What's NP-completeness mean?
What's $P=NP$ or $P\neq NP$ mean?
How do we show a problem is intractable?

"unlikely to be solved efficiently"

$P$ = class of all search problems that are solvable in polynomial time
$NP$ = class of all search problems.

Search problem:
- roughly - problem where we can efficiently verify solutions.

Formally: Given instance $I$ (e.g., graph $G$), we are asked to find a solution if one exists & if no solutions then output NO.

Moreover, if we are given solution $S$, we can verify in time polynomial in $|I|$ that $S$ is a solution to $I$. 
So given input instance \( I \) & solution \( S \)
we can verify that \( S \) is a solution to \( I \) efficiently = time \( \text{poly}(|I|) \).
if there are no solutions to \( I \) then we don't need to do anything.

How do we show this?

Give an algorithm that takes input \( (I, S) \)
& in \( \text{poly}(|I|) \) time verifies that 
\( S \) is a solution to \( I \)

(note, it must be that \( |S| = \text{poly}(|I|) \)
otherwise can't read all of \( S \).
Examples of search problems:

\[ k \text{-coloring:} \]

Input: graph \( G = (V, E) \) & integer \( k \geq 0 \)

Output: assignment of a color \( \{1, 2, \ldots, k\} \) to each vertex so that adjacent vertices get different colors, and NO if no \( k \)-coloring of \( G \).

Claim: \( k \)-coloring \( \in \text{NP} \)

Proof: Given \( G \) & a \( k \)-coloring \( \sigma : V \rightarrow \{1, 2, \ldots, k\} \) in \( O(n+m) \) time we can check every edge & verify that the endpoints get different colors.
SAT:

input: Boolean formula $f$ in CNF where $f$ has $n$ variables $x_1, \ldots, x_n$ & $m$ clauses $C_1, \ldots, C_m$

output: Satisfying assignment if one exists
              NO otherwise.

Claim: SAT in NP.

Proof: Given $f$ & an assignment $\sigma$,
in $O(nm)$ time we can check each clause & verify that at least one literal satisfied in every clause.

What does CNF mean? conjunctive normal form.

$\land =$ AND, $\lor =$ OR

AND of clauses & OR within a clause.

Example:

$$(x_3 \lor x_2) \land (x_4) \land (x_3 \lor \overline{x_4} \lor x_1) \land (x_2 \lor x_4 \lor x_1)$$

\[\uparrow\]

$(x_3 = F \lor x_2 = T) \land (x_4 = T) \land (x_3 = T \lor x_4 = F \lor x_1 = T) \land (x_2 = T \lor x_4 = T \lor x_1 = F)$
Knapsack:

**Input:** n objects with integer weights w₁,...,wₙ, integer values v₁,...,vₙ, and capacity B.

**Output:** subset S of objects with total weight ≤ B and maximum total value.

Is knapsack in NP?

Given instance \( \sum wᵢ,...,wₙ, vᵢ,...,vₙ, B \) & subset \( S \) in time \( O(n \log B) \) can verify that the total weight ≤ B. But how do we verify that it has max value?

Could run DP-alg. in \( O(nB) \) time to compare the value obtained from DP to \( S \)'s value.

But \( O(nB) \) is exponential in the input size \( |I| = O(n \log B) \) number B takes \( O(\log B) \) bits.
Standard form of knapsack is optimization version.

Search version:

**Knapsack-search:**

**Input:** \( w_1, \ldots, w_n, v_1, \ldots, v_n, B, \text{ and } \text{goal } g \)

**Output:** subset \( S \) with:

- Total weight \( \leq B \)
- Total value \( \geq g \)

No if no such \( S \) exists.

Claim: Knapsack-search \( \leq_{NP} \)

Proof: Given \( \{ w_1, \ldots, w_n, v_1, \ldots, v_n, B, g \} \) and \( S \),

in \( O(AB) \) time can verify that \( \sum_{i \in S} w_i \leq B \)

\& in \( O(n \log B) \) time can verify \( \sum_{i \in S} v_i \geq g \).
Note, knapsack \implies knapsack-search.

This means that if we can solve knapsack-search in poly-time, then we can use it as a black-box to solve knapsack in poly-time — just do binary search for max \( g \in \{1, \ldots, V \} \) which has a solution where \( V = \sum_{i=1}^{N} V_i \).

\[\text{MST:} \]
\[\text{input: } G = (V, E) \text{ with } w(e) > 0.\]
\[\text{output: } \text{tree } T \text{ with min weight.}\]

\[\text{Claim: } \text{MST} \in \text{NP}\]

\[\text{Proof: } \text{Given } G \& T, \text{ can run DFS to verify that } T \text{ is a tree. But how do we verify its min weight?}\]
\[\text{Run Kruskal's & compare its weight, this is } \text{poly}(n) \text{ time.}\]

Thus, \text{MST} \in \text{NP.}
$NP = \text{nondeterministic poly-time}$  
$
=\text{problems that can be solved in poly-time on a non-deterministic machine.}$

↑ allowed to guess at each step

(↑ an accepting path)

$NP = \text{all search problems} = \text{problems can verify solutions efficiently.}$

$P = \text{search problems that can be solved in poly-time.}$  
$= \text{problems can solve efficiently.}$

$P \subseteq NP$

$\text{solve efficiently} \rightarrow \text{verify efficiently}$

is verifying as hard as constructing solutions?

is verifying a proof as hard as constructing a proof?
Clearly, $P \neq NP$.

If $P = NP$ then if we can verify solutions efficiently then we can construct solutions efficiently.

If $P \neq NP$ then there are some search problems that can't be solved efficiently.

What problems can't be solved efficiently?

(assuming $P \neq NP$)

**NP-complete problems:** hardest problems in NP.

Colorings is NP-complete which means:

\[
\begin{aligned}
\text{a)} & \; \text{Colorings} \in NP \\
\text{b)} & \; \text{if we can solve colorings in polytime then we can solve every problem in NP in polytime.}
\end{aligned}
\]

Thus if $P \neq NP$, then no polytime algorithm for colorings.
How to show (b)?
Problems A & B (for example, A=MST, B=Colorings)

A → B (or A ≤ B)
Means we can reduce A to B.

if we can solve B in poly-time then we can use that algorithm as a black-box to solve A in poly-time.

Suppose there is an efficient (poly-time) alg. for B
We'll build an alg. for A:

Reducing BA to AB
A → B → BA (see even I mess it up!)
We need to define $f$ & $h$:

$f$: transform input for A to input for B
$h$: transform solution $S$ to B to solution for A

Then $h(S)$ for I

Need to prove that:

$S$ is a solution to $f(I)$

Then $h(S)$ is a solution for I

& if NO solution for $f(I)$

Then NO solution for I

in other words, $S$ is a solution to $f(I)$ \iff $h(S)$ is a solution to I.
To show Colorings is NP-complete, we need to show:

a) Colorings ∈ NP
b) for all A ∈ NP, A → Colorings.

How to do (b) for all A ∈ NP?

Suppose we know SAT is NP-complete.

Thus, A → SAT. for all A ∈ NP.

Suppose we show SAT → Colorings.

Then: A → SAT → Colorings so A → Colorings.

Therefore to show colorings is NP-complete:

a) Colorings ∈ NP
b) for a known NP-complete Problem A, show A → Colorings.