6.5 REDUCTIONS

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

Overview: introduction to advanced topics

Main topics. [final two lectures]
- 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!

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, closest pair, Euclidean MST, ...</td>
</tr>
<tr>
<td>quadratic</td>
<td>( N^2 )</td>
<td>?</td>
</tr>
<tr>
<td>\cdot</td>
<td>\cdot</td>
<td>\cdot</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

---

**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 finding the median. \( N \log N + 1 \).

**Ex 2.** [element distinctness reduces to sorting]
To solve element distinctness on \( N \) items:
- Sort \( N \) items.
- Check adjacent pairs for equality.

Cost of element distinctness. \( N \log N + N \).
Reduction

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

Reductions: quiz 1

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

I. MAX-FLOW reduces to MIN-CUT.
II. MIN-CUT reduces to MAX-FLOW.

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

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

6.5 Reductions

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.

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 $X$?
3-collinear

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

Brute force $N^3$. For all triples of points $(p, q, r)$, check if they are collinear.

3-collinear reduces to sorting

Sorting-based algorithm. For each point $p$,
- Compute the slope that each other point $q$ makes with $p$.
- Sort the $N-1$ points by slope.
- Collinear points are adjacent.

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.

Cost of solving undirected shortest paths. $E \log V + (E + V)$.

Some reductions in combinatorial optimization

baseball elimination mincut bipartite matching undirected shortest paths (nonnegative)

maxflow directed shortest paths (nonnegative) arbitrage shortest paths (in a DAG)

assignment problem directed shortest paths (no neg cycles)

linear programming
6.5 REDUCTIONS

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.

Bird's-eye view

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 $Y$.

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 $Y$.

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 $Y$.

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. 3-COLLINEAR reduces to SORTING.
D. BURROWS-WHEELER-TRANSFORM reduces to SUFFIX-SORTING.
E. I don’t know.
**ELEMENT-DISTINCTNESS** linear-time reduces to **2D-CLOSEST-PAIR**

**ELEMENT-DISTINCTNESS.** Given $N$ elements, are any two equal?

**2D-CLOSEST-PAIR.** Given $N$ points in the plane, find the closest pair.

Some linear-time reductions in computational geometry

- **element distinctness** (N log N lower bound)
- **sorting**
- **2d closest pair**
  - **2d convex hull**
  - **2d Euclidean MST**
  - **smallest enclosing circle**
    - Delaunay triangulation
      - **Voronoi diagram**
        - largest empty circle (N log N lower bound)

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

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

More geometric reductions and lower bounds

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:

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

$$= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3)$$

$$= (a - b)(b - c)(c - a)(a + b + c)$$
Complexity of 3-SUM

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

---

### 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 $O(n^{3/2}$, that there is a randomized 3SUM algorithm running in $O(n^2 \log \log n)$ time, and a deterministic algorithm running in $O(n^2 (\log \log n)^{3/2} \log^{3/2}$ time. These results refute the strongest version of the 3SUM conjecture, namely that the decision tree (and algorithms) complexity is $O(n^2)$.

---

6.5 REDUCTIONS

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

---

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 Euclidean-MST algorithm exists?
A2. [easy way] Linear-time reduction from element distinctness.

---

Classifying problems: summary

Desiderata. Problem with algorithm that matches lower bound.

Ex. Sorting and element distinctness have complexity $N \log N$.

Desiderata’. Prove that two problems $X$ and $Y$ have the same complexity.

First, show that problem $X$ linear-time reduces to $Y$.
- Second, show that $Y$ linear-time reduces to $X$.
- Conclude that $X$ has complexity $N^b$ iff $Y$ has complexity $N^b$ for $b \geq 1$.

---

- $X = \text{sorting}$
- $Y = \text{element distinctness}$
- $\text{integer multiplication}$
- $\text{integer division}$
Integer arithmetic reductions

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

Brute force. $N^2$ bit operations.

\[
\begin{array}{ccccccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\times & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

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>$N^{1.460}$, $N^{1.404}$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom–Cook</td>
<td>$N^{1.772}$</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>$N \log N 2^{3.076}$</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.

Numerical linear algebra reductions

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

Brute force. $N^3$ flops.

\[
\begin{array}{cccccc}
0.1 & 0.2 & 0.8 & 0.1 & 0.4 & 0.3 & 0.1 & 0.1 \\
0.5 & 0.3 & 0.9 & 0.6 & 0.2 & 0.2 & 0.0 & 0.6 \\
0.1 & 0.0 & 0.7 & 0.4 & 0.0 & 0.0 & 0.4 & 0.5 \\
0.0 & 0.3 & 0.3 & 0.1 & 0.8 & 0.4 & 0.1 & 0.9 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0.16 & 0.11 & 0.34 & 0.62 & 0.74 & 0.45 & 0.47 & 1.22 \\
0.36 & 0.19 & 0.33 & 0.72 & 0.14 & 0.12 & 0.13 & 0.42 \\
0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47 \\
\end{array}
\]
Numerical 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>$L A L^T$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>$Ax = b$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>$A = L U$</td>
<td>MM($N$)</td>
</tr>
<tr>
<td>least squares</td>
<td>$\min |Ax - b|_2$</td>
<td>MM($N$)</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>$N^{1.780}$</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>$N^{2.3757}$</td>
</tr>
<tr>
<td>2012</td>
<td>Williams</td>
<td>$N^{2.372873}$</td>
</tr>
<tr>
<td>2014</td>
<td>de Gall</td>
<td>$N^{2.372864}$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$N^{2 + \varepsilon}$</td>
</tr>
</tbody>
</table>

History of complexity of matrix multiplication (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 core problem: satisfiability

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

Ex.

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

*instance I*  **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)

**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.

Integer linear programming

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

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

*instance I*  **solution S**

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 & \lor x_2 \lor x_3 = \text{true} \\
x_1 & \lor \neg x_2 \lor x_3 = \text{true} \\
\neg x_1 & \lor \neg x_2 \lor \neg x_3 = \text{true} \\
\neg x_1 & \lor \neg x_2 \lor x_3 \lor x_4 = \text{true} \\
\neg x_2 & \lor \neg x_3 \lor x_4 = \text{true} \\
\neg x_2 & \lor \neg x_3 \lor \neg 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

More 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.

Implications of poly-time reductions from 3-satisfiability

Reductions: quiz 3

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

A. If X can be solved in poly-time, then so can Y.

B. If X cannot be solved in cubic time, Y cannot be solved in poly-time.

C. If Y can be solved in cubic time, then X can be solved in poly-time.

D. If Y cannot be solved in poly-time, then neither can X.

E. I don’t know.
Search problems

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

**Ex 1.** 3-Sat.

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

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

instance I

solution S

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

\[
\begin{align*}
147573952589676412927 \\
193707721
\end{align*}
\]

Consensus opinion. No.

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.

Cook-Levin theorem

A problem is **NP-Complete** if

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

**Cook-Levin theorem.** 3-Sat is **NP-Complete**.

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

Two worlds.

Implications of Cook-Levin theorem

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

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

Reductions: quiz 4

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

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

A. I only.
B. II only.
C. I and II only.
D. I, II, and III.
E. I don’t know.

Birds-eye view: review

Desiderata. Classify problems according to computational requirements.

<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>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.

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, 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>$NP$-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 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.