Reductions

- from individual problems to problem-solving models
- from linear / quadratic to polynomial / exponential scale
- from details of implementation to conceptual framework

* classify problems according to computational requirements
  - complexity
  - order of growth

$\boxed{\text{Def}}$: 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

Example 1: Finding the median reduces to sorting

=> cost of solving finding the median

$\underbrace{N \log N}_{\text{cost of sorting}} + (1) \longrightarrow$ cost of reduction

Example 2: Element distinctness reduces to sorting

=> cost = $N \log N + \underbrace{N}_{\text{reduction}}$

Designing Algorithms

=> given algorithm for Y, can also solve X

Example 3: Convex hull reduces to sorting

=> Graham scan algorithm =>

cost is $N \log N + N$

Example 4: Shortest path in undirected graph reduces to shortest path in directed graph

=> replace each undirected edge by two directed edges

cost => $\underbrace{E \log V}_{\text{shortest path}} + E)$ reduction

* reduction is invalid for edge-weighted graphs with negative weights

Establishing Lower Bounds

$\boxed{\text{Goal}}$: Prove that a problem requires a certain number of steps

* Spread lower bound to Y by reducing $\underbrace{\text{sorting}}_{\text{algo}}$ to Y

Linear time reductions

$\boxed{\text{Def}}$ 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

Classifying Problems

- prove problems X and Y have the same complexity
- X linear-time reduces to Y
- Y linear-time reduces to X
- X and Y have the same complexity

Example I: Integer multiplication

* given two N-bit integers => compute their product

Example II: Matrix Multiplication

* Complexity class = set of problems sharing some computational property