Key questions:
- What's NP-completeness mean?
- What's \( P = NP \) or \( P \neq NP \) mean?
- How do we show that a problem is intractable? Intractable = unlikely to be solved efficiently

Definitions:

\[ P = \text{class of all search problems that can be solved in polynomial time} \]

\[ NP = \text{class of all search problems (regardless of time required to solve)} \]

What's a search problem?
Roughly, a problem where we can efficiently verify solutions

Given a solution, we can verify this is a solution in polynomial-time.
Formally, what is a search problem?

Search problem:

has the following input/output form:

Given instance $I$ (e.g., graph $G$)

we are asked to find a solution if one exists and if no solution exists we output NO.

Additional requirement:

if we are given a solution $S$ for instance $I$,
then we can verify (i.e., check) that $S$ is a solution to $I$ in time polynomial in $|I|$. 

So $P=NP$ or $P\neq NP$ addresses whether or not:

Solving a problem (i.e., constructing a solution) is as easy as verifying a solution.
To show a problem $P$ is a search problem, we need to show an algorithm $A$ that takes as input $(I, S)$ & in poly-time verifies that $S$ is a solution to $I$.

Examples of search problems:

$k$-coloring:

- input: undirected $G = (V, E)$ & integer $k > 0$.
- output: Assign each vertex to a color in $P_1, ..., P_k$ so that adjacent vertices get different colors, and output NO if no such $k$-coloring exists for $G$.

Given $G$ & a $k$-coloring $\sigma$ (so $\sigma(v)$ is the color for vertex $v$) then in $O(|V| + |E|)$ time we can verify that $\sigma$ is a valid coloring.

Hence, coloring $\in \text{NP}$.
**SAT:**

**Input:** Boolean formula \( f \) in CNF with \( n \) variables & \( m \) clauses

**Output:** Satisfying assignment if one exists
No otherwise.

**SAT \∈\ NP:**

Given \( f \) and assignment \( \sigma \),
in \( O(n) \) time/clause can verify that the clause is satisfied & hence in \( O(nm) \) total time we can verify that \( \sigma \) satisfies \( f \).

**Knapsack:**

**Input:** \( n \) objects with integer weights \( w_1, \ldots, w_n \) & integer values \( v_1, \ldots, v_n \) & capacity \( B \)

**Output:** Subset \( S \) of objects with total weight \( \leq B \) & maximum total value.
Is knapsack $\in \mathsf{NP}$?

Given instance $\{w_1, \ldots, w_n; v_1, \ldots, v_n; B\}$ & solution $S$ in $O(n)$ time can check total weight is $\leq B$

but how do we verify that $S$ has maximum value?

(need to do it in time poly $(n, \log B)$

but only approach is to run knapsack to find an optimal solution $S$ which takes time $\text{poly}(n, B)$)

This is the optimization version, not search.

Look at search version:

**Knapsack-search**: as before

- input: $w_1, \ldots, w_n; v_1, \ldots, v_n; B$ & goal $g$
- output: subset $S$ with
  - total weight $\leq B$
  - total value $\geq g$
  - \text{NO} if no such $S$ exists.

**Knapsack-search $\in \mathsf{NP}$**: Given instance & solution $S$, in poly-time $O(n)$ can check that it has total weight $\leq B$ & total value $\geq g$. 
**Note:** if we can solve the search version in poly-time then we can solve the optimization version in poly-time by binary search over $g(T), \ldots, Vg$.

So knapsack $\rightarrow$ knapsack-search.

---

**MST:**

**input:** $G = (V,E)$ with $w(e) \geq 0$ for $e \in E$.

**output:** tree $T$ with min weight.

---

**MST in NP:**

Given $G$ & $T$, run Kruskal's or Prim's alg. to find in poly($n$) time a MST $T'$. Check that $w(T) = w(T')$ & then run BFS/DFS to check that $T$ is a tree.
NP stands for nondeterministic polynomial time

= problems that can be solved in poly-time
or a non-deterministic machine

↑

allowed to guess at each step
(there exists a path to the accepting state)

NP = all search problems.
P = search problems that can be solved in poly-time

Hence, \( P \subseteq NP \).

\[ \text{NP} \]

if \( P = \text{NP} \):
means that if we can verify solutions efficiently
then we can construct solutions efficiently.
(e.g., verifying a proof is as hard as constructing a proof)

if \( P \neq \text{NP} \):
means there are some search problems that can't be solved in poly-time.
If $P \neq NP$: what problems can't be solved in poly-time?

$NP$-complete problems = hardest problems in $NP$ = most difficult to solve.

SAT is $NP$-complete

This means:

a) $SAT \in NP$

b) if we can solve $SAT$ in poly-time then we can solve every problem in $NP$ in poly-time.

Thus if $P \neq NP$ then there is no poly-time algorithm for $SAT$.

How to show (b)?
Reductions:

Problems A & B (example: A=MST, B=SAT or A=2SAT, B=SCC)

A $\Rightarrow$ B or A $\leq$ B

means: A reduces to B

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

Algorithm for A

I \rightarrow f(I) \rightarrow Algorithm for B \rightarrow S \rightarrow h \rightarrow h(s) \rightarrow NO

Take input I for A:
1) define input $f(I)$ for B
2) run black-box alg. for B on $f(I)$
3) given solution $S$ for $f(I)$ produce solution $h(S)$ for I
& given NO for $f(I)$ then output NO for I.
To reduce \( A \rightarrow B \),

need to define \( f \) & \( h \)

\& prove that

if \( S \) is a solution to \( f(I) \)
then \( h(S) \) is a solution to \( I \)

\& if no solution for \( f(I) \)
then no solution for \( I \).

To show:
- \( \text{SAT} \) is \( \text{NP} \)-complete

need to show:

a) \( \text{SAT} \in \text{NP} \)
b) for all \( A \in \text{NP} \)
\( A \rightarrow \text{SAT} \).

How to do (b)?
Suppose we know SAT is NP-complete (somehow?)
So for every \( A \in \text{NP} \), \( A \rightarrow \text{SAT} \).

Suppose we want to show:

Colorings is NP-complete.
If we show:

\( \text{SAT} \rightarrow \text{Colorings} \)

Then \( A \rightarrow \text{SAT} \rightarrow \text{Colorings} \)

Hence, to show Colorings is NP-complete, we need to show:

a) Colorings \( \in \text{NP} \)

& b) for a known NP-complete problem \( B \), \( B \rightarrow \text{Colorings} \).