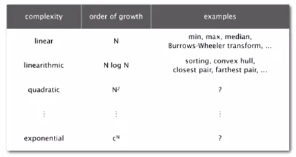
Reductions

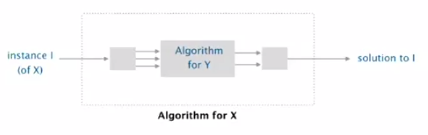
Desiderada: Classify problems according to computational requirements



Frustrating news: large number of problems we want to solve for which we don’t know the difficulty

Desiderada… suppose we could (could not) solve problem X efficiently. What else could (could not) we solve efficiently?

Reduction: Problem X reduces to problem Y is you can use an algorithm that solves problem Y to help solve X



Cost of solving X = total cost of solving Y (perhaps many calls to Y on problems of different sizes) + cost of reduction (preprocessing and postprocessing)

Example 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) and the cost of reduction (+ 1)… N log N + 1

Example 2 [element distinctness reduces to sorting]

To solve element distinctness on N items

* Sort N items
* Check adjacent pairs for equality

Cost of solving is N log N (sort) + N (reduce)

Designing Algorithms

Given an algorithm for Y, can also solve X

* 3-collinear points reduces to sorting
* Finding the median reduces to sorting
* Element distinctness reduces to sorting
* CPM reduces to topological sort
* Arbitrage reduces to shortest paths
* 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 on the plane, identify the extreme points of the convex hull (in counterclockwise order)

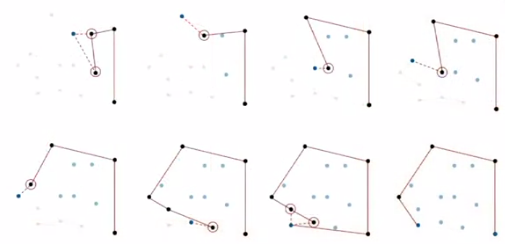
Proposition: convex hull reduces to sorting  
Proof: Graham scan algorithm



Cost: N log N (sort) + N (reduce)

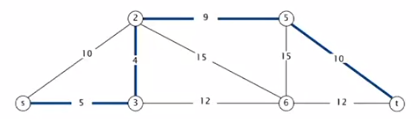
Graham scan

* 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

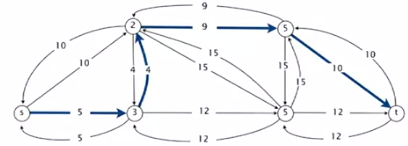


Shortest paths on edge-weighted graphs and digraphs

Proposition: Undirected shortest paths (with nonnegative weights) reduces to directed shortest path

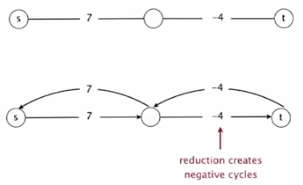


Proof: Replace each undirected edge by two directed edges



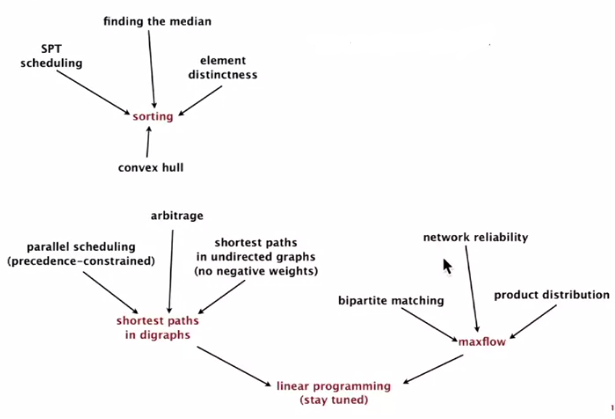
Cost of undirected shortest paths: E log V (cost of shortest paths) + E (reduction)

Caveat: Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles)



Note: can still solve shortest-paths problem in undirected graphs (if no negative cycles), but need more sophisticated techniques… reduces to weighted non-bipartite matching

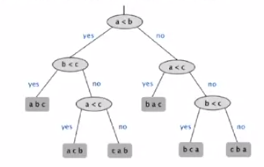
Reduction overview



Establishing lower bounds

Goal: Prove that a problem requires a certain number of computational steps

Example: In decision tree model, any compare-based sorting algorithm requires N log N compares in the worst case.



* Bad news: Very difficult to establish lower bounds from scratch. Argument must apply to all conceivable algorithms.
* Good news: Spread lower bound N log N to Y by reducing sorting to Y (assuming cost of reduction is not too high)

Linear-time reductions

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

Example: almost all of the reductions we’ve seen thus far.

Establish lower bound:

* If X takes Ω(N log N) steps, then so does Y
* If X takes Ω(N2) steps, then so does Y

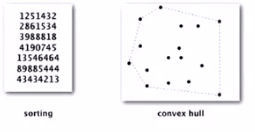
Mentality

* 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 Ω(N log N) steps

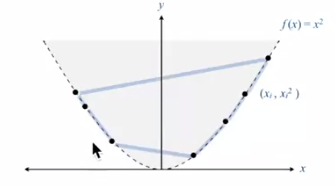
Proposition:   
sorting linear-time reduces to convex hull (lower-bound mentality: if I can solve convex hull efficiently, I can sort efficiently)



Implication: Any ccw-based convex hull algorithm requires Ω(N log N (linear or quadratic tests)) ops

Proposition: Sorting linear-time reduces to convex hull

* Sorting instance: x1, x2, … , xN
* Convex hull instance: (x1, x12), (x2, x22), … , (xN, xN2)



Proof:

* Region { x : x2 >= x } is convex -> all points are on hull
* Starting at point with most negative x, counterclockwise order of hull points yields integers in ascending order

Establishing lower bounds summary

Very important tool…

Q: How do you convince yourself there is no linear-time convex-hull algorithm in existence?

A (hard way): Long futile search for a linear-time algorithm  
A (easy way): Linear-time reduction from sorting

Classifying problems

Desiderata: prove that two problems X and Y have same complexity

* First, show that problem X linear-time reduces to Y
* Second, shoe 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)

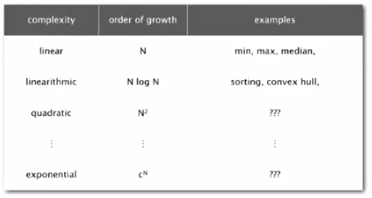
SORT: given N distinct integers, rearrange them in ascending order  
CONVEX HULL: Given N points in the plane, identify the extreme points of the convex hull (in ccw order)

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

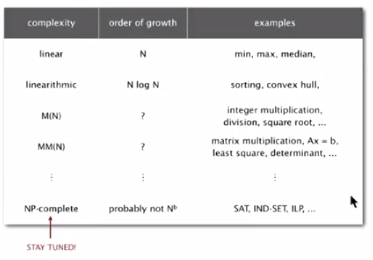
A possible real-world scenario

* System designer specs the APIs for the project
* Alice implements sort() using convexHull() 🡨 technically not realistic
* Bob implements (convexHull() using sort()
* Infinite reduction loop
* Who’s fault?

Classification of problems according to computational requirements



Classification as above, yet with more context now understood



Complexity zoo: set of problems sharing some computational property

Many complexity classes:



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, STs, sets, graphs
  + Sorting, regex, Delaunay triangulation
  + MST, shortest path, maxflow, linear programming
* Determine difficulty of your problem and choose right tool
  + Use exact algorithm for tractable problems
  + Use heuristics for intractable problems