6.5 Reductions

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

Main topics. [next 3 lectures]
- Reduction: design algorithms, establish lower bounds, classify problems.
- Linear programming: the ultimate practical problem-solving model.
- Intractability: problems beyond our reach.

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
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></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.
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 $X = \text{total cost of solving } Y + \text{cost of reduction}$.
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. $N \log N + 1$. 

instance $I$ (of $X$) \[\rightarrow\] Algorithm for $Y$ \[\rightarrow\] solution to $I$
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
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
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.**
- 3-collinear reduces to sorting.  [assignment]
- Finding the median reduces to sorting.
- Element distinctness reduces to sorting.
- CPM reduces to topological sort.  [shortest paths lecture]
- Arbitrage reduces to shortest paths.  [shortest paths lecture]
- Burrows-Wheeler transform reduces to suffix sort.  [assignment]
- ...

**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](image)

![Sorting](image)

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

**Pf.** Graham scan algorithm (see next slide).

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

Cost of sorting 

Cost of reduction
Graham scan algorithm

Graham scan.

- Choose point \( p \) with smallest (or largest) y-coordinate.
- **Sort** points by polar angle with \( p \) to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

Pf. Replace each undirected edge by two directed edges.
Shortest paths on edge-weighted graphs and digraphs

**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).

\[
\begin{array}{c}
\text{s} \quad 7 \quad \text{--} \quad -4 \quad \text{t} \\
\text{s} \quad 7 \quad \text{--} \quad -4 \quad \text{t}
\end{array}
\]

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

- finding the median
- element distinctness
- sorting
- convex hull

Note: See the text for references.

- parallel scheduling (precedence-constrained)
- shortest paths in digraphs
- shortest paths in undirected graphs (no negative weights)
- convex hull

- SPT scheduling
- sorting
- arbitrage

- convex hull
- shortest paths in digraphs
- linear programming (stay tuned)
- network reliability
- product distribution
- maxflow
- bipartite matching

- product distribution
- maxflow
- bipartite matching
- network reliability
- convex hull
- shortest paths in digraphs
- linear programming (stay tuned)
- parallel scheduling (precedence-constrained)
- shortest paths in undirected graphs (no negative weights)
- convex hull
- shortest paths in digraphs
- linear programming (stay tuned)
- parallel scheduling (precedence-constrained)
- shortest paths in undirected graphs (no negative weights)
- convex hull
- shortest paths in digraphs
- linear programming (stay tuned)
6.5 Reductions

- Introduction
- Designing algorithms
- Establishing lower bounds
- Classifying problems
6.5 REductions

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

---

**Diagram:**

- Root node $a < b$.
- Branch $b < c$: yes leads to $\{a, b, c\}$, no leads to $\{a < c\}$.
- Branch $a < c$: yes leads to $\{a, c, b\}$, no leads to $\{b, a, c\}$.
- Branch $b < c$: yes leads to $\{b, c, a\}$, no leads to $\{c, b, a\}$.

*Argument must apply to all conceivable algorithms.*
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:
\[
x_i < x_j \text{ or } (x_j - x_i)(x_k - x_i) - (x_j)(x_j - x_i) < 0
\]

**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: \((x_1, x_1^2), (x_2, x_2^2), \ldots, (x_N, x_N^2)\).

\[
f(x) = x^2
\]

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 mentality: if I can solve convex hull efficiently, I can sort efficiently
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
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
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!
Caveat

**SORT.** 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.** SORT linear-time reduces to CONVEX HULL.

**Proposition.** CONVEX HULL linear-time reduces to SORT.

**Conclusion.** SORT and CONVEX HULL have the same complexity.

A possible real-world scenario.

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

well, maybe not so realistic
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>(\lceil \sqrt{a} \rceil)</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-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>

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.

used in Maple, Mathematica, gcc, cryptography, ...

\[\text{GMP}\]

«Arithmetic without limitations»
Linear algebra reductions

Matrix multiplication. Given two \( N \text{-by-} N \) 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


<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>
**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>:</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, determinant, ...</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>NP-complete</td>
<td>probably not N^b</td>
<td>SAT, IND-SET, ILP, ...</td>
</tr>
</tbody>
</table>

---

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

---

STAY TUNED!
Complexity zoo

Complexity class. Set of problems sharing some computational property.

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

Bad news. Lots of complexity classes.
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, Delaunay triangulation
  – MST, shortest path, maxflow, linear programming
• Determine difficulty of your problem and choose the right tool.
6.5 Reductions

- introduction
- designing algorithms
- establishing lower bounds
- classifying problems
6.5 Reductions

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