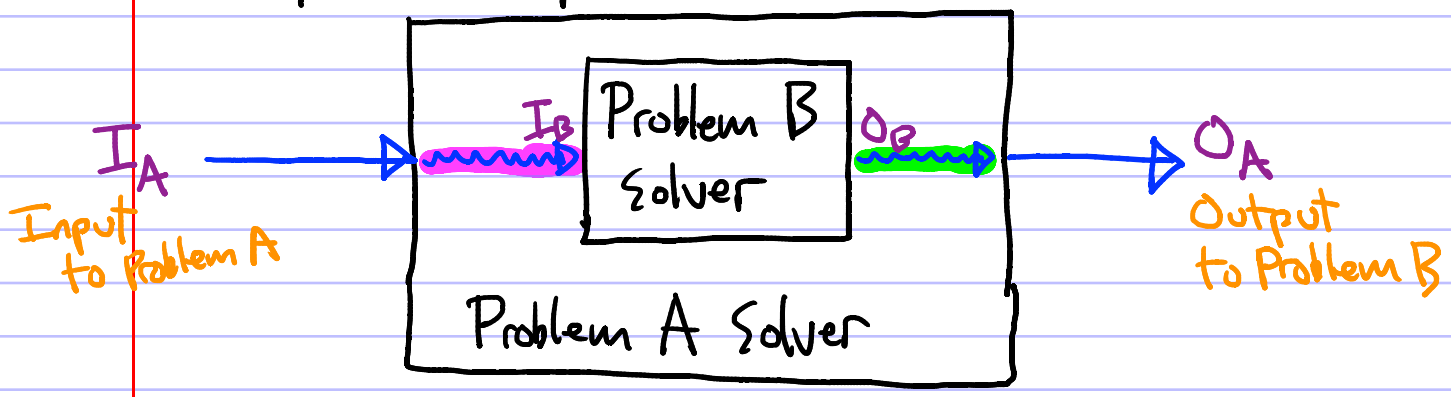
REDUCTIONS

One of the most prevalent concepts in the theory of computation:



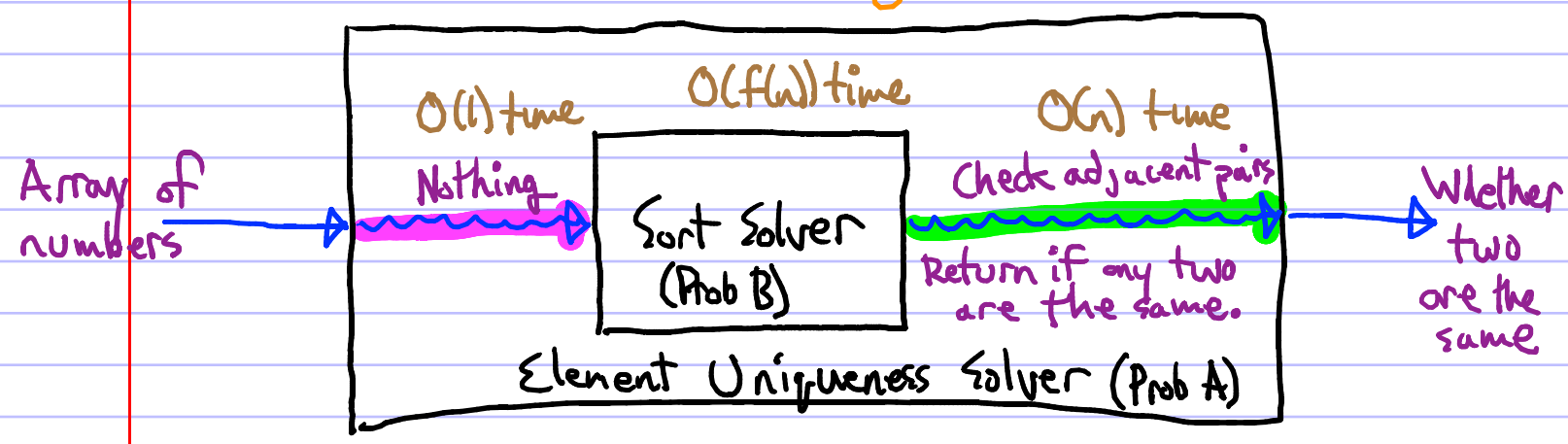
This way of building a "Problem A Solver" (a.k.a an algorithm for Problem A is called a reduction to Problem B.

The ⟿ and ⟿ specify the actual reduction.

Example:

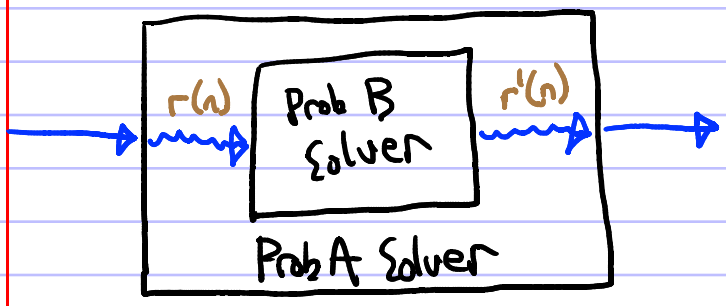
Problem A: Given an array of numbers, are any two the same? (Element Uniqueness)

Problem B: Given an array of numbers, sort them. (Sorting)



If ∃ an algorithm for sorting in $O(f(n))$ time, then ∃ an algorithm for element uniqueness in $O(f(n)+n)$ time. Say anything else?

A reduction implies two things:



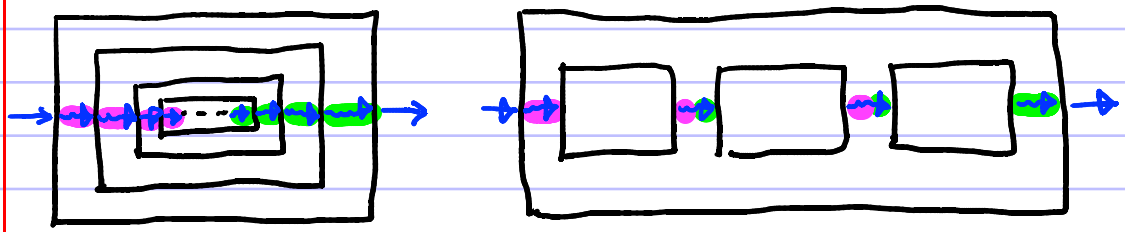
algorithms

#1: If ∃ an $O(f(n))$ solver for Problem B, UPPER BOUNDS
then ∃ an $O(r(n)+f(n)+r'(n))$ solver for Problem A.

#2: If ∄ an $O(r(n)+f(n)+r'(n))$ solver for Problem A,
then ∄ an $O(f(n))$ solver for Problem B. LOWER BOUNDS

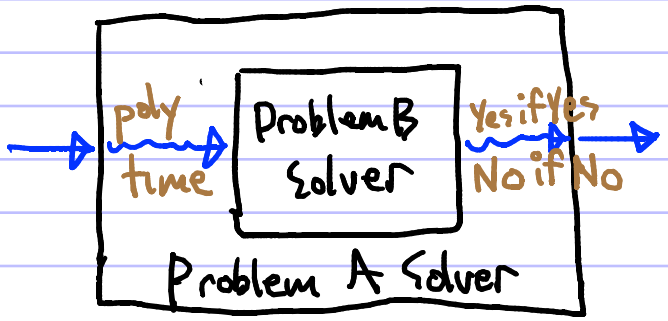
NP-hardness,
3SUM-hardness

Sequences / Nesting are also allowed, but give more complicated statements.



NP-hardness reductions are popular.

A problem B is NP-hard if ∀ problem A in the class NP
∃ a polynomial-time many-one reduction to B:



3SUM-HARDNESS

real RAM

Problem (3SUM): Given an array of integers S, do there exist three integers $a,b,c \in S$ such that $a+b+c=0$?

$[-20, 9, 4, -8, 1, 15, -11, -4, 3]$ YES

$[-4, -9, 8, 11, -1, 7, -20]$ NO

Algorithm for 3SUM: $S[j]-S[i]+S[k]-S[i]$

1. Sort S. $O(n \log n)$
2. For i from 1 to $S.size()$: $O(n)$

   Construct array R, where $R = S + S[i]$ $O(n)$

   Search R for two elements $R[j], R[k]$ such that $R[j]+R[k]=S[i]$ $O(n)$ using two-finger algorithm

$S = [-20, -11, -8, -4, 1, 3, 4, 9, 15]$

$R = S + S[9] = [-5, 4, 7, 11, 16, 18, 19, 24, 30]$ $R[2]+R[4]=15=S[9]$

Conjecture: Any algorithm for 3SUM takes $\Omega(n^2)$ time.

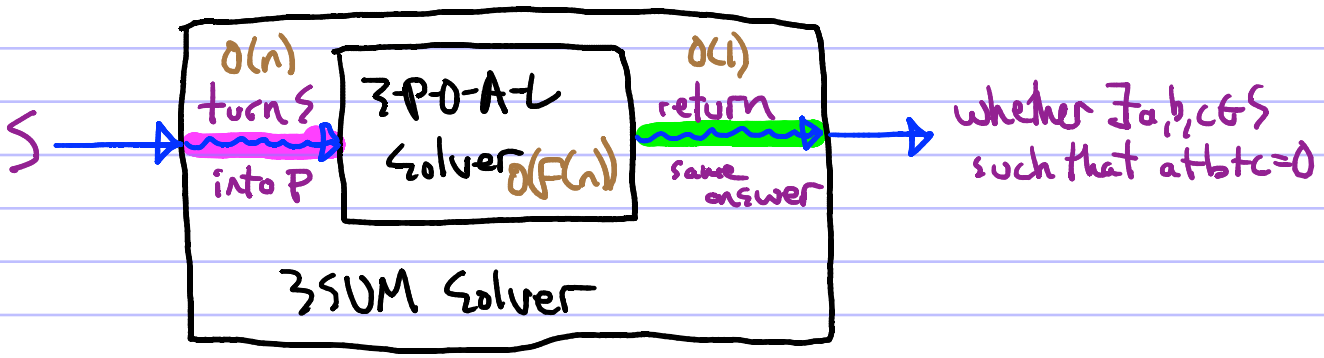
Evidence is that people have tried hard and failed plus other stuff

If I wanted to show (give evidence of) that Problem B also takes $\Omega(n^2)$ time to solve using any algorithm, I can use a reduction from 3SUM to Problem B. Such a reduction implies that Problem B is 3SUM-hard.

3SUM-hard PROBLEMS

Problem (3-Points-on-a-line): Given a set of n points in the plane, are any 3 collinear?

Reduction:
- $O(n)$: Take integers $S=\{x_1, x_2, x_3, \ldots, x_n\}$.
- $O(n)$: Create point set $P = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$
- $O(f(n))$: Feed P to 3-P-O-A-L solver
  - $O(1)$: If 3-P-O-A-L says yes, then return yes.
  - $O(1)$: Else return no.



3SUM solver works b/c $a+b+c=0$ iff $(a,a^2),(b,b^2),(c,c^2)$ collinear.

The reduction says "3-P-O-A-L is 3SUM-hard".

If any 3SUM solver must take $\Omega(n^2)$ time, then since this solver takes $O(n+f(n)+1)$ time, $f(n)=\Omega(n^2)$.

3SUM-hardness $\approx$ Takes $\Omega(n^2)$ time if 3SUM takes $\Omega(n^2)$ time.

This reduction also implies that if 3-P-O-A-L can be solved in $O(f(n))$ time, then 3SUM can be solved in $O(n+f(n)+1)$ time

A trivial $f(n)$ is $O(\binom{n}{2})=O(n^2)$, but we already had $O(n^2)$ 3SUM algo.

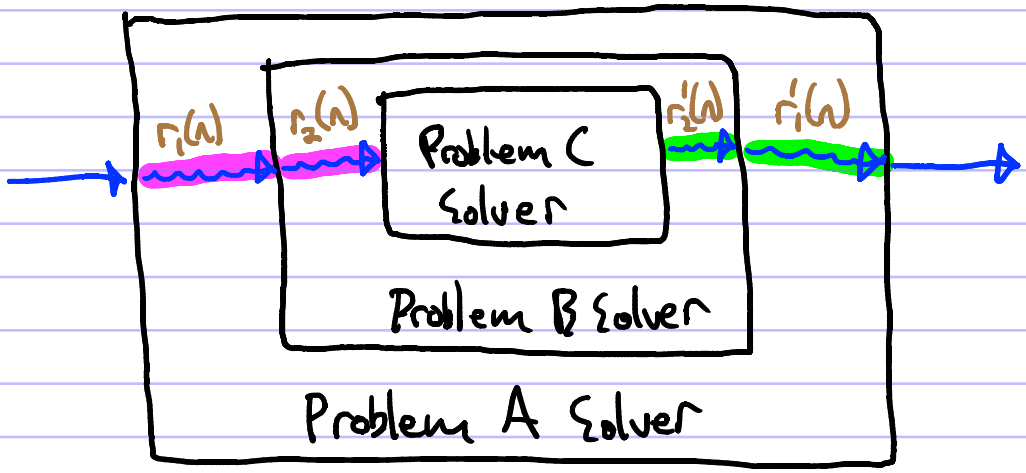
NP-hard: "as hard as any problem in class NP"

3SUM-hard: "as hard as 3SUM"

Problem (Minimum Area Triangle): given a set of n points in the plane, what is the smallest area triangle induced by 3 of the points?

Thm: M-A-T is 3SUM-hard. How to show this?

Suppose we nested reductions:



If Problem A takes $\Omega(n^2)$ time, then Problem C takes $\Omega(n^2)$ time provided $r_1(n)+r_2(n)+r_2'(n)+r_1'(n)$ is $o(n^2)$.

"The $\Omega(n^2)$ time has to be spent somewhere"

Claim: M-A-T is 3SUM-hard

Proof:

THE COMPLEXITY CLASS NP

A complexity class is the set of problems satisfying some conditions.

The complexity class P (polynomial-time) is the set of decision problems with polynomial-time algorithms.

Shortest path, max flow, longest common substring, integer multiplication, maximum matching, convex hull, minimum spanning tree, ...

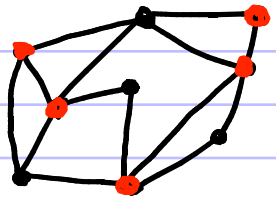
The complexity class NP (non-deterministic polynomial-time) is the set of decision problems with
#1 non-deterministic polynomial-time algorithms
#2 polynomial-time verifiable certificates of valid solutions

Everything in P + vertex cover, set cover, independent set, graph coloring, Hamiltonian path/cycle, dominating set

Why everything in P? Polynomial-time ⇒ Non-deterministic polynomial-time.

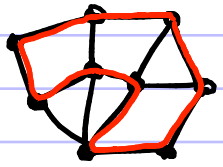
Why #1 equivalent to #2? A non-deterministic machine correctly guesses a certificate of a valid solution.

Vertex cover certificate:



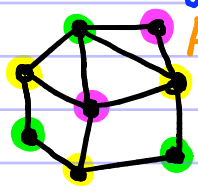
A cover

Hamiltonian cycle:



A Hamiltonian cycle

Graph coloring:

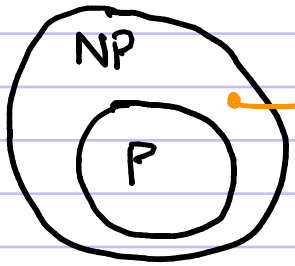


A coloring

Since P⊆NP, the Venn diagram is

Open if ∃ problem p∈NP & p∉P

P≠NP



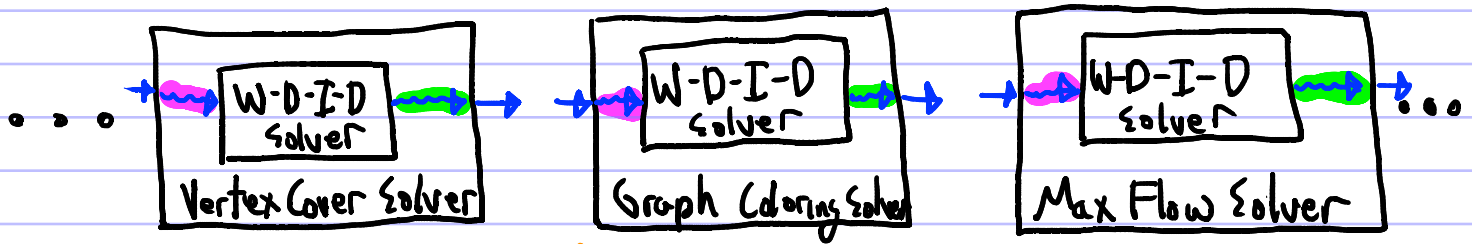
REVISITING NP-HARDNESS

3SUM-hardness requires being as hard as one problem. (1 reduction)

NP-hardness requires being as hard as all problems in class NP. (lots o' reductions)

How do we get away with one reduction?

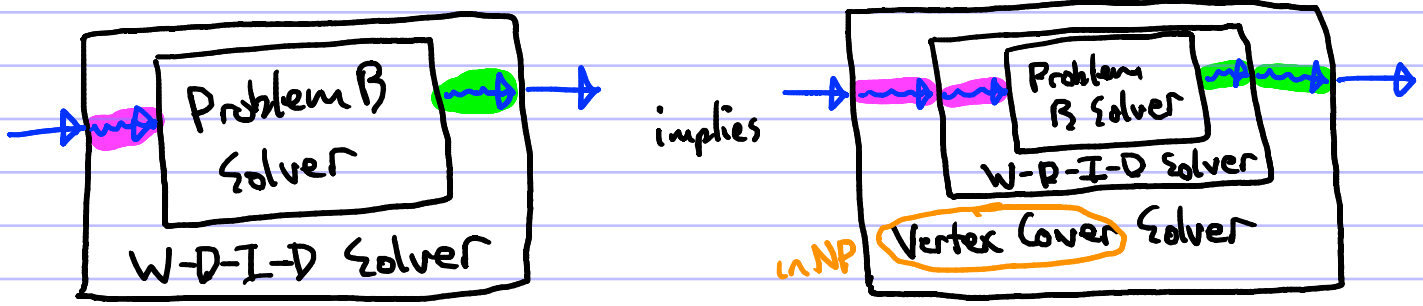
Suppose we started with one problem we could reduce any problem in NP to, called the What-Does-It-Do Problem.



What-Does-It-Do Problem (NP edition): given a non-deterministic polynomial time algorithm and an input to that problem, what does the algorithm do?

W-D-I-D is NP-hard by definition!

If we can reduce W-D-I-D to a problem B, then problem B is NP-hard provided reduction isn't doing the heavy lifting



Problem B is NP-hard provided reduction is not NP-hard.

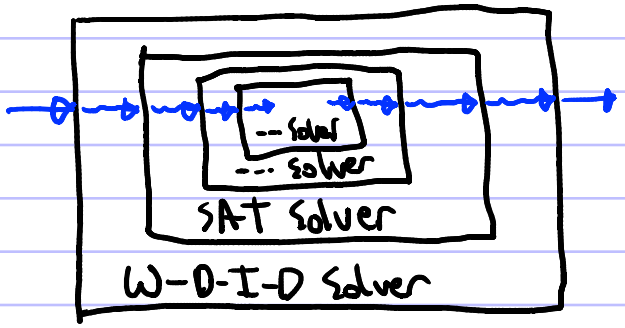
Specifically, provided reduction only takes polynomial-time.

(we think polynomial-time isn't enough to solve some problems in NP, so Problem B solver is doing the hard stuff)

Might ask "how to reduce W-D-I-D to a normal problem like vertex cover or hypergraph edge cover"

Someone already did this, [Cook 1971] [Levin 1973?] USSR reducing to SAT, a problem about boolean formulas and whether they are ever true.

People also reduced SAT to many other problems. [Karp 1972]



Still a polynomial-time reduction, since each rest takes polynomial-time and increases input size by polynomial factor.

These days, 1000s of NP-hard problems to reduce from.

Evidence for $P \neq NP$!

A polynomial-time algorithm for one implies a polynomial-time W-D-I-D solver, which we can use to solve any problem in NP in polynomial-time.

No one has found one for any of these problems.

NP-hardness is "just hardness for the class NP"

What about "hardness for the class P"?

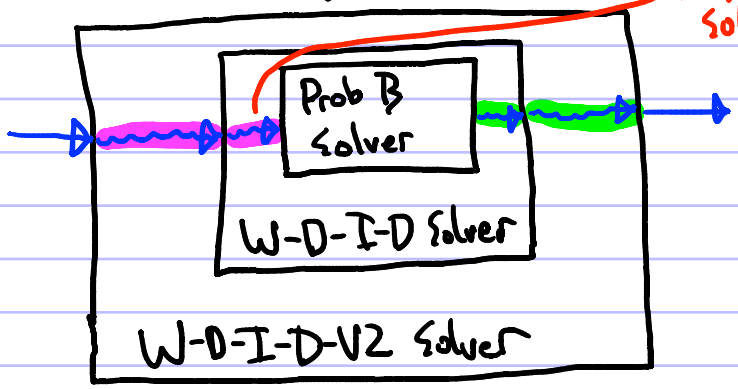
P-HARDNESS

(P edition)

What-Does-It-Do-Version-2: given a polynomial-time algorithm and an input, what does the algorithm do?

W-D-I-D-V2 is P-hard by definition!

Might want to say any NP-hard problem is P-hard.
But we can't say that:

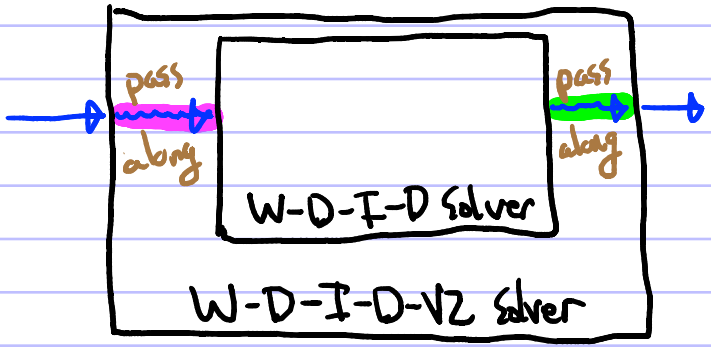


polynomial-time reduction can solve W-D-I-D-V2 by itself!

Can't conclude anything about Problem B, since it may not be doing the "heavy lifting"

To show a problem is P-hard, need a reduction we think isn't strong enough to do the "heavy lifting".

What's (probably) weaker than polynomial-time? Parallelizable polynomial-time, Log-space, constant-depth circuits.



Since pass along is probably not strong enough to solve W-D-I-D-V2, it must be W-D-I-D that is strong enough.

LOWER BOUNDS

Many lower bounds of the form "if conjecture X is true, then problem B takes Y time/space"

Ex: NP-hardness "if P≠NP, then vertex cover takes superpolynomial time"

Ex: 3SUM-hardness "if 3SUM takes $\Omega(n^2)$ time, then 3-POINT-LINE takes $\Omega(n^2)$ time"

Very few unconditional lower bounds.

- Sorting in comparison model takes $\Omega(n \log n)$ time. Counting argument
  (but in word RAM, $O(n \log\log n)$ time...)
  Comparison model: only operation on values is ≥. Pay only for these.

- Maximum, minimum, median, etc. in comparison, word RAM, others takes $\Omega(n)$ time. Adversary argument

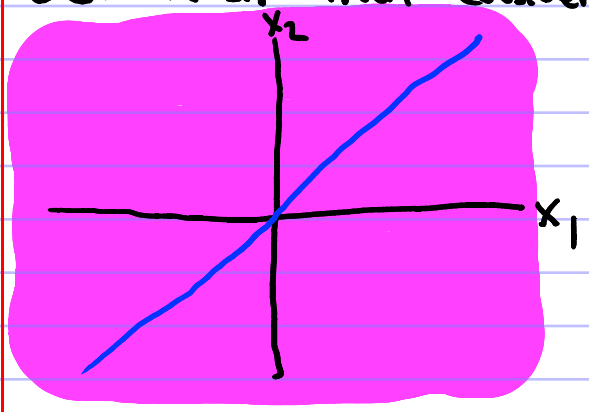
What about element uniqueness: given a collection of numbers, are any pair the same? (Comparison model)

Returns YES (collection is unique) or NO (collection not unique)

Thm: Let $C \subseteq \mathbb{R}^n$ and $\#C$ be the number of connected components of C. Then deciding membership in C takes time t for some input with $2^t \cdot 3^{n+t} \geq \#C$. [Ben-Or 1983]

A theorem linking computation time to complexity of solution space.

Let $n=2$. Then consider element uniqueness solution space C:



input is $[x_1, x_2]$

YES is region

NO is region

For higher n, more and more halfplanes through origin, one for each pair of elements.

$[x_1, x_2, \dots, x_n]$ gives halfplanes $x_i = x_j$ $\forall\ 1 \le i, j \le n$.

So the maximal connected component containing a point $(x_1, x_2, \dots, x_n) \subseteq \mathbb{R}^n$ is the set $\{(x_1, x_2, x_3, \dots, x_n) : x_1 < x_2 < x_3 < \dots < x_n\}$.

All points in sorted order

So $\#C = n!$.

Then $2^t \cdot 3^{n+t} \ge n!$

$$t \cdot \log_2 2 + (n+t) \cdot \log_2 3 \ge \log_2(n!) \ge \frac{n}{2} \log_2 \frac{n}{2}$$

$$n \cdot \log_2 3 + (1 + \log_2 3)\, t = \Omega(n \lg n)$$

$$c \cdot t = \Omega(n \log n - n) = \Omega(n \log n)$$

$$t = \Omega(n \log n)$$ Same as sorting!

Can use this same theorem for results on many problems.

Subset Sum Problem: given a set S of n real numbers $\exists S' \subseteq S$ such that $\sum_{x \in S'} x = 1$? Use 1 for technical reasons.

A generalization of 3SUM (replace "3" with "subset") except real numbers

How many connected components of the NO solution space:

$$\{(x_1, x_2, \ldots, x_n) : \nexists S' \subseteq \{x_1, x_2, \ldots, x_n\} \text{ with } \sum_{x \in S'} x = 1\}$$

Two points not in same connected component if

$(x_1, x_2 \ldots, x_n)$ with $\sum_{j=1}^{k} x_{ij} > 1$ and

$(x_1', x_2', \ldots, x_n')$ with $\sum_{j=1}^{k} x_{ij}' < 1$

Sum of this subset must cross over at some point.

weighted linear combination is more or less than 1

Can show that every threshold function is implemented by a component, and there are at least $\sqrt{2}^{n^2}$ such functions (and conn. components)

So $2^t \cdot 3^{n+t} \geq \sqrt{2}^{n^2}$

$$t \log_2 2 + (n+t) \log_2 3 \geq n^2 \log_2 \sqrt{2}$$

$$t = \Omega(n^2)$$

So deciding membership in NO set (and thus YES set) takes $\Omega(n^2)$ time