6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
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”.

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

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
**Def.** Problem \(X\) reduces to problem \(Y\) if you can use an algorithm that solves \(Y\) to help solve \(X\).

\[
\text{Cost of solving } X = \text{ total cost of solving } Y + \text{ cost of reduction.}
\]

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. $N \log N + 1$. 
**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. $N \log N + N$. 
**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 \( X \).
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.

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

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

**Pf.** Graham scan algorithm.

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

![Convex hull diagram]

- 1251432
- 2861534
- 3988818
- 8111033
- 13546464
- 89885444
- 43434213
- 34435312

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

- baseball elimination
- mincut
- bipartite matching
- undirected shortest paths (nonnegative)
- directed shortest paths (no neg cycles)
- arbitrage
- shortest paths (in a DAG)
- seam carving
- maxflow
- directed shortest paths (nonnegative)
- assignment problem
- directed shortest paths (no neg cycles)
- linear programming
6.5 Reductions

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

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

Pf. [see next slide]

Implication. Any convex hull algorithm requires $\Omega(N \log N)$ ops.

lower-bound reasoning: I can't sort in linear time, so I can't solve convex hull in linear time either
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 \( \{(x, y) : y \geq x^2\} \) 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?
A2. [easy way] Linear-time reduction from sorting.
Our lower bound proof strategy for \textsc{Convex-Hull} would work even if:

A. Our reduction invoked \textsc{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 \textsc{Sorting} = total cost of \textsc{Convex-Hull} + cost of reduction.
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
**Bird’s-eye view**

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

<table>
<thead>
<tr>
<th>complexity</th>
<th>order of growth</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$N$</td>
<td>$\text{min, max, median, Burrows-Wheeler transform, ...}$</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>$\text{sorting, element distinctness, closest pair, Euclidean MST, ...}$</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>?</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>exponential</td>
<td>$c^N$</td>
<td>?</td>
</tr>
</tbody>
</table>

**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 $\Omega(N \log N)$.

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 $T(N)$.

\[ X = \text{sorting} \quad \leftrightarrow \quad Y = \text{element distinctness} \]

\[ \quad \leftrightarrow \quad \]

\[ \text{integer multiplication} \quad \leftrightarrow \quad \text{integer division} \]

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.
Integer arithmetic reductions

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

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

<table>
<thead>
<tr>
<th>problem</th>
<th>arithmetic</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer multiplication</td>
<td>$a \times b$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer division</td>
<td>$a / b, a \mod b$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer square</td>
<td>$a^2$</td>
<td>$M(N)$</td>
</tr>
<tr>
<td>integer square root</td>
<td>$\lfloor \sqrt{a} \rfloor$</td>
<td>$M(N)$</td>
</tr>
</tbody>
</table>

integer arithmetic problems with the same complexity as integer multiplication

**Q.** Is brute-force algorithm optimal?
## History of complexity of integer multiplication

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>brute force</td>
<td>$N^2$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba</td>
<td>$N^{1.585}$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom–3, Toom–4</td>
<td>$N^{1.465}$, $N^{1.404}$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom–Cook</td>
<td>$N^{1 + \varepsilon}$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage–Strassen</td>
<td>$N \log N \log \log N$</td>
</tr>
<tr>
<td>2007</td>
<td>Fürer</td>
<td>$N \log N 2^{\log^*N}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$N$</td>
</tr>
</tbody>
</table>

**Number of bit operations to multiply two $N$-bit integers**

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

![3-sum](image1.png)

![3-collinear](image2.png)
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]

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

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

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

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

**We hope to prove:** If $a$, $b$, and $c$ are distinct, then $a + b + c = 0$ if and only if $(a_1, f(a_1)), (b_2, f(b_2)), \ldots, (c_N, f(c_N))$ are collinear.
**3-SUM linear-time reduces to 3-COLLINEAR**

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

- **3-SUM instance:** \(x_1, x_2, \ldots, x_N\).
- **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 \((a, a^3), (b, b^3), (c, c^3)\) are collinear iff:

\[
\begin{vmatrix}
a & a^3 & 1 \\
b & b^3 & 1 \\
c & c^3 & 1 \\
\end{vmatrix} = (a - b)(b - c)(c - a)(a + b + c)
\]
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 $\Omega(n^2)$.

---

[1] This work is supported in part by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO). S. Pettie is supported by NSF grants CCF-1217338 and CNS-1318294 and a grant from the US-Israel Binational Science Foundation.
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.
6.5 ReducTIONS

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

Frustrating news. Very few successes.
A core problem: satisfiability

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

**Ex.**

\[
\begin{array}{cccc}
\neg x_1 & \text{or} & x_2 & \text{or} & x_3 & = & \text{true} \\
x_1 & \text{or} & \neg x_2 & \text{or} & x_3 & = & \text{true} \\
\neg x_1 & \text{or} & \neg x_2 & \text{or} & \neg x_3 & = & \text{true} \\
\neg x_1 & \text{or} & \neg x_2 & \text{or} & \text{or} & x_4 & = & \text{true} \\
\neg x_2 & \text{or} & x_3 & \text{or} & x_4 & = & \text{true} \\
\end{array}
\]

instance I

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

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 ≠ NP**). 3-SAT is intractable (no poly-time algorithm).
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.

\[
\begin{align*}
3x_1 + 5x_2 + 2x_3 + x_4 + 4x_5 & \geq 10 \\
5x_1 + 2x_2 + 4x_4 + 1x_5 & \leq 7 \\
x_1 + x_3 + 2x_4 & \leq 2 \\
3x_1 + 4x_3 + 7x_4 & \leq 7 \\
x_1 + x_4 & \leq 1 \\
x_1 + x_3 + x_5 & \leq 1 \\
\text{all } x_i & = \{ 0, 1 \}
\end{align*}
\]

**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.

\[
\begin{align*}
\neg x_1 & \text{ or } x_2 & \text{ or } x_3 & = \text{ true} \\
 x_1 & \text{ or } \neg x_2 & \text{ or } x_3 & = \text{ true} \\
\neg x_1 & \text{ or } \neg x_2 & \text{ or } \neg x_3 & = \text{ true} \\
\neg x_1 & \text{ or } \neg x_2 & \text{ or } & = \text{ true} \\
\neg x_2 & \text{ or } x_3 & \text{ or } x_4 & = \text{ true} \\
\end{align*}
\]

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

\[
\begin{align*}
(1-x_1) & + x_2 & + x_3 & \geq 1 \\
x_1 & + (1-x_2) & + x_3 & \geq 1 \\
(1-x_1) & + (1-x_2) & + (1-x_3) & \geq 1 \\
(1-x_1) & + (1-x_2) & + & + x_4 & \geq 1 \\
(1-x_2) & + x_3 & + x_4 & \geq 1 \\
\end{align*}
\]

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

\[
\begin{align*}
\neg x_1 & \lor x_2 \lor x_3 = true \\
x_1 & \lor \neg x_2 \lor x_3 = true \\
\neg x_1 & \lor \neg x_2 \lor \neg x_3 = true \\
\neg x_1 & \lor \neg x_2 \lor x_4 = true \\
\neg x_2 & \lor x_3 \lor x_4 = true
\end{align*}
\]

<table>
<thead>
<tr>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1 x_2 x_3 x_4</td>
<td>T T F T</td>
</tr>
</tbody>
</table>

**Ex 2. Factor.** Given an N-bit integer \( x \), find a nontrivial factor.

<table>
<thead>
<tr>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>147573952589676412927</td>
<td>193707721</td>
</tr>
</tbody>
</table>

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.**

```
NP

P
NPC

P ≠ NP

P = NP

P = NP
```
Implications of Cook-Levin theorem

3-SAT

3-COLOR reduces to 3-SAT

3-COLOR

IND-SET

ILP

VERTTEX-COVER

CLIQUE

HAM-CYCLE

3-COLOR

EXACT-COVER

SUBSET-SUM

PARTITION

KNAPSACK

BIN-PACKING

Stephen Cook

'82 Turing award

Leonid Levin

All of these problems (and many, many more) poly-time reduce to 3-SAT.
All of these problems are \textbf{NP–COMPLETE}; they are manifestations of the same really hard problem.
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.

<table>
<thead>
<tr>
<th>complexity</th>
<th>order of growth</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$N$</td>
<td>$\text{min, max, median, Burrows-Wheeler transform, ...}$</td>
</tr>
<tr>
<td>linearithmetic</td>
<td>$N \log N$</td>
<td>$\text{sorting, element distinctness, ...}$</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>?</td>
</tr>
<tr>
<td>exponential</td>
<td>$c^N$</td>
<td>?</td>
</tr>
</tbody>
</table>

**Frustrating news.** Huge number of problems have defied classification.
**Desiderata.** Classify problems according to computational requirements.

<table>
<thead>
<tr>
<th>complexity</th>
<th>order of growth</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$N$</td>
<td>$min, max, median,$ Burrows-Wheeler transform, ...</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>sorting, element distinctness, ...</td>
</tr>
<tr>
<td>$M(N)$</td>
<td>?</td>
<td>integer multiplication, division, square root, ...</td>
</tr>
<tr>
<td>$MM(N)$</td>
<td>?</td>
<td>matrix multiplication, $Ax = b$, least square, determinant, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP–complete</td>
<td><em>probably not</em> $N^b$</td>
<td>$3$-Sat, IND-SET, ILP, ...</td>
</tr>
</tbody>
</table>

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

Complexity class. Set of problems sharing some computational property.

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