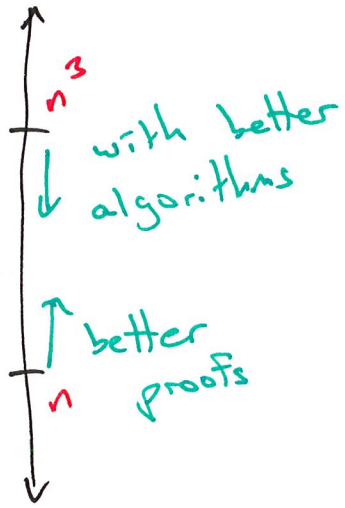
Problem Complexity

Def: The inherent worst-case complexity of a problem.

- For many problem we don't the problem complexity, but we can put bounds on it

  - Each better algorithm brings the upper bound downwards.

  - Each better proof brings the lower bound up



Problem Lower Bounds

Two main techniques

1. Information-theoretic Lower bound
2. Lower bound by reduction

Trivial Lower Bound

Finding the minimum of a set of n numbers that are not sorted or structured in any way.

Theorem:

This problem takes $\Omega(n)$-time

Proof

- A hypothetical algorithm that runs in $o(n)$ time can't examine all numbers in the input.
- If we run this algorithm on some input we can identify an element that was never examined.
- changing that element to be the smallest number. would cause the algorithm to output an incorrect result.

Sorting Lower Bound

Theorem: Every comparison-based sorting algorithm has a worst-case complexity of $\Omega(n \log n)$

Proof:

- Assume elements are the distinct numbers 1 through n.
- Each input is a permutation of $\{1 \ldots n\}$
  $n!$ possible inputs.
- we want to find a value k such that for any input it takes no more than k comparisons to correctly sort the input.

- Each comparison results in $>$ or $<$ which gives us 2^k possible computation paths.
- If $2^k < n!$ there are at least 2 inputs that share a computation path.
  - this would mean that for one of those inputs the algorithm is incorrect.
- So by correctness

$$2^k \geq n!$$

- So $k \geq \lg(n!)$

$$\in \Omega(n \log n)$$

$$\lg(n!) \in \Omega(n \log n)$$

$$\lg(n!) = \lg(n) + \lg(n-1) + \lg(n-2) + \ldots + \lg(2) + \lg(1)$$

$$\geq \lg(n) + \lg(n-1) + \ldots + \lg\left(\frac{n}{2}\right)$$

$$\geq \lg\left(\frac{n}{2}\right) + \lg\left(\frac{n}{2}\right) + \ldots + \lg\left(\frac{n}{2}\right)$$

$$\geq \frac{n}{2} \lg\left(\frac{n}{2}\right)$$

Element Uniqueness

Given a collection of n real numbers, determine whether they are all unique.

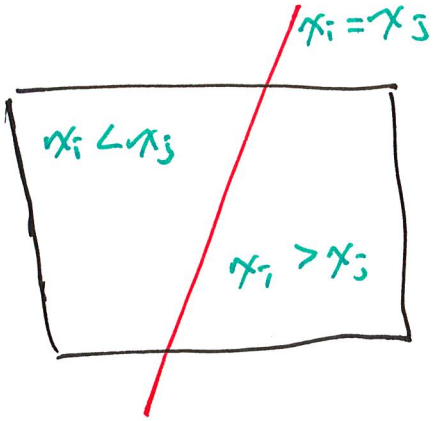
Theorem

Element Uniqueness requires $\Omega(n \log n)$ operations using a comparison decision tree model of computation.

Proof

- consider decision trees for element uniqueness inputs of size n on array of inputs $x_1, x_2, \ldots, x_n$

- we'll derive a lower bound by finding a lower bound on the height, h, of any possible decision tree.
- we can derive the height from a lower bound on the number of leaves.
- Consider an arbitrary decision tree
  - view each input a point in n-dimensional space $\mathbb{R}^n$
  - each decision defines a corresponding hyperplane



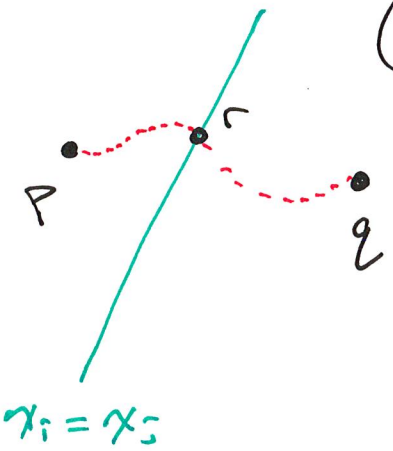
- an input reaches if it passes all classification tests along the path from the root.
- Let $W(v)$ be the set of all inputs that reach leaf v
- $W(v)$ is alway a convex region of $\mathbb{R}^n$ and hence path connected.
    - path connected: for any two points in $W(v)$ there is a continuous path that remains in $W(v)$

- Consider an arbitrary decision tree for element uniqueness and an input of n distinct numbers.
- This input ~~str~~ must be in some $W(v)$ for some leaf marked distinct.
  - Every permutation of this input must also be in some $W(v')$ for some leaf v' marked distinct.
  - Claim: All of these leaves must be different.

- assume that there are two different permutations $P, Q \in W(v)$ for some single leaf v

- Because these permutations must differ by swapping the order of at least 2 elements, there must exist i, j such that $x_i < x_j$ in p but $x_i > x_j$ in q.

- any continuous path between p and q must pass through the hyperplane $x_i = x_j$ at some point r (i.e some input where $x_i = x_j$)



- Because r is path connected to both p and q it must also be in $w(v)$ and would therefore produce the answer distinct.
- r must be labelled not distinct for the algorithm to be correct.
- All $n!$ permutations of the input must reach distict leaves.
- Since a Ternary tree of height h can have at most 3^h leaves we have

$$n! \leq \ell \leq 3^h$$

- So $h \geq \log_3(n!) \in \Omega(n \log n)$