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' = \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.** [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. \(N \log N + N\).

**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 \(X\)?
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.

![Diagram of shortest paths](image)

**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).

![Diagram of shortest paths with negative weights](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
- Median
- Element distinctness
- 2d closest pair
- 2d Euclidean MST
- Delaunay triangulation

**Combinatorial optimization**
- Convex hull
- Sorting
- Bipartite matching
- Maximum flow
- Baseball elimination
- Linear programming

**Undirected shortest paths** (nonnegative)
- Undirected shortest paths
- Convex hull
- Sorting
- Bipartite matching
- Maximum flow
- Baseball elimination
- Linear programming

**Directed shortest paths** (nonnegative)
- Directed shortest paths
- Bipartite matching
- Maximum flow
- Baseball elimination
- Linear programming

**Arbitrage**
- Arbitrage
- Shortest paths (no neg cycles)

**Reductions**
- Designing algorithms
- Establishing lower bounds
- Classifying problems
**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

---

**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.

---

**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**

**element distinctness**

\((N \log N \text{ lower bound})\)

**3-sum**

\((\text{conjectured } N^2 \text{ lower bound})\)

**2d convex hull**

**2d Euclidean MST**

**3-concurrent**

**min area triangle**

**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.

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.

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

**Linear algebra reductions**

**Matrix multiplication.** Given two \( N \)-by-\( N \) matrices, compute their product.

**Brute force.** \( N^3 \) flops.

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

*Remark.* GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.
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>$\text{MM}(N)$</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>$A\mathbf{x} = \mathbf{b}$</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>$\min |A\mathbf{x} - \mathbf{b}|$</td>
<td>$\text{MM}(N)$</td>
</tr>
</tbody>
</table>

Q. Is brute-force algorithm optimal?

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, $A\mathbf{x} = \mathbf{b}$, least squares, determinants, ...</td>
</tr>
<tr>
<td>$\mathbb{I}$</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>NP-complete</td>
<td>probably not $N$</td>
<td>3-SAT, IND-SET, ILP, ...</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