CS 173 Lecture 18a: Computational Complexity

Given a computational problem Q,
how quickly can we solve Q (wrt input size n)

Given an array $A[1..n]$, how quickly can we sort A?

$\exists$ algorithm "MergeSort" $O(n \log n)$ time.

$\exists$ algorithm that does better?

Know: if "comparison-based" then NO

Know: if all elements come from a set of size U.
"radix-sort" $O(nU)$.

Computational Complexity:

1) How good of alg can we give for Q?
2) Can we prove "lower bounds" for Q?
   ↪ we cannot get an alg that's as good...

Complexity Classes:

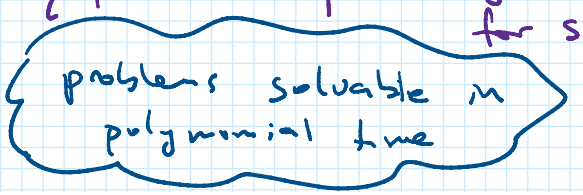
Classes (primarily) for Yes/No problems

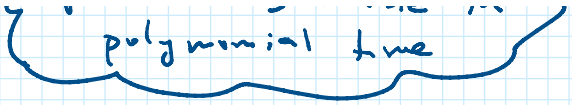
Given a graph G, & $\#k$, is $\chi(G) \leq k$?

Given a number n, is n prime?

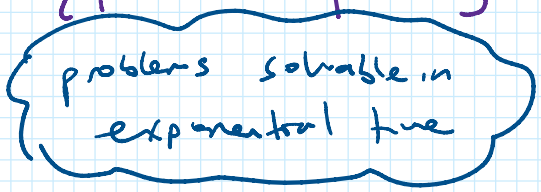
Given a graph G, vertices u, v, is uv an edge?

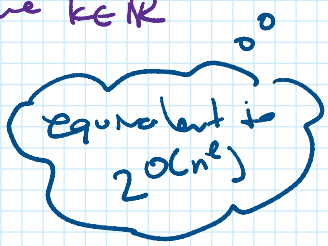
$P = \{$ problems Q | $\exists$ alg for Q w/ running time $O(n^k)$ for some $k \in \mathbb{R}$ $\}$





$EXP = \{$ problems $Q \mid \exists$ alg for Q w/ running time $O(k^n)\}$ for some $k \in \mathbb{R}$





Remark: $P \subseteq EXP$.

if alg A takes time $O(n^k)$, then A takes $2^{O(n)}$ because $n^k = O(2^{O(n)})$

Can think of P as (historically) problems solvable in a small amount of time

$EXP - P$ problems which we cannot.

Historical? Edit Distance widely used in text analysis computational biology

can be solved in $O(n^2)$ standard exercise in courses like CS374.

Unfortunately, in applications n is in the billions, if not trillions.

EXP - P? even for n in the millions, we don't have an alg that's useable & solves the problem properly.

→ heuristics, approximations for faster algorithms

→ usually the solution in practice.

On beyond:

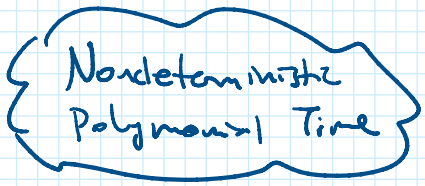
In Algebraic Geometry: Ideal Membership Problem ← polynomials.

known to not be in EXP.

→ no $2^{O(n)}$ alg for Ideal Membership

↳ workaround in practice, if needed:
convert to linear algebra
which is "in P".
(typically in $O(n^3)$).

Important Class "NP"



→ Part 6.

More complexity in general:
CS 374 & 473 talk about NP in more detail.
CS 579 & CS 598 Special Topics