6.5 REDUCTIONS

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
Overview: introduction to advanced topics

Main topics. [next 2 lectures]

- Reduction: design algorithms, establish lower bounds, classify problems.
- Intractability: problems beyond our reach.
- Combinatorial search: coping with intractability.

Shifting gears.

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

Goals.

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

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- 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>min, max, median, Burrows-Wheeler transform, ...</td>
</tr>
<tr>
<td>linearithmic</td>
<td>N log N</td>
<td>sorting, element distinctness, convex hull, closest pair, ...</td>
</tr>
<tr>
<td>quadratic</td>
<td>N²</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exponential</td>
<td>c^N</td>
<td>?</td>
</tr>
</tbody>
</table>
Desiderata. Classify problems according to computational requirements.

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

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

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

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

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

preprocessing and postprocessing
Reduction

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

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

To find the median of $N$ items:

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

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

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

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

To solve element distinctness on $N$ items:
- Sort $N$ items.
- Check adjacent pairs for equality.

**Cost of solving element distinctness.** $N \log N + N$. 
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
Reduction: design algorithms

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

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

More familiar reduction examples.
- 3-collinear reduces to sorting.
- CPM reduces to topological sort.
- Arbitrage reduces to shortest paths.
- Baseball elimination reduces to maxflow.
- Burrows-Wheeler transform reduces to suffix sort.
- ...

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

*programmer’s version: I have code for $Y$. Can I use it for $X?$*
Convex hull reduces to sorting

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

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

![Convex hull diagram]

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

**Pf.** Graham scan algorithm.

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

---

<table>
<thead>
<tr>
<th>Sorting</th>
<th>Convex Hull</th>
</tr>
</thead>
<tbody>
<tr>
<td>1251432</td>
<td>2861534</td>
</tr>
<tr>
<td>3988818</td>
<td>8111033</td>
</tr>
<tr>
<td>13546464</td>
<td>89885444</td>
</tr>
<tr>
<td>43434213</td>
<td>34435312</td>
</tr>
</tbody>
</table>
Shortest paths on edge-weighted graphs and digraphs

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

**Pf.** Replace each undirected edge by two directed edges.
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

Cost of undirected shortest paths. $E \log V + E$. 
Shortest paths with negative weights

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

![Graph with edges and weights]

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

computational geometry

- 2d farthest pair
- convex hull
- median
- element distinctness
- 2d closest pair
- 2d Euclidean MST
- Delaunay triangulation

combinatorial optimization

- undirected shortest paths (nonnegative)
- directed shortest paths (no neg cycles)
- maximum flow
- arbitration
- directed shortest paths (nonnegative)
- bipartite matching
- baseball elimination
- linear programming
- arbitrage
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.

![Decision tree diagram](image)

**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 $Y$.

argument must apply to all conceivable algorithms

assuming cost of reduction is not too high
Linear-time reductions

**Def.** Problem $X$ linear-time reduces to problem $Y$ if $X$ can be solved with:
- Linear number of standard computational steps.
- Constant number of calls to $Y$.

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

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

**Mentality.**
- If I could easily solve $Y$, then I could easily solve $X$.
- I can’t easily solve $X$.
- Therefore, I can’t easily solve $Y$. 
Lower bound for convex hull

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

allows linear or quadratic tests:

\[ x_i < x_j \text{ or } (x_j - x_i)(x_k - x_i) - (x_j)(x_k - x_i) < 0 \]

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

**Pf.** [see next slide]

![](image)

**Implication.** Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ops.
Proposition. Sorting linear-time reduces to convex hull.

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

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.
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?
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]

**Conjecture.** Any algorithm for *3-SUM* requires $\Omega(N^2)$ steps.

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

---

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

your $N^2 \log N$ algorithm was pretty good
3-SUM linear-time reduces to 3-COLLINEAR

**Proposition.** 3-\textit{SUM} linear-time reduces to 3-\textit{COLLINEAR}.
- \textit{3-SUM instance}: $x_1, x_2, \ldots, x_N$.
- \textit{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:

\[
0 = \begin{vmatrix}
  a & a^3 & 1 \\
  b & b^3 & 1 \\
  c & c^3 & 1 \\
\end{vmatrix} \\
= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3) \\
= (a - b)(b - c)(c - a)(a + b + c)
\]
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.
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
- intractability
Classifying problems: summary

Desiderata. Problem with algorithm that matches lower bound.
Ex. Sorting and convex hull have complexity $N \log N$.

Desiderata'. Prove that two problems $X$ and $Y$ have the same complexity.
  • First, show that problem $X$ linear-time reduces to $Y$.
  • Second, show that $Y$ linear-time reduces to $X$.
  • Conclude that $X$ and $Y$ have the same complexity.

even if we don't know what it is!
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 algorithm depending on size of operands.

GMP — Arithmetic without limitations
Matrix multiplication. Given two $N$-by-$N$ matrices, compute their product.

Brute force. $N^3$ flops.

\[
\begin{bmatrix}
0.1 & 0.2 & 0.8 & 0.1 \\
0.5 & 0.3 & 0.9 & 0.6 \\
0.1 & 0.0 & 0.7 & 0.4 \\
0.0 & 0.3 & 0.3 & 0.1 \\
\end{bmatrix}
\times
\begin{bmatrix}
0.4 & 0.3 & 0.1 & 0.1 \\
0.2 & 0.2 & 0.0 & 0.6 \\
0.0 & 0.0 & 0.4 & 0.5 \\
0.8 & 0.4 & 0.1 & 0.9 \\
\end{bmatrix}
= \begin{bmatrix}
0.16 & 0.11 & 0.34 & 0.62 \\
0.74 & 0.45 & 0.47 & 1.22 \\
0.36 & 0.19 & 0.33 & 0.72 \\
0.14 & 0.10 & 0.13 & 0.42 \\
\end{bmatrix}
\]

\[0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47\]
## Linear algebra reductions

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

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

<table>
<thead>
<tr>
<th>problem</th>
<th>linear algebra</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix multiplication</td>
<td>$A \times B$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>$A^{-1}$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>determinant</td>
<td>$</td>
<td>A</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>$Ax = b$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>$A = LU$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>least squares</td>
<td>$\min</td>
<td></td>
</tr>
</tbody>
</table>

Numerical linear algebra problems with the same complexity as matrix multiplication

**Q.** Is brute-force algorithm optimal?
### History of complexity of matrix 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^3 )</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>( N^{2.808} )</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>( N^{2.796} )</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>( N^{2.780} )</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>( N^{2.522} )</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>( N^{2.517} )</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>( N^{2.496} )</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>( N^{2.479} )</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>( N^{2.376} )</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>( N^{2.3737} )</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>( N^{2.3727} )</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>( N^{2 + \varepsilon} )</td>
</tr>
</tbody>
</table>

Number of floating-point operations to multiply two \( N \)-by-\( N \) matrices
6.5 Reductions

- Introduction
- Designing algorithms
- Establishing lower bounds
- Classifying problems
- Intractability
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?

**Frustrating news.** Very few successes.
A key problem: satisfiability

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

Ex.

\[ -x_1 \; or \; x_2 \; or \; x_3 = true \]
\[ x_1 \; or \; \neg x_2 \; or \; x_3 = true \]
\[ \neg x_1 \; or \; \neg x_2 \; or \; \neg x_3 = true \]
\[ \neg x_1 \; or \; \neg x_2 \; or \; x_4 = true \]
\[ x'_2 \; or \; x_3 \; or \; x_4 = true \]

\[ \begin{array}{c|cccc}
   & x_1 & x_2 & x_3 & x_4 \\
\hline
   T & T & F & T \\
\end{array} \]

**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).
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)

Mentality.
- 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 $Y$. 
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 $k$.

Applications. Scheduling, computer vision, clustering, ...
3-satisfiability reduces to independent set

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

**Pf.** Given an instance $\Phi$ of 3-SAT, create an instance $G$ of IND-SET:
- For each clause in $\Phi$, create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

$$\Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4)$$
3-satisfiability reduces to independent set

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

**Pf.** Given an instance \(\Phi\) of \(3-SAT\), create an instance \(G\) of \(IND-SET\):
- For each clause in \(\Phi\), create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

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

- \(\Phi\) 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 \( \Phi \) of 3-SAT, create an instance \( G \) of IND-SET:
- For each clause in \( \Phi \), create 3 vertices in a triangle.
- Add an edge between each literal and its negation.

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

- \( \Phi \) satisfiable \( \Rightarrow \) \( G \) has independent set of size \( k \).
- \( G \) has independent set of size \( k \) \( \Rightarrow \) \( \Phi \) 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-\textit{SAT} poly-time reduces to \textit{IND-SET}.

**Implication.** Assuming 3-\textit{SAT} is intractable, so is \textit{IND-SET}.

\[ \Phi = (x_1 \text{ or } x_2 \text{ or } x_3) \text{ and } (\neg x_1 \text{ or } \neg x_2 \text{ or } x_4) \text{ and } (\neg x_1 \text{ or } x_3 \text{ or } \neg x_4) \text{ and } (x_1 \text{ or } x_3 \text{ or } x_4) \]
**Integer linear programming**

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

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

**yes instance:** 
\[
\begin{array}{cccccc}
0 & 1 & 0 & 1 & 1
\end{array}
\]

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

**Remark.** Finding a real-valued solution is tractable (linear programming).
**Independent set reduces to integer linear programming**

**Proposition.** *IND-SET* poly-time reduces to *ILP*.

**Pf.** Given instance \( \{ G, k \} \) of *IND-SET*, create an instance of *ILP* as follows:

\[
\begin{align*}
\text{number of vertices selected} & \quad \text{is there an independent set of size 3?} \\
\text{at most one vertex selected from each edge} & \\
\text{binary variables} & \\
\text{is there a feasible solution?}
\end{align*}
\]

\[
x_1 + x_2 + x_3 + x_4 + x_5 = 3 \\
x_1 + x_2 \leq 1 \\
x_2 + x_3 \leq 1 \\
x_1 + x_3 \leq 1 \\
x_1 + x_4 \leq 1 \\
x_3 + x_5 \leq 1 \\
\text{all } x_i = \{ 0, 1 \}
\]

**Intuition.** \( x_i = 1 \) if and only if vertex \( v_i \) is in independent set.
3-satisfiability reduces to integer linear programming

**Proposition.** 3-	extit{SAT} poly-time reduces to \textit{IND-SET}.

**Proposition.** \textit{IND-SET} poly-time reduces to \textit{ILP}.

**Transitivity.** If $X$ poly-time reduces to $Y$ and $Y$ poly-time reduces to $Z$, then $X$ poly-time reduces to $Z$.

**Implication.** Assuming 3-	extit{SAT} is intractable, so is \textit{ILP}.

lower-bound mentality: if I could solve ILP efficiently, I could solve IND-SET efficiently; if I could solve IND-SET efficiently, I could solve 3-SAT efficiently
More poly-time reductions from 3-satisfiability

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.

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

\[ x_1 = \text{true}, \ x_2 = \text{true}, \ x_3 = \text{true}, \ x_4 = \text{true} \]

Ex 2. IND-SET.

\[ k = 3 \]

\{ v_2, \ v_4, \ v_5 \}
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.
Importance. What scientists and engineers aspire to compute feasibly.

Fundamental question.

Consensus opinion. No.
Cook-Levin theorem

An NP problem is **NP-COMPLETE** if all problems in NP poly-time to reduce to it.

**Cook-Levin theorem.** 3-\textit{SAT} is **NP-COMPLETE**.

**Corollary.** 3-\textit{SAT} is tractable if and only if \( P = \text{NP} \).

Two worlds.

![Diagram](image-url)
Implications of Cook-Levin theorem

All of these problems (and many, many more) poly-time reduce to 3-SAT.
Implications of Karp + Cook-Levin

All of these problems are NP-complete; they are manifestations of the same really hard problem.
**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>min, max, median, Burrows-Wheeler transform, ...</td>
</tr>
<tr>
<td>linearithmic</td>
<td>( N \log N )</td>
<td>sorting, element distinctness, convex hull, closest pair, ...</td>
</tr>
<tr>
<td>quadratic</td>
<td>( N^2 )</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>:</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.
Birds-eye view: revised

**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,</td>
</tr>
<tr>
<td>linearithmic</td>
<td>N log N</td>
<td>sorting, convex hull,</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,</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>NP-complete</td>
<td>probably not N^b</td>
<td>3-SAT, IND-SET, ILP, ...</td>
</tr>
</tbody>
</table>

**Good news.** Can put many problems into equivalence classes.
Complexity class. Set of problems sharing some computational property.

Bad news. Lots of complexity 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 path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.