6.5 Reductions

- 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, convex hull, closest pair, farthest pair, ...</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
**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.** [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. $N \log N + N.$
**Def.** Problem $X$ reduces to problem $Y$ if you can use an algorithm that solves $Y$ to help solve $X$.

![Diagram](image_url)

**Ex 2.** [3-collinear reduces to sorting]

To solve 3-collinear instance on $N$ points in the plane:

- For each point, sort other points by polar angle or slope.
  - check adjacent triples for collinearity

**Cost of solving 3-collinear.** $N^2 \log N + N^2$. 
› 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 $X$.

**Ex.**
- Element distinctness reduces to sorting.
- 3-collinear reduces to sorting.
- CPM reduces to topological sort. [shortest paths lecture]
- h-v line intersection reduces to 1d range searching. [geometric BST lecture]
- Baseball elimination reduces to maxflow. [assignment 7]
- Burrows-Wheeler transform reduces to suffix sort. [assignment 8]
- ...

**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).

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

**Pf.** Graham scan algorithm.

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

**Diagram:**
- Convex hull
- Sorting

**Table:**
- 1251432
- 2861534
- 3988818
- 4190745
- 13546464
- 89885444
- 43434213
- 34435312
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

Proof. 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 negative weights and cycles](image)

**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
- Sorting
- 2d closest pair
- 2d Euclidean MST
- Delaunay triangulation

**Combinatorial Optimization**
- Undirected shortest paths (nonnegative)
- Directed shortest paths (nonnegative)
- Bipartite matching
- Maximum flow
- Arbitrage
- Shortest paths (no neg cycles)
- Baseball elimination
- Linear programming (see ORF 307)
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.

**Bad news.** Very difficult to establish lower bounds from scratch.

**Good news.** Can 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:

$x_i < x_j$ or $(x_j - x_i) (x_k - x_i) - (x_i) (x_j - x_i) < 0$

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

**Pf.** [see next slide]  

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

- **Sorting instance:** $x_1, x_2, \ldots, x_N$.
- **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 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?

```
1251432
-2861534
3988818
-4190745
13546464
89885444
-43434213
```

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]

**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.
**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:

\[
0 = \begin{vmatrix}
  a & a^3 & 1 \\
  b & b^3 & 1 \\
  c & c^3 & 1 \\
\end{vmatrix}
\]

\[
= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3)
\]

\[
= (a - b)(b - c)(c - a)(a + b + c)
\]
More linear-time reductions and lower bounds

**sorting**

- element distinctness
  - (N log N lower bound)

**3-sum**

- 3-sum
  - (conjectured N^2 lower bound)

**2d convex hull**

**2d Euclidean MST**

**Delaunay triangulation**
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.

Q. How to convince yourself no sub-quadratic \textit{3-COLLINEAR} algorithm likely.
A2. [easy way] Linear-time reduction from \textit{3-SUM}.
designing algorithms
establishing lower bounds
classifying problems
intractability
Classifying problems: summary

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

**Ex.** Sorting, convex hull, and closest pair 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!
Primality testing

**PRIME.** Given an integer $x$ (represented in binary), is $x$ prime?

**COMPOSITE.** Given an integer $x$, does $x$ have a nontrivial factor?

**Proposition.** $PRIME$ linear-time reduces to $COMPOSITE$.

```java
public static boolean isPrime(BigInteger x) {
    if (isComposite(x)) return false;
    else                return true;
}
```

147573952589676412931

prime

147573952589676412927

composite
Primality testing

**PRIME.** Given an integer $x$ (represented in binary), is $x$ prime?

**COMPOSITE.** Given an integer $x$, does $x$ have a nontrivial factor?

**Proposition.** COMPOSITE linear-time reduces to PRIME.

```java
public static boolean isComposite(BigInteger x)
{
    if (isPrime(x)) return false;
    else            return true;
}
```

147573952589676412931

prime

147573952589676412927

composite
Caveat

**PRIME.** Given an integer $x$ (represented in binary), is $x$ prime?

**COMPOSITE.** Given an integer $x$, does $x$ have a nontrivial factor?

**Proposition.** **PRIME** linear-time reduces to **COMPOSITE**.

**Proposition.** **COMPOSITE** linear-time reduces to **PRIME**.

**Conclusion.** **PRIME** and **COMPOSITE** have the same complexity.

A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements `isComposite()` using `isPrime()`.
- Bob implements `isPrime()` using `isComposite()`.
- Infinite reduction loop!
- Who's fault?

best known deterministic algorithm is about $N^6$ for N-bit integer
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>$\left\lfloor \sqrt{a} \right\rfloor$</td>
<td>$M(N)$</td>
</tr>
</tbody>
</table>

integer arithmetic problems with the same complexity as integer multiplication

**Q.** Is brute-force algorithm optimal?
### Complexity of integer multiplication history

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</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>

**Remark.** GNU Multiple Precision Library uses one of five different algorithms depending on the size of operands.
Matrix multiplication. Given two $N$-by-$N$ matrices, compute their product.

Brute force. $N^3$ flops.

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

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

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

Brute force. $N^3$ flops.

<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?
## Complexity of matrix multiplication history

<table>
<thead>
<tr>
<th>year</th>
<th>algorithm</th>
<th>order of growth</th>
</tr>
</thead>
<tbody>
<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 + \epsilon}</td>
</tr>
</tbody>
</table>

*number of floating-point operations to multiply two N-by-N matrices*
› 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.
3-satisfiability

Literal. A boolean variable or its negation. \( x_i \) or \( \neg x_i \)

Clause. An or of 3 distinct literals. \( C_1 = (\neg x_1 \lor x_2 \lor x_3) \)

Conjunctive normal form. An and of clauses. \( \Phi = (C_1 \land C_2 \land C_3 \land C_4 \land C_5) \)

3-SAT. Given a CNF formula \( \Phi \) consisting of \( k \) clauses over \( n \) literals, does it have a satisfying truth assignment?

\[
\Phi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_2 \lor x_3 \lor x_4)
\]

\[
(\neg T \lor T \lor F) \land (T \lor \neg T \lor F) \land (\neg T \lor \neg T \lor \neg F) \land (\neg T \lor \neg T \lor T) \land (\neg T \lor F \lor T)
\]

yes instance

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

Applications. Circuit design, program correctness, ...
3-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*. 
An independent set is a set of vertices, no two of which are adjacent.

**IND-SET.** Given a 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.

$k = 4$

\[
\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)
\]
Proposition. \(3\text{-SAT}\) poly-time reduces to IND-SET.

**Pf.** Given an instance \(\Phi\) of \(3\text{-SAT}\), 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.

\[
\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)
\]

- \(G\) 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.** $3$-$SAT$ 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.

$k = 4$

\[
\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)
\]

- $G$ has independent set of size $k \Rightarrow \Phi$ satisfiable.
- $\Phi$ 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.** \(3\text{-SAT}\) poly-time reduces to \(IND\text{-SET}\).

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

\[k = 4\]

\[\Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4)\]
**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:**

\[
\begin{array}{ccccc}
x_1 & x_2 & x_3 & x_4 & x_5 \\ 0 & 1 & 0 & 1 & 1
\end{array}
\]

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

**Remark.** Finding a real-valued solution is tractable (linear programming).
**Proposition.** *IND-SET* poly-time reduces to *ILP*.

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

\[
x_1 + x_2 + x_3 + x_4 + x_5 = 3
\]

\[
x_1 + x_2 \leq 1
\]

\[
x_2 + x_3 \leq 1
\]

\[
x_1 + x_3 \leq 1
\]

\[
x_1 + x_4 \leq 1
\]

\[
x_3 + x_5 \leq 1
\]

\[
\text{all } x_i = \{ 0, 1 \}
\]

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

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

**Proposition.** \(\text{IND-SET}\) poly-time reduces to \(\text{ILP}\).

**Transitivity.** If \(X\) poly-time reduces to \(Y\) and \(Y\) poly-time reduces to \(Z\), then \(X\) poly-time reduces to \(Z\).

**Implication.** Assuming \(3\text{-SAT}\) is intractable, so is \(\text{ILP}\).

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

3-SAT

3-COLOR

IND-SET

VERTEX COVER

EXACT COVER

SUBSET-SUM

ILP

CLIQUE

HAM-CYCLE

TSP

HAM-PATH

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-$\text{SAT}$).

**A2.** [easy way] Reduction from 3-$\text{SAT}$. 

**Caveat.** Intricate reductions are common.
Search problems

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

Ex 1. 3-SAT.

\[ \Phi = (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_3 \lor x_4) \]

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

Ex 2. IND-SET.

\[ \{v_2, v_4, v_5\} \]
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’s theorem**

An \( NP \) problem is **NP-complete** if all problems in \( NP \) poly-time to reduce to it.

**Cook’s theorem.** \( 3\text{-SAT} \) is \( NP\)-complete.

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

---

**Two worlds.**

![Diagram showing two worlds: One with \( P \neq NP \) and \( P, NPC \) in \( NP \), and another with \( P = NP \).]
Implications of Cook's theorem

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

Stephen Cook
'82 Turing award
Implications of Karp + Cook

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, convex hull, closest pair, farthest pair, ...</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>
**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, convex hull, closest pair, farthest pair, ...</td>
</tr>
<tr>
<td>M(N)</td>
<td>?</td>
<td>integer multiplication, division, square root, ...</td>
</tr>
<tr>
<td>3-SUM complete</td>
<td>probably N²</td>
<td>3-SUM, 3-COLLINEAR, 3-CONCURRENT, ...</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>probably not Nᵇ</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 that share some computational property.

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

**Bad news.** Lots of complexity classes.
Summary

Reductions are important in theory to:
- Establish tractability.
- Establish intractability.
- Classify problems according to their computational requirements.

Reductions are important in practice to:
- Design algorithms.
- Design reusable software modules.
  - stacks, queues, priority queues, symbol tables, sets, graphs
  - sorting, regular expressions, Delaunay triangulation
  - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems