CS5112: Algorithms and Data Structures for Applications

Reductions, lower bounds, approximation algorithms

Ramin Zabih

Some content from: Wikipedia, Abhiram Ranade (IIT), Jonathan Backer (UBC)

Administrivia

- Lectures ~~will be recorded "Real Soon Now"~~ are recorded!
- HW2 will be out Real Soon Now™
- Prelim (midterm) date is Wednesday October 23
  - In class, closed book exam
  - Review session in class on Monday October 21
    - Bring your questions!

Lecture Outline

- Roles in computer science
- Lower bounds are hard
- Sorting lower bound
- Consequences
- A 2-approximation algorithm for a TSP variant

What do we do with reductions?

- Reductions turn one problem into another one
- Light side: turn a seemingly hard one into an easy one
  - Many examples!
- Dark side: show that your seemingly easy problem would also solve problems known to be hard
  - This is surprisingly useful
- Beyond ‘uh oh, my problem is NP-hard’

Beyond upper bounds

- In CS we compute worst case upper bounds
  - Big O and all that
- Generally pretty useful, consistent with practice
- We know how to figure out big O complexity
  - Though specific cases can be difficult to get a tight bound
- Other options are harder
  - Average case requires a realistic input distribution
  - Often gives results that are inconsistent with practice

Lower bounds

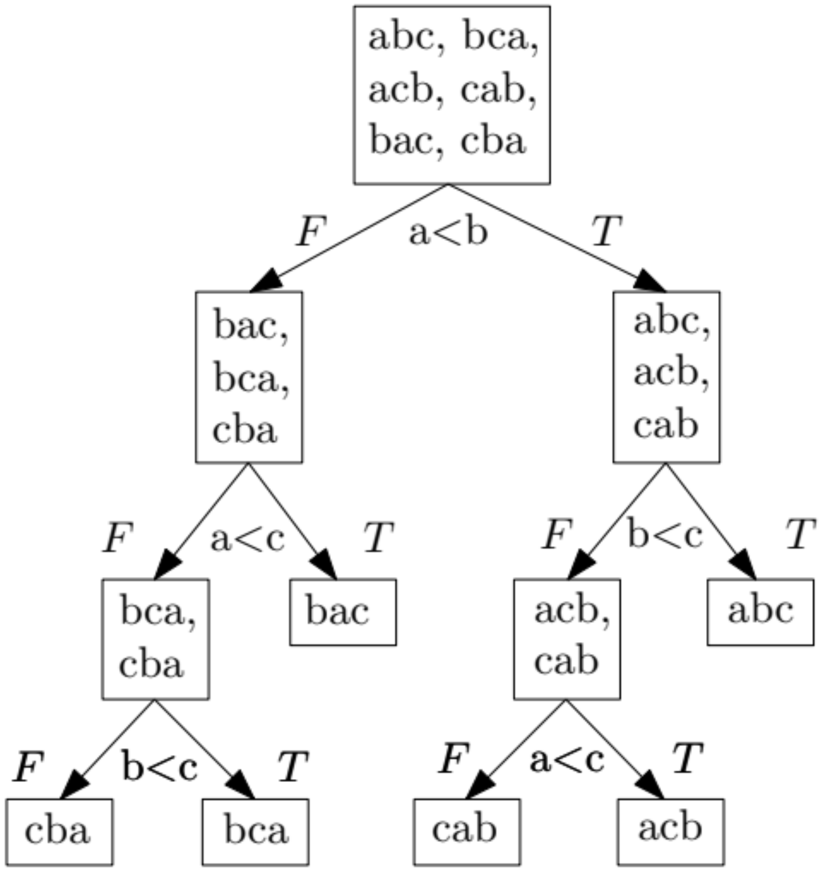
- Lower bounds are **much** harder
- Need to show that NO ALGORITHM can do better
  - Important subtlety: in the worst case
    - Best case can be really fast
  - There is almost always a trivial $O(n)$ lower bound
    - Why? Can you think of an exception?
    - What about the best case?
- Requires a model of computation
  - Example: comparison-based sorting

Comparison based sorting

- Most sorting algorithms use comparisons
  - What are the obvious exceptions?
- There are $n!$ possible outputs of the algorithm
  - For simplicity assume no ties
- We can view this as a comparison tree
  - At the top we compare items #3 and #75, e.g.
    - Then if #3 is larger we compare #2 and #17, etc.
  - There are $n!$ leaves in the tree

Comparison tree for insertion sort

Algorithm: Insertion sort.
Instance ($n = 3$): the numbers a, b, c.



Slide from:
[Jonathan Backer]

Lower bound for sorting

- Tree with $n!$ leaves has depth at least $O(n \log n)$
- So we need this many comparisons to sort in the worst case
- This does NOT imply that a correct sorting algorithm must do this many comparisons on a particular input
  - Example: any algorithm, modified to check if input already sorted
- Comparison-based sorting must be $O(n \log n)$ or slower
  - Recall that big O is worst case
  - "Worst case behavior must be at least this bad"

Convexity and convex hulls

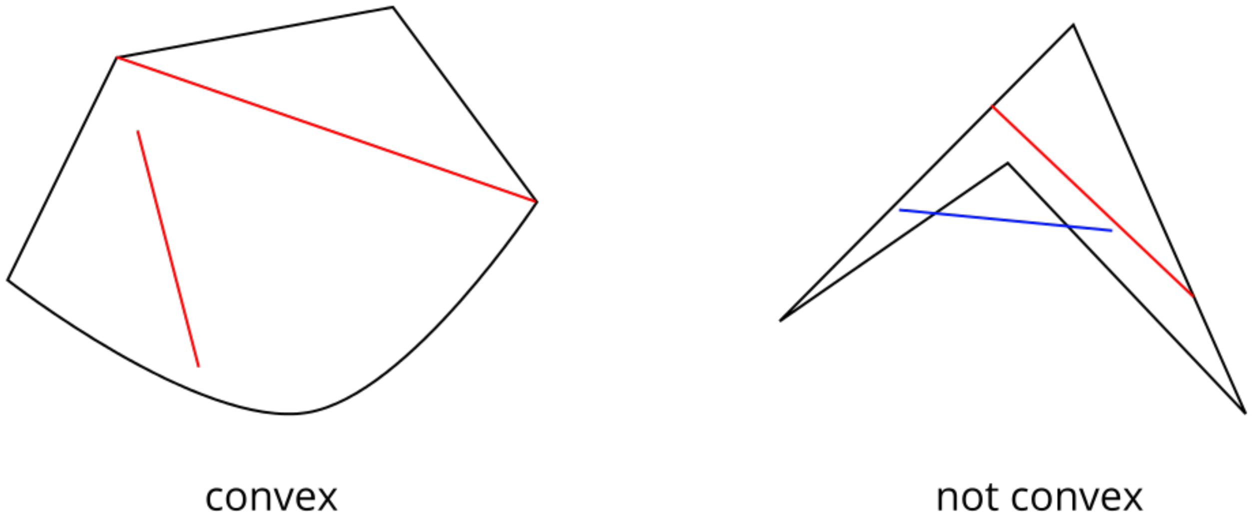


Figure source

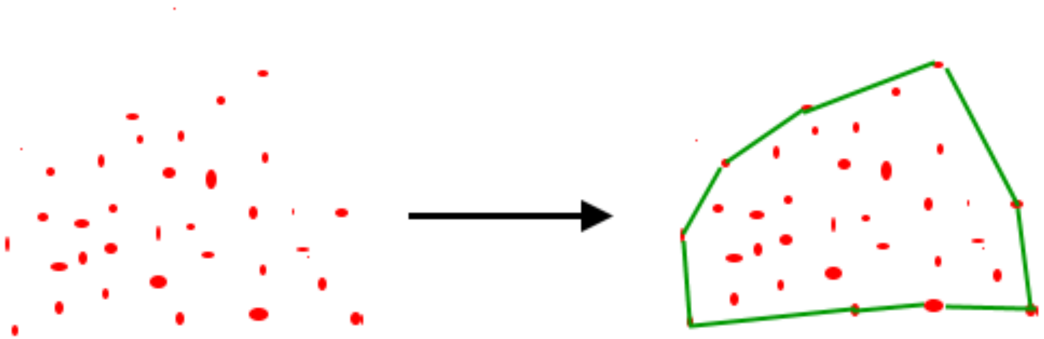


Figure source

Convex hull problem

- Definition: a convex set contains the line between any two points in the set
  - This captures the intuition about convexity
- Definition: the convex hull of a set of points is the smallest convex set containing those points
  - More precisely, the algorithm contains the points on the hull in a specific order (assume counter-clockwise)
- How fast can we compute the convex hull?

Sorting via convex hull

- Suppose you want to sort the positive numbers $\{x_i\}$
- Consider the convex hull of the points $\{(x_i, x_i^2)\}$
  – The points fall along a parabola
- By computing the convex hull we sort!
  – So we have an $O(n \log n)$ lower bound
- Why is it sufficient to consider positive numbers?
- What fact have we skipped over?

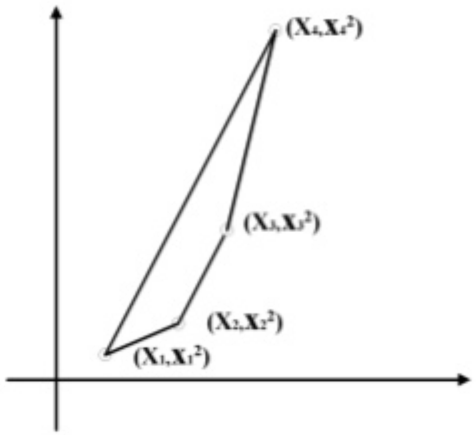


Figure source

CORNELL TECH

Euclidean MST

- Consider the MST problem in 2D
  - Edges exist between any pair of points (complete graph)
  - Weight = Euclidean distance
  - We can use this to sort!
- Suppose you want to sort the numbers $\{x_i\}$
- Place these on the x-axis, i.e. the points will be $\{(x_i, 0)\}$
- If the Euclidean MST contains the edge $(x_i, 0) - (x_j, 0)$ then x_i and x_j are consecutive in sorted order

Approximation algorithms

- Very advanced topic (many a PhD thesis)
- Often you have an NP-hard problem of computing the best solution under some objective function
  - Such as the cost of a traveling salesman tour, cost = OPT
- We can't find the best one fast
- Sometimes we can find one nearly as good!
  - Where we can bound how much more expensive it is than the best
- Ideally cost $\leq OPT + \epsilon$

Approximation algorithm for metric TSP

- Metric TSP: distances obey triangle inequality
  - I.e., the shortest way to get from v_1 to v_2 is to go directly, and not via some other vertex w
  - Still NP-hard to find the optimal tour, call the cost OPT
  - But we can get provable close using MST!
- Deleting an edge from a tour gives us a spanning tree
  - And does not increase the cost
- If the MST has cost C_{MST} then $C_{MST} \leq OPT$

The 2-approximation algorithm

- Find an MST
- Double each edge (so we have a multigraph)
- Degree of all vertices is even, so there's an Euler tour
- Find it in linear time, cost is $2 \cdot C_{MST}$
- Convert to a TSP solution by skipping previously visited vertices
  - This is where we need the metric assumption
- This gives us a TSP solution with cost $2 \cdot C_{MST} \leq 2 \cdot OPT$