Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

**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 \( X \).

Cost of solving \( X \) = total cost of solving \( Y \) + 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. [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. $N \log N + N$.

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. $N^2 \log N + N^2$.

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.
- 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).

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

**Pf.** Graham scan algorithm.

Cost of convex hull. \( N \log N + N \).

---

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.

---

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.
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).

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

Some reductions involving familiar problems

<table>
<thead>
<tr>
<th>Computational geometry</th>
<th>Combinatorial optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d farthest pair</td>
<td>undirected shortest paths (nonnegative)</td>
</tr>
<tr>
<td>median</td>
<td>directed shortest paths (nonnegative)</td>
</tr>
<tr>
<td>element distinctness</td>
<td>arbitrage</td>
</tr>
<tr>
<td>sorting</td>
<td>maximum flow</td>
</tr>
<tr>
<td>2d closest pair</td>
<td>shortest paths (no neg cycles)</td>
</tr>
<tr>
<td>2d Euclidean MST</td>
<td>baseball elimination</td>
</tr>
<tr>
<td>Delaunay triangulation</td>
<td>linear programming</td>
</tr>
</tbody>
</table>

R E D U C T I O N S

- Designing algorithms
- Establishing lower bounds
- Classifying problems
### 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.

**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 $Y$.

### More linear-time reductions and lower bounds

- **sorting**: $N \log N$ lower bound
- **element distinctness**: $N \log N$ lower bound
- **3-sum**: conjectured $N^2$ lower bound
- **2d convex hull**
- **2d Euclidean MST**
- **3-concurrent**
- **min area triangle**
- **Delaunay triangulation**

---

**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.** Spread $\Omega(N \log N)$ lower bound to $Y$ by reducing sorting to $Y$. Assuming cost of reduction is not too high, argument must apply to all conceivable algorithms.

---

**Element distinctness linear-time reduces to closest pair**

**Closest pair.** Given $N$ points in the plane, find the closest pair.

**Element distinctness.** Given $N$ elements, are any two equal?

**Proposition.** Element distinctness linear-time reduces to closest pair.

**Pf.**
- Element distinctness instance: $x_1, x_2, \ldots, x_N$.
- Closest pair instance: $(x_1, x_1), (x_2, x_2), \ldots, (x_N, x_N)$.
- Two elements are distinct if and only if closest pair $= 0$.

**Element distinctness lower bound.** In quadratic decision tree model, any algorithm that solves element distinctness takes $\Omega(N \log N)$ steps.

**Implication.** In quadratic decision tree model, any algorithm for closest pair takes $\Omega(N \log N)$ steps.
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.

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.

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?
**Integer arithmetic reductions**

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

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

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

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

**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>$N^{1.447}$, $N^{1.494}$</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>$N \log N \log^2 N$</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.

**Integer arithmetic reductions**

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

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

**Problem** | **Arithmetic** | **Order of growth**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>$\sqrt{a}$</td>
<td>$M(N)$</td>
</tr>
</tbody>
</table>

integer arithmetic problems with the same complexity as integer multiplication

**Q.** Is brute-force algorithm optimal?

**Linear algebra reductions**

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

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

\[
\begin{array}{cccc}
0.1 & 0.2 & 0.8 & 0.1 \\
0.5 & 0.3 & 0.9 & 0.6 \\
0.1 & 0 & 0.7 & 0.4 \\
0 & 0.3 & 0.3 & 0.1 \\
\end{array}
\times
\begin{array}{cccc}
0.4 & 0.3 & 0.1 & 0.1 \\
0.2 & 0.2 & 0 & 0.6 \\
0 & 0 & 0.4 & 0.5 \\
0.8 & 0.4 & 0.1 & 0.9 \\
\end{array}
= \begin{array}{cccc}
0.16 & 0.11 & 0.34 & 0.62 \\
0.74 & 0.45 & 0.47 & 1.22 \\
0.36 & 0.19 & 0.33 & 0.72 \\
0.14 & 0.1 & 0.13 & 0.42 \\
\end{array}
\]

\[
\begin{array}{cccc}
0.5 & 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47
\end{array}
\]
Linear algebra reductions

Matrix multiplication. Given two \( N \times N \) 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>( \text{A} \times \text{B} )</td>
<td>( \text{MM}(N) )</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>( \text{A}^{-1} )</td>
<td>( \text{MM}(N) )</td>
</tr>
<tr>
<td>determinant</td>
<td>(</td>
<td>\text{A}</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>( \text{A}x = b )</td>
<td>( \text{MM}(N) )</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>( \text{A} = \text{L} \text{U} )</td>
<td>( \text{MM}(N) )</td>
</tr>
<tr>
<td>least squares</td>
<td>( \min |\text{Ax} - \text{b}|_2 )</td>
<td>( \text{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^{2.700} )</td>
</tr>
<tr>
<td>1981</td>
<td>Schönage</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>Ströher</td>
<td>( N^{2.3707} )</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: 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, closest pair, farthest pair, ...</td>
</tr>
<tr>
<td>( \text{M}(N) )</td>
<td>!</td>
<td>integer multiplication, division, square root, ...</td>
</tr>
<tr>
<td>( \text{MM}(N) )</td>
<td>!</td>
<td>matrix multiplication, ( \text{Ax} = b ), least square, determinant, ...</td>
</tr>
<tr>
<td>( \cdot )</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>( \text{NP-complete} )</td>
<td>!</td>
<td>probably not ( N^p )</td>
</tr>
</tbody>
</table>

\( 3\text{-SAT}, \text{IND-SET}, \text{ILP}, ... \)

Good news. Can put many problems into equivalence 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