Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.
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>$N$</td>
<td>?</td>
</tr>
<tr>
<td>exponential</td>
<td>$c$</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
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
**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 $X$.

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

- 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.**
- 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).

![Diagram of convex hull and sorting]

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

Proposition. Convex hull reduces to sorting.

Pf. 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.

![Diagram of Graham scan algorithm](image)
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.
**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. $E \log V + E$. 
**Caveat.** Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles).

![Graph](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 closest pair
- 2d farthest pair
- Sorting
- Convex hull
- Median
- Element distinctness

Combinatorial optimization
- Undirected shortest paths (nonnegative)
- Directed shortest paths (nonnegative)
- Arbitrage
- Maximum flow
- Baseball elimination
- Linear programming
Reductions

- 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.
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 $Y$. 
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 $= 0$.

allows quadratic tests of the form:

\[ x_i < x_j \text{ or } (x_i - x_k)^2 - (x_j - x_k)^2 < 0 \]

**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.
More linear-time reductions and lower bounds

- **Sorting**
  - Element distinctness (N log N lower bound)
  - 2d convex hull
  - 2d Euclidean MST
  - Delaunay triangulation

- **3-sum**
  - 3-sum (conjectured N^2 lower bound)
  - 3-collinear
  - 3-concurrent
  - Dihedral rotation
  - Min area triangle

- **2d closest pair**
  - 2d Euclidean MST
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.
REDUCTIONS

- Designing algorithms
- Establishing lower bounds
- Classifying problems
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!
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.** \textit{SORT} linear-time reduces to \textit{CONVEX HULL}.

**Proposition.** \textit{CONVEX HULL} linear-time reduces to \textit{SORT}.

**Conclusion.** \textit{SORT} and \textit{CONVEX HULL} have the same complexity.

A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements \texttt{sort()} using \texttt{convexHull()}.
- Bob implements \texttt{convexHull()} using \texttt{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>( \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</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba-Ofman</td>
<td>N</td>
</tr>
<tr>
<td>1963</td>
<td>Toom-3, Toom-4</td>
<td>N</td>
</tr>
<tr>
<td>1966</td>
<td>Toom-Cook</td>
<td>N</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</td>
</tr>
</tbody>
</table>

number of bit operations to multiply two N-bit integers used in Maple, Mathematica, gcc, cryptography, ...
### Linear algebra reductions

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

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

<table>
<thead>
<tr>
<th>row i</th>
<th>0.1</th>
<th>0.2</th>
<th>0.8</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
<td>0.3</td>
<td>0.9</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
<td>0</td>
<td>0.7</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$\times$

<table>
<thead>
<tr>
<th>column j</th>
<th>0.4</th>
<th>0.3</th>
<th>0.1</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.2</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>0.4</td>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

$= $

<table>
<thead>
<tr>
<th>j</th>
<th>0.16</th>
<th>0.11</th>
<th>0.34</th>
<th>0.62</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.74</td>
<td>0.45</td>
<td>0.47</td>
<td>1.22</td>
</tr>
<tr>
<td>i</td>
<td>0.36</td>
<td>0.19</td>
<td>0.33</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>0.14</td>
<td>0.1</td>
<td>0.13</td>
<td>0.42</td>
</tr>
</tbody>
</table>

$0.5 \cdot 0.1 + 0.3 \cdot 0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47$
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>$\text{MM}(N)$</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>$A$</td>
<td>$\text{MM}(N)$</td>
</tr>
<tr>
<td>determinant</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>system of linear equations</td>
<td>$Ax = b$</td>
<td>$\text{MM}(N)$</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>$A = L$</td>
<td>$\text{MM}(N)$</td>
</tr>
<tr>
<td>least squares</td>
<td>$\min |Ax - b|$</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</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>N</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>N</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>N</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>N</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>N</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>N</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>N</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>N</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>N</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>N</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>N</td>
</tr>
</tbody>
</table>

Number of floating-point operations to multiply two $N$-by-$N$ matrices
**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>$M(N)$</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>
</tbody>
</table>

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