

Can bound a problem above and below.

- Develop an algorithm (big O).
- Prove a lower bound (big Ω).



Worst case performance
for optimal algorithm

Gap?

- Lower the upper bound (discover a new algorithm).
- Raise the lower bound (more difficult).

Example: Sorting.

- Insertion sort tells us that sorting is $O(N^2)$.
- Decision tree argument tells us that sorting is $\Omega(N \log N)$.

Example: Hamilton Path.

- Brute force: $O(N!)$ different permutations to check.

Uses of Reduction

Proving a problem Π is O(f(N))

- Prove linear-time reduction to a problem that is O(f(N)).
- Examples:
  - Convex hull reduces to sorting (Graham scan). N log N N log N
  - Bipartite matching reduces to max-flow.
  - Baseball elimination reduces to max-flow.
  - Currency arbitrage reduces to negative cycle detection.
  - Wordnet's shortest ancestral path reduces to directed shortest paths.
  - Seam carving reduces to directed shortest paths.

Developing code to solve problems

- Write a translation routine from Π.

Proving a problem Π is Ω(f(N))

- Stay tuned!

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

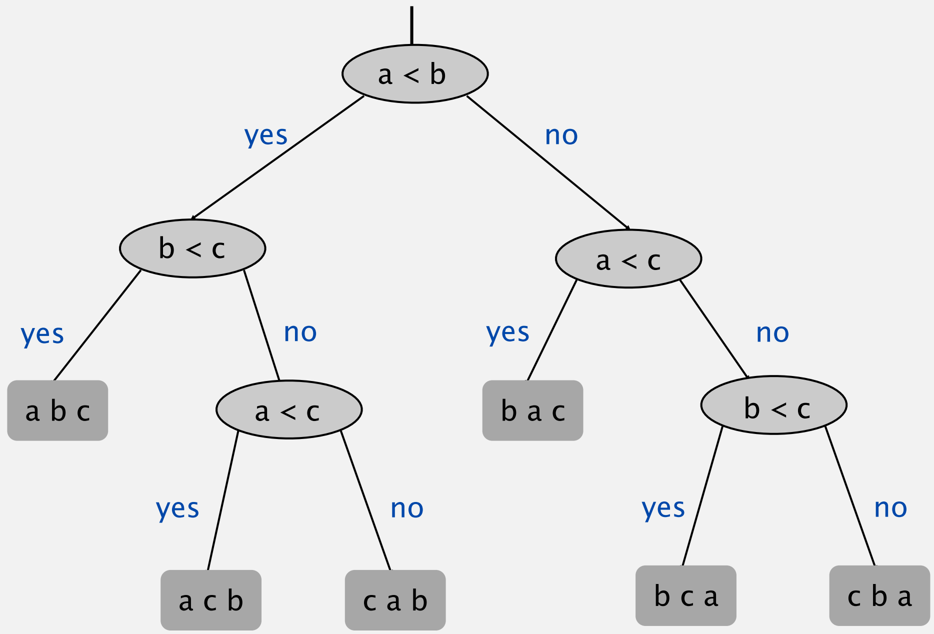
Reductions and Tractability

- *linear reductions*
- *theoretical uses of linear reductions*
- *tractability, P, and NP*

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

Simple lower bound through reductions example

Goal. Construct a BST in linear time from a set of N randomly ordered elements using compare operations.

model of computation: compares

Proposition. Linear time BST construction on random elements is impossible.

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

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

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

A bit counter-intuitive at first.

Proposition. Sorting linear-time reduces to BST construction.

Pf. Construct BST from elements. Perform an in-order traversal.

Linear time??

Linear time

Contradiction. If construction is linear, the reduction provides a linear time sorting algorithm, which is impossible to do only using compares.

Linear-time reductions

Suppose problem X linear-time reduces to problem Y, i.e. solvable with:

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

or any other function of N

Establish lower bound:

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

Example:

X: Sorting

Y: BST Construction

X reduces to 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.

Uses of Reduction

Proving a problem Π is $O(f(N))$

- Prove linear-time reduction to a problem that is $O(f(N))$.

Developing code to solve problems

- Write a translation routine from Π.

Proving a problem Π is $\Omega(f(N))$

- Prove linear-time reduction **from** a known $\Omega(f(N))$ problem.

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:

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

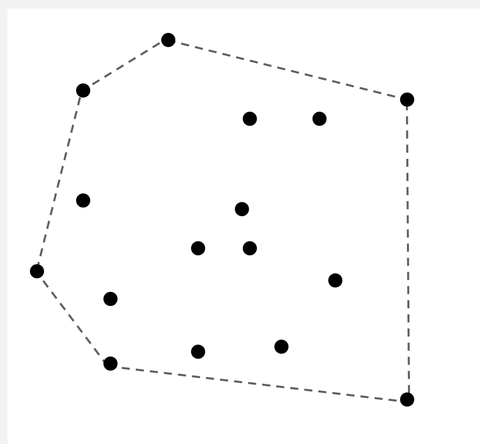
Proposition. Sorting linear-time reduces to convex hull.

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

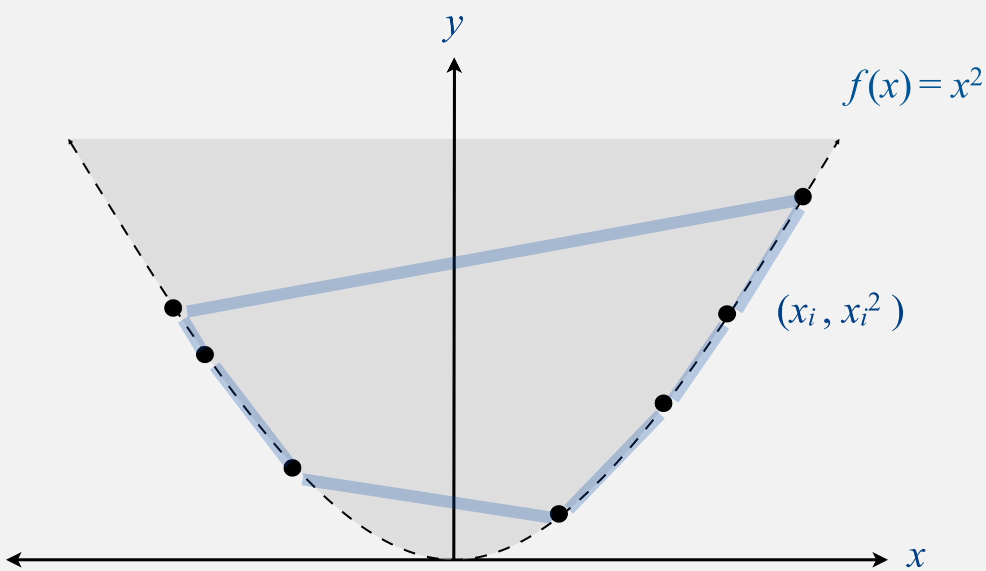
linear or
quadratic tests

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



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.

Uses of Reduction

Proving a problem Π is O(f(N))

- Prove linear-time reduction to a problem that is O(f(N)).

Developing code to solve problems

- Write a translation routine from Π.

Proving a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a known Ω(f(N)) problem.

Suggest that a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a problem suspected to be Ω(f(N)).

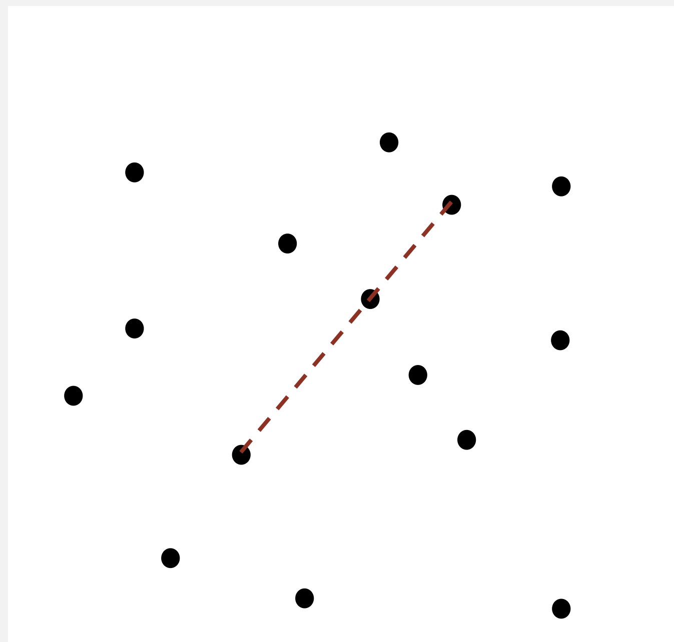
Lower bound for 3-COLLINEAR

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 that all 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 that all lie on the same line?

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

Pf. [next two slides]

(Not covered in class)

lower-bound mentality:
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. Any algorithm for *3-SUM* requires $\Omega(N^2)$ steps.

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

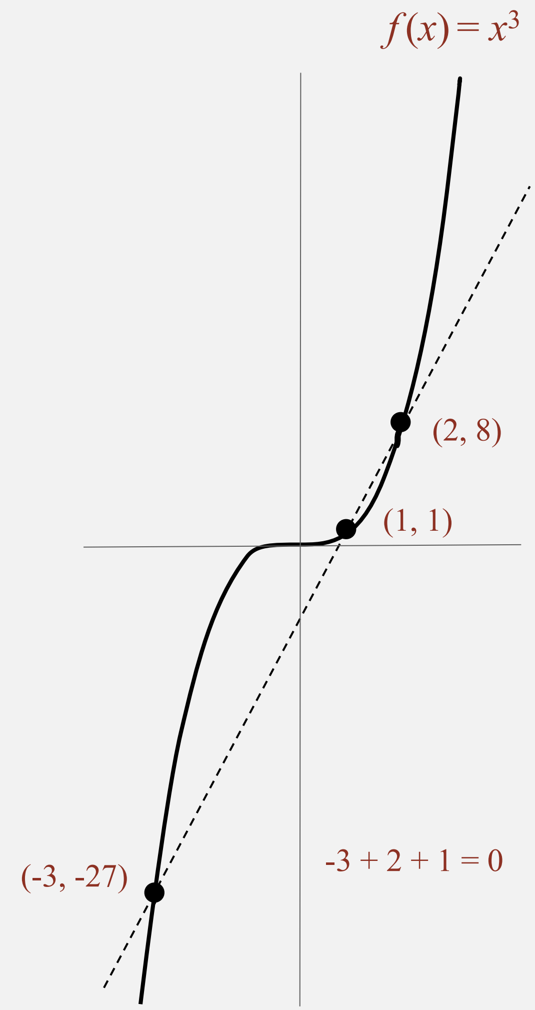
your $N^2 \log N$ algorithm was pretty good

3-SUM linear-time reduces to 3-COLLINEAR

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

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

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.



3-SUM linear-time reduces to 3-COLLINEAR

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

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

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), and (c, c^3) are collinear iff:

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

Uses of Reduction

Proving a problem Π is O(f(N))

- Prove linear-time reduction to a problem that is O(f(N)).

Developing code to solve problems

- Write a translation routine from Π.

Proving a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a known Ω(f(N)) problem.

Suggest that a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a problem suspected to be Ω(f(N)).

Prove that two problems Π and X have the same complexity, i.e. are Θ(f(N))

- Prove that Π linear-time reduces to X
- Prove that X linear-time reduces to Π

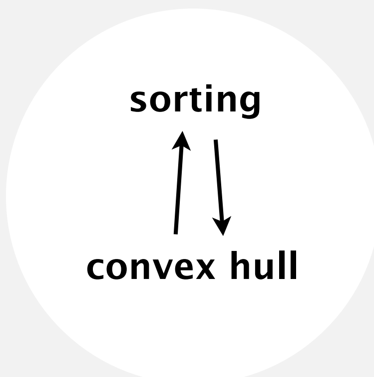
Have same worst case order of growth, given by unknown function!

Classifying problems: summary

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!



Linear algebra reductions

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

Brute force. N^3 flops.

$$\begin{array}{cc} & \\ & \\ \textbf{row i} & \\ & \\ & \end{array} \begin{bmatrix} 0.1 & 0.2 & 0.8 & 0.1 \\ 0.5 & 0.3 & 0.9 & 0.6 \\ 0.1 & 0.0 & 0.7 & 0.4 \\ 0.0 & 0.3 & 0.3 & 0.1 \end{bmatrix} \times \begin{array}{c} \begin{array}{cccc} & & \textbf{column j} & \end{array} \\ \begin{bmatrix} 0.4 & 0.3 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.0 & 0.6 \\ 0.0 & 0.0 & 0.4 & 0.5 \\ 0.8 & 0.4 & 0.1 & 0.9 \end{bmatrix} \end{array} = \begin{array}{c|c} & \begin{array}{cccc} & & \textbf{j} & \end{array} \\ \begin{array}{c} \\ \textbf{i} \\ \\ \\ \end{array} & \begin{bmatrix} 0.16 & 0.11 & 0.34 & 0.62 \\ 0.74 & 0.45 & 0.47 & 1.22 \\ 0.36 & 0.19 & 0.33 & 0.72 \\ 0.14 & 0.10 & 0.13 & 0.42 \end{bmatrix} \end{array}$$

$0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47$

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	$\lvert A \rvert$	MM(N)
system of linear equations	$Ax = b$	MM(N)
LU decomposition	$A = L\,U$	MM(N)
least squares	$\min \lVert Ax - b \rVert_2$	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	$N^{2.3727}$
?	?	$N^{2+\varepsilon}$

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

Uses of reduction

Proving a problem Π is O(f(N))

- Prove linear-time reduction to a problem that is O(f(N)).

Developing code to solve problems

- Write a translation routine from Π.

Proving a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a known Ω(f(N)) problem.

Suggest that a problem Π is Ω(f(N))

- Prove linear-time reduction **from** a problem suspected to be Ω(f(N)).

Prove that two problems Π and X have the same complexity, i.e. are Θ(f(N))

- Prove that Π linear-time reduces to X
- Prove that X linear-time reduces to Π

Unknown function!

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Reductions and Tractability

- *linear reductions*
- *theoretical uses of linear reductions*
- *tractability, P, and NP*

Intractability

Desiderata. Understand which problems are easy, and which are hard.

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

- Run-time grows faster than N^k.

Tractable.

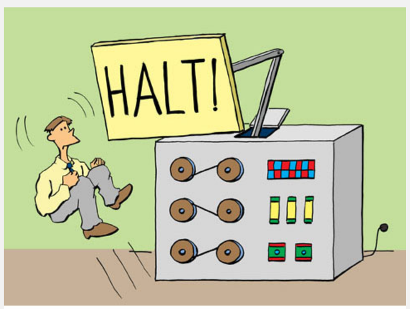
- Comparison sorting: $O(N^2)$
- Collinear: $O(N^3)$

Intractable.

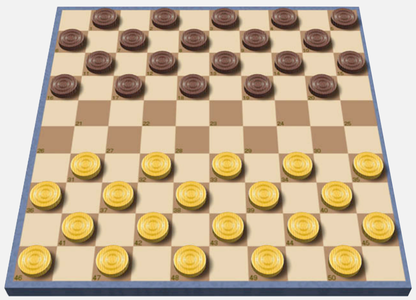
input size = c + lg K

- 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?

using forced capture rule



Alan designed the perfect computer



Unknown difficulty

Decision problems of unknown difficulty.

- Does there exist a Hamilton path in a graph?
- Does there exist a path a traveling salesman can take that is of total weight less than W?
- Does there exist a set of inputs for a circuit such that the output is true?
- Given a set of axioms, can we prove mathematical theorem X?

Optimization problems of unknown difficulty.

- What is the minimum weight path for a traveling salesman?
- Given a set of basic axioms, what is the shortest proof?

Amazing fact:

- A solution to ANY of these problems provides a solution to all of them.
  - Every one of these problems reduces to every other problem.
  - Nobody knows whether or not these problems can be solved in polynomial time. Does P = NP?

Decision problems vs. function problems

Easier to reason about, output is only 1 bit.

Decision Problem

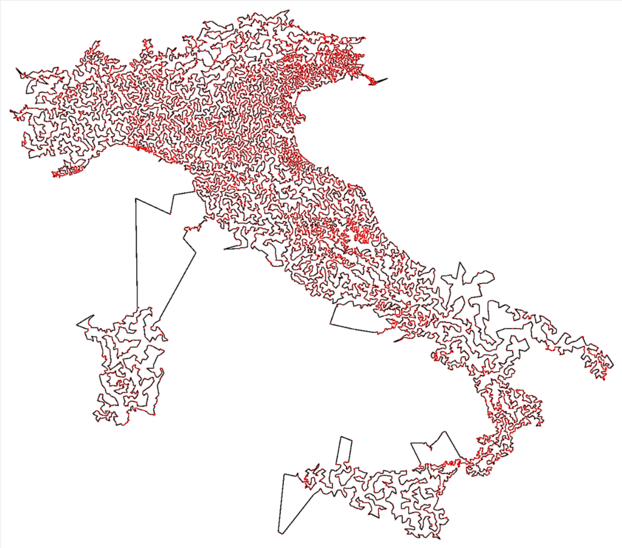
- Given some input, gives "yes" or "no" as answer.

Function problem

- Given some input, give some output as an answer.

Examples:

- Decision problems
  - Does a TSP tour exist of length < M?
  - Is N the product of two primes?
- Function problems
  - What is the minimal weight TSP tour?
  - What are the factors of N?
  - What is the sorted version of X?



TSP Tour of Italy's Cities

Solving function problems via decision problems

TSP

- What is the minimal weight TSP tour?
- Does a TSP tour exist of length < M?

- Example
  - Does a TSP tour exist of length < 20000?
  - Yes. What about < 10000?
  - Yes. What about < 5000?
  - No. What about < 7500?
  - ...

Full discussion beyond the scope of our course.

The class P

Classic definition. Book defines P as a class of “search problems”.

A problem is in P if

All problems in P are tractable!

- It is a decision problem.
- It can be solved in $O(N^k)$ time.
  - $O(N^k)$ - Worst case order of growth is $\leq N^k$.
  - N is number of bits needed to specify input.

Example

- Is vertex X reachable from vertex S?
  - Total bits used for adjacency list representation: $N = c_1E + c_2V$
  - DFS, worst case order of growth: E+V
  - In terms of big O: O(E+V) = O(N)

Easy as P

Why $O(N^k)$?

- P seems rather generous.
- $O(N^k)$ closed under addition, multiplication and polynomial reduction.
  - Consecutively run two algorithms in P, still in P.
  - Run an algorithm N times, still in P.
  - Reduce to a problem Π in P, then Π is in P.
- Exponents for practical problems are typically small.

The class NP

A problem is in NP if

Also called a certificate.

- It is a decision problem.
- If answer is "Yes", a proof exists that can be verified in polynomial time.
  - NP: Does a TSP tour exist of length less than 1000?
  - Not NP: Is a given TSP tour optimal? This is in a class called co-NP.
  - Not NP: What is the optimal TSP tour? Defining NP in terms of "search problems" puts this problem into NP.

- Stands for "non-deterministic polynomial"
  - Name is a bit confusing. Don't worry about it.

- **Most important detail: Verifiable in Polynomial Time.**
  - "In an ideal world it would be renamed P vs VP" - Clyde Kruskal

> *"Joseph Kruskal [inventor of Kruskal's algorithm] should not be confused with his two brothers Martin Kruskal(1925–2006; co-inventor of solitons and of surreal numbers) and William Kruskal(1919–2005; developed the Kruskal-Wallis one-way analysis of variance), or his nephew Clyde Kruskal." -Dbenbenn*

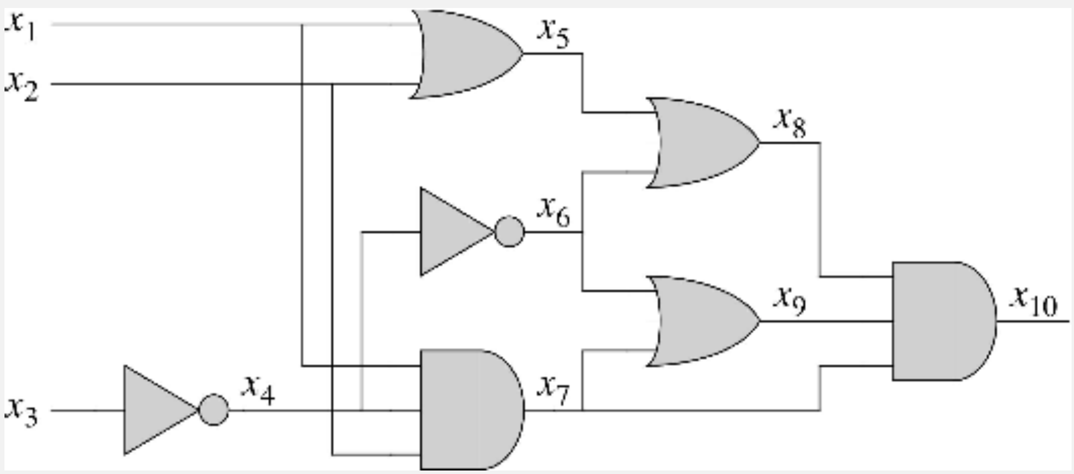
http://en.wikipedia.org/wiki/Joseph_Kruskal

Verification example

Verifiable in polynomial time

- Circuit satisfiability: Do there exist x_1, x_2, x_3 such that x_{10} is true?
  - If true, easy proof is x_1=true, x_2=true, x_3=false.
  - Linear time simulation with this input yields x_{10}=true.

Verification takes polynomial time.



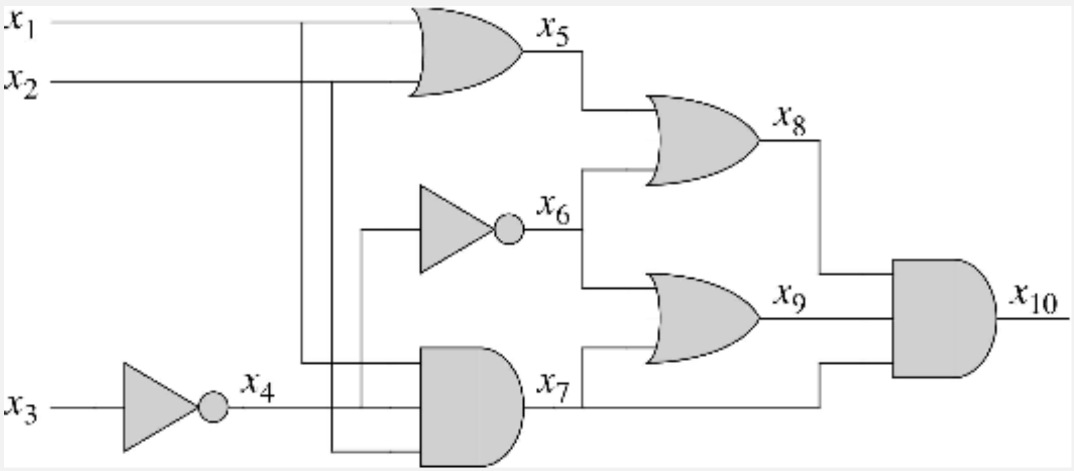
Not verifiable in polynomial time

- Checkers: From a given checkerboard position, is there some sequence of moves such that player 1 wins?
  - Certificate cannot be easily verified.

Solving the circuit satisfiability problem

Solving circuit satisfiability

- 2^N possible inputs.
- Brute force solution is exponential.
- Best known solution is exponential.



NP

NP includes a vast number of interesting problems.

- Hand-wavy reason: Many (most?) practical problems can be analyzed in terms of interesting NP decision problems.
- Example: Managing an airline
  - Can we assign planes to our routes such that we use < N gallons/year?
- Example: Destroying the global e-commerce system.
  - Given Z, are there two primes such that X*Y = Z.

See COS432

- Counter-example?
  - Is move X better than move Y in this chess game on N^2 board?

Completeness (short detour)

Completeness

- Let Q be a class of problems and let π be a specific problem.

- π is Q-Complete if
  - π is in Q.
  - Everything in Q time reduces to π [π solves any problem in Q].

many glossed over details!

- If a solution is known, can use π as a tool to solve any problem in Q.

NP-complete

NP-complete

- A problem π is NP-complete if: many glossed over details!
  - π is in NP.
  - All problems in NP poly-time reduce to π.

- Solution to an NP-complete problem would be a key to the universe!

Two questions

- Are there any NP-complete problems?
- Do we know how to solve any of them?

Existence of an NP complete problem

3SAT

Also in NP!

- Cook (71), Levin (73) proved every NP problem poly-time reduces to 3SAT.
  - 3SAT is at least as hard as every other problem in NP.
  - A solution to 3SAT provides a solution to every problem in NP.
  - Every problem in NP is $O(F_{3SAT}(N))$.

- Does there exist a truth value for boolean variables that obeys a set of 3-variable disjunctive constraints: (x1 || x2 || !x3) && (x1 || !x1 || x1)

Stephen Cook

Leonid Levin

Existence of an NP complete problem

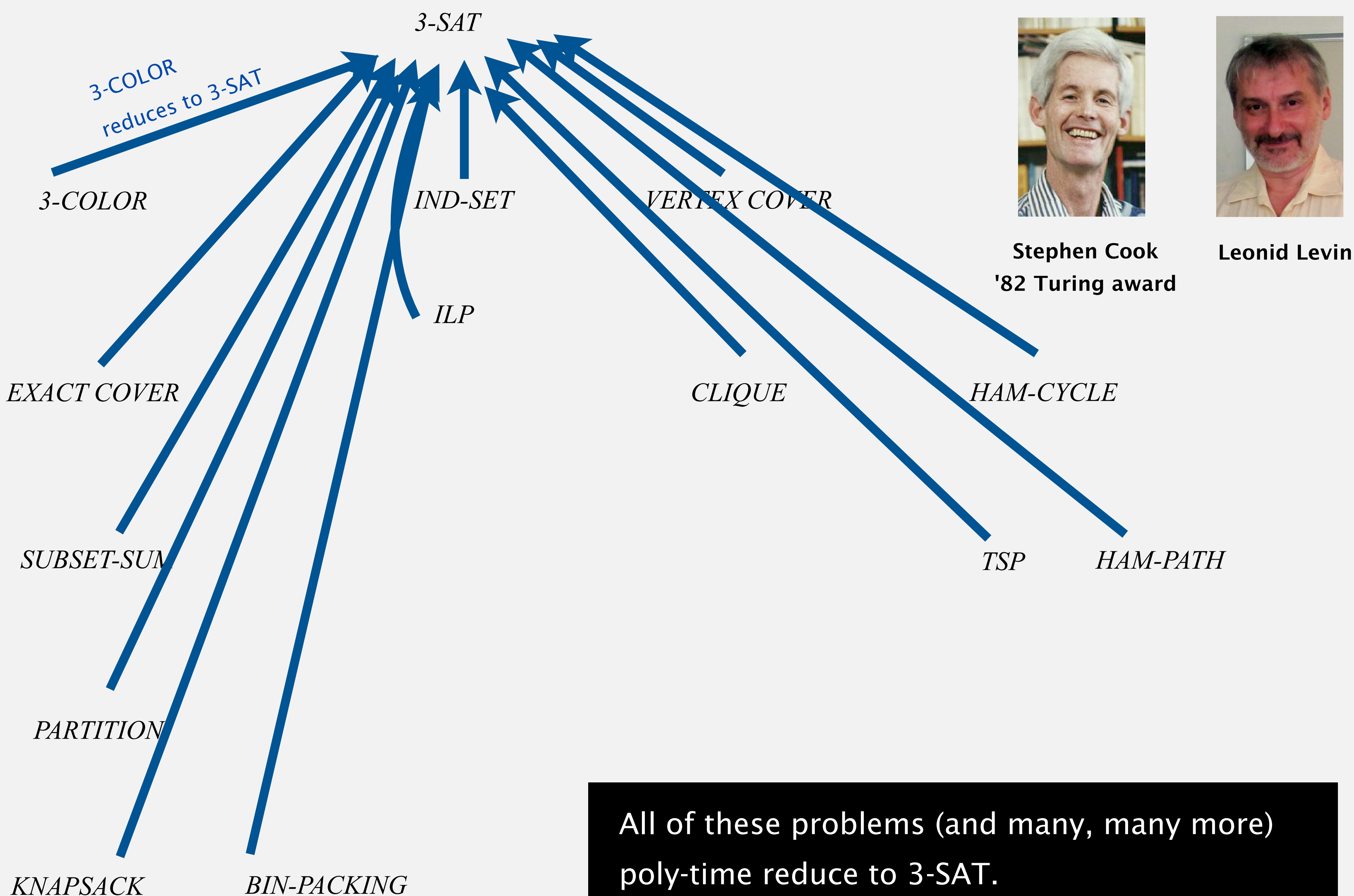
Rough idea of Cook-Levin theorem

- Create giant (!!) boolean logic expression that represents entire state of your computer at every time step.
- If solution takes polynomial time, boolean logic circuit is polynomial in size.
- Example boolean logic variable: True if 57173rd bit of memory is true and we're on line 38 of code during cycle 7591872 of execution.

Stephen Cook

Leonid Levin

Implications of Cook-Levin theorem



Stephen Cook
'82 Turing award

Leonid Levin

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

3SAT

Great, 3SAT solves most well defined problems of general interest!

Can we solve 3SAT efficiently?

- Nobody knows how to solve 3SAT efficiently.
- Nobody knows if an efficient solution exists.
  - Unknown if 3SAT is in P.

Other NP Complete problems?

- Are there other keys to this magic kingdom?

NP Complete

There are more

- Dick Karp (72) proved that 3SAT reduces to 21 important NP problems.
  - Example: A solution to TSP provides a solution to 3SAT.
  - All of these problems join 3SAT in the NP Complete club.
  - These 21 problems are $\Omega(F_{3SAT}(N))$.
- Proof applies only to these 21 problems. Each was its own special case.

Dick Karp

More poly-time reductions from 3-satisfiability

3-SAT → *3-COLOR*, *IND-SET*, *VERTEX COVER*

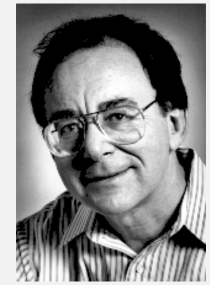
3-COLOR → *EXACT COVER* → *SUBSET-SUM* → *PARTITION* → *KNAPSACK*, *BIN-PACKING*

IND-SET → *ILP*

3-SAT reduces to ILP

VERTEX COVER → *CLIQUE*, *HAM-CYCLE*

HAM-CYCLE → *TSP*, *HAM-PATH*

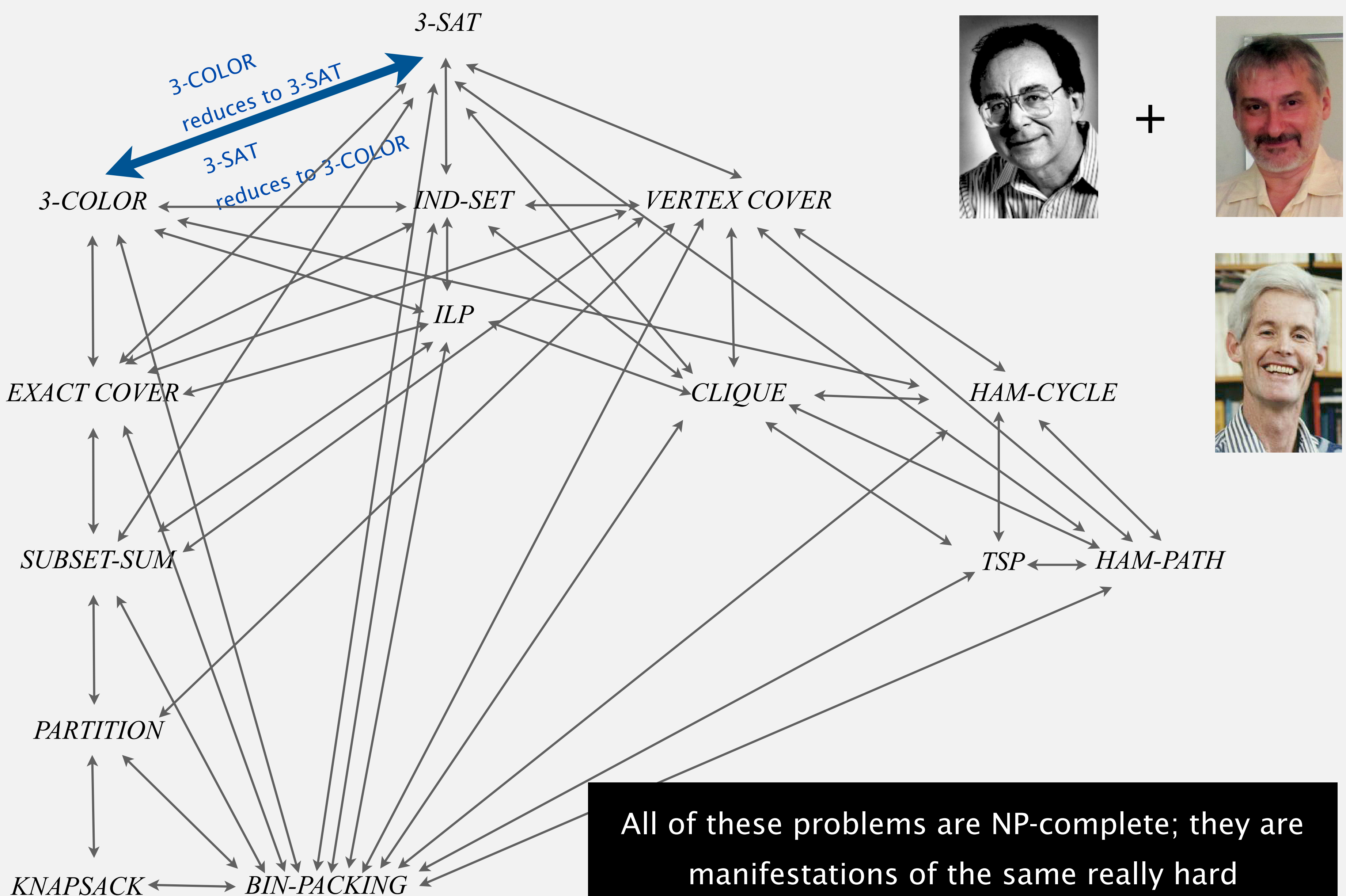


Dick Karp
'85 Turing award

something interesting to say about big o or big omega here?

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

Implications of Karp + Cook-Levin



All of these problems are NP-complete; they are manifestations of the same really hard problem.

Summary

Cook and Levin

- Every NP problem is $O(F_{3SAT}(N))$.
- 3SAT is in NP and solves every NP problem, i.e. it is NP-Complete.

Karp

- 21 specific NP problems are $\Omega(F_{3SAT}(N))$.
- These 21 problems solve 3SAT.
- All of these problems are also therefore NP-Complete.

Later work

- Thousands of practical NP problems are also $\Omega(F_{3SAT}(N))$.
- All of these problems are also therefore NP-Complete.

How to tell if your problem is NP Complete?

- Prove that it is in NP [easy].

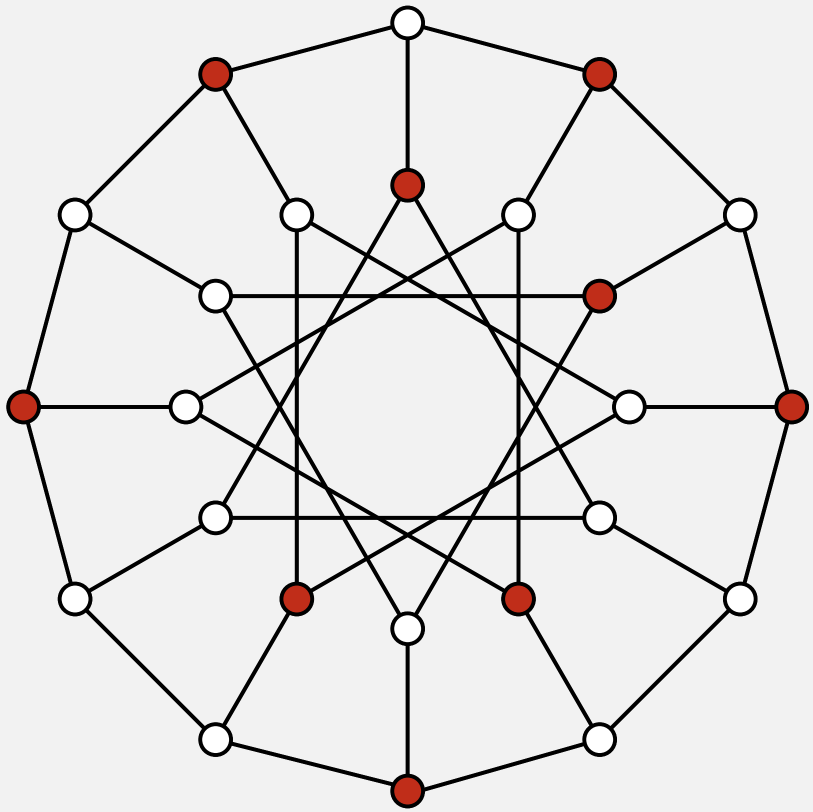
- Prove that **some** NP Complete problem reduces to your problem [tricky!]

Independent set

An independent set is a set of vertices, no two of which are adjacent.

IND-SET. Given graph G and an integer k, find an independent set of size k.

k = 9



Applications. Scheduling, computer vision, clustering, ...

3-satisfiability reduces to independent set

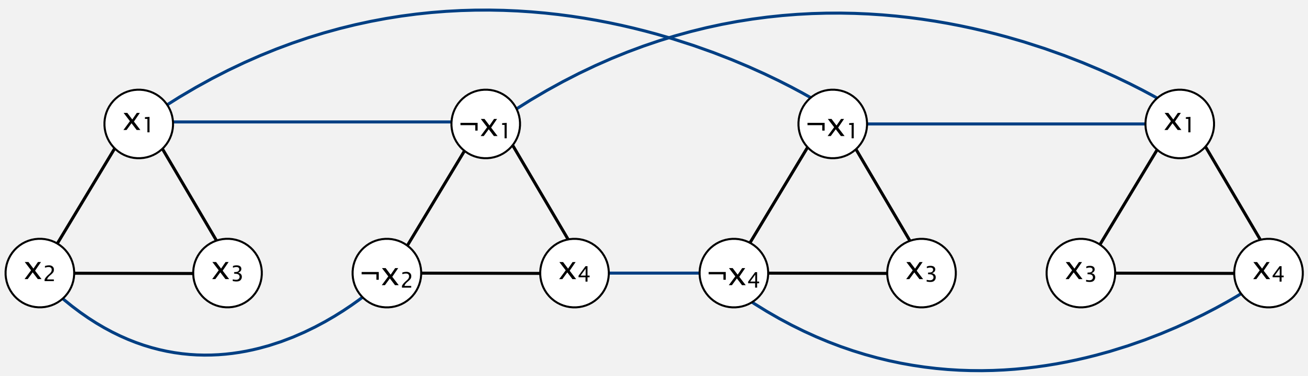
Proposition. *3-SAT* poly-time reduces to *IND-SET*. ←

lower-bound mentality:
if I could solve IND-SET efficiently,
I could solve 3-SAT efficiently

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



$$\Phi \;=\; (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

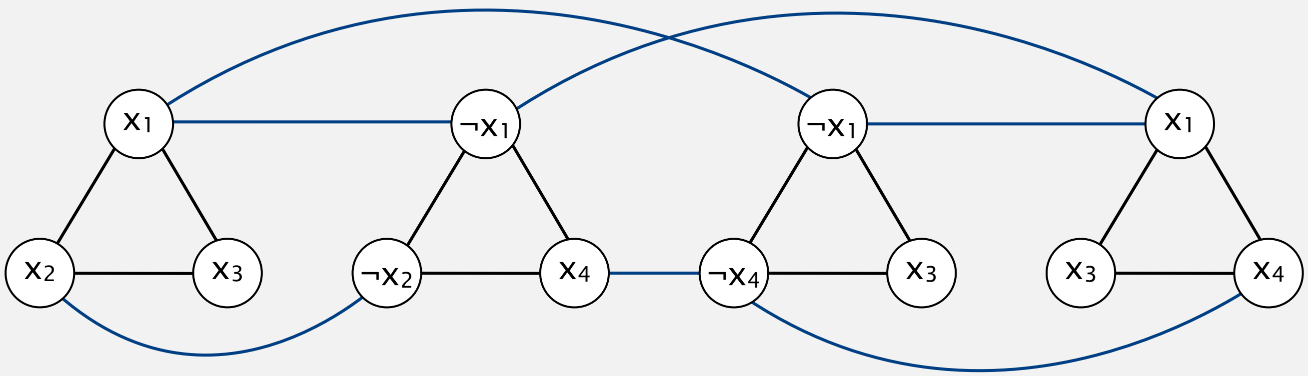
3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



$$\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

- Φ satisfiable $\Rightarrow$ G has independent set of size k.

for each of k clauses, include in independent set one vertex corresponding to a true literal

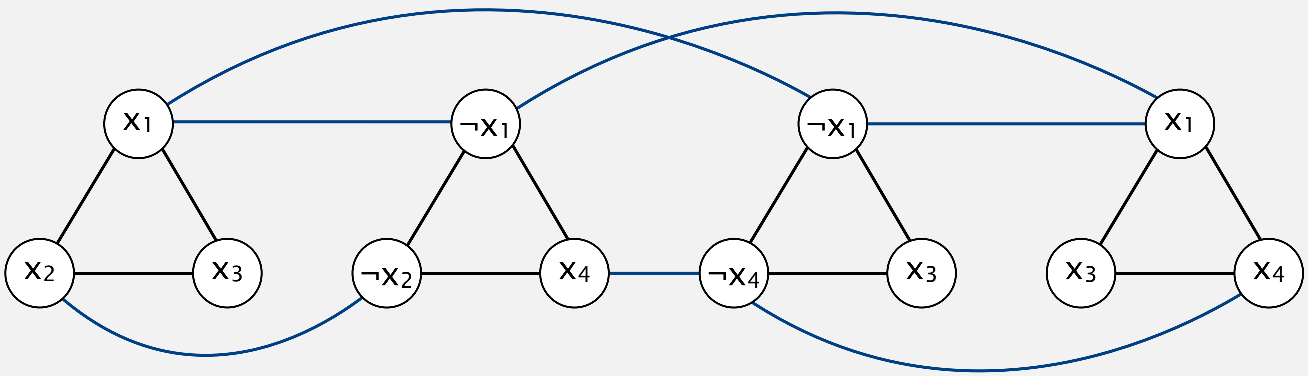
3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Pf. Given an instance Φ of *3-SAT*, create an instance G of *IND-SET*:

- For each clause in Φ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

k = 4



$$\Phi \;=\; (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

- Φ satisfiable $\Rightarrow$ G has independent set of size k.
- G has independent set of size k $\Rightarrow$ Φ satisfiable.

set literals corresponding to k vertices in independent set to true

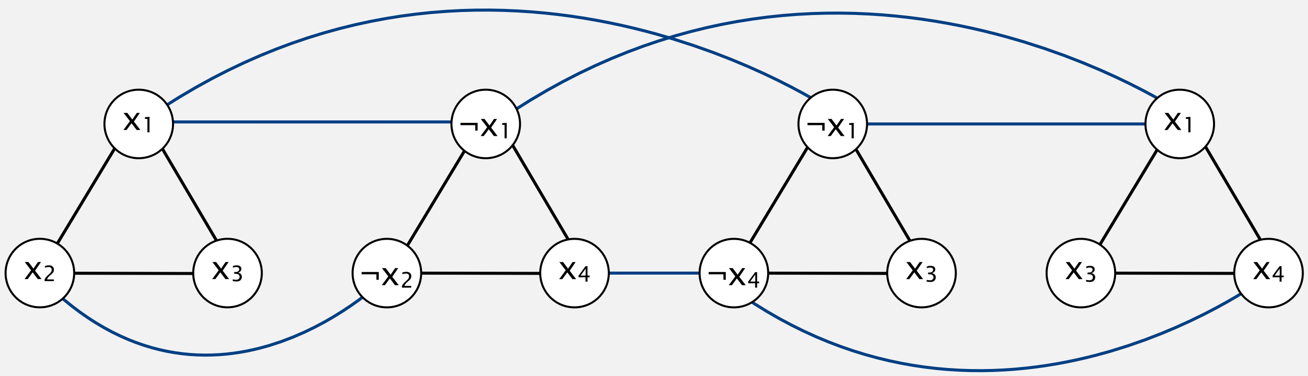
(set remaining literals in any consistent manner)

3-satisfiability reduces to independent set

Proposition. *3-SAT* poly-time reduces to *IND-SET*.

Implication. Assuming *3-SAT* is intractable, so is *IND-SET*.

k = 4

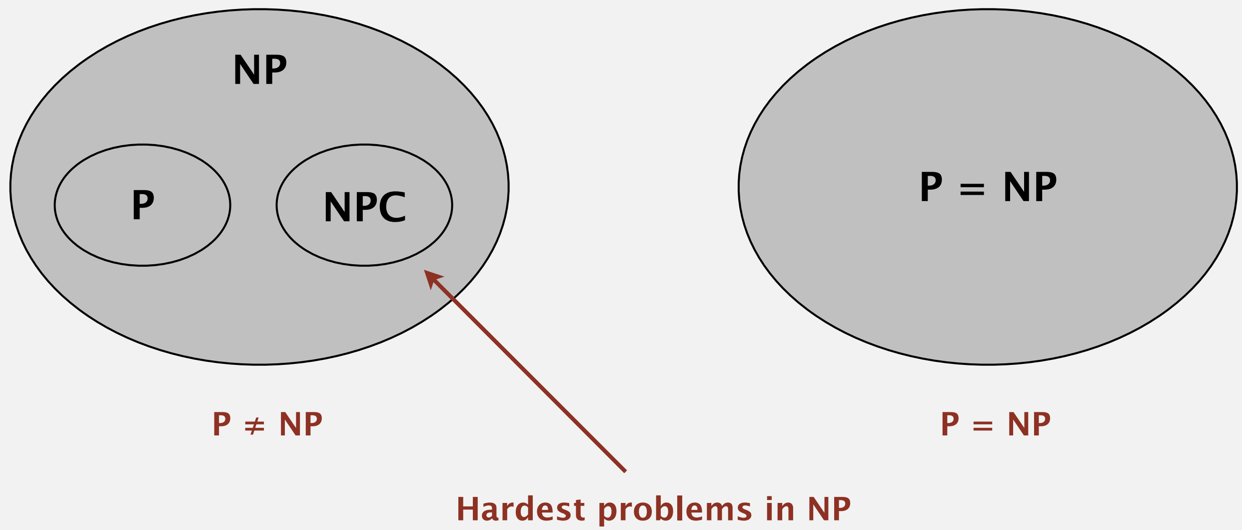


$$\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$

P = NP?

Does P = NP?

- Equivalently: Is any NP Complete problem also in P?
- Equivalently: Efficiently verifiable ⇒ efficiently solvable?



Reminder: NP may as well have been called VP for "Verifiable in Polynomial Time"

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, element distinctness, convex hull, closest pair, ...
quadratic	N^2	?
⋮	⋮	⋮
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, ...
⋮	⋮	⋮
NP-complete	probably not N^b	3-SAT, IND-SET, ILP, ...

Good news. Can put many problems into equivalence classes.

Complexity zoo

Complexity class. Set of problems sharing some computational property.

http://qwiki.stanford.edu/index.php/Complexity_Zoo

Bad news. Lots of complexity classes.