Next section:

- What's NP-completeness mean?
- What's P=NP or P≠NP mean?
- How do we show that 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 (regardless of time required to solve them)

Search problem:

Roughly - problem where we can efficiently verify solutions

So, P=NP or P≠NP addresses whether or not:

Solving a problem is as easy as verifying a solution
Formally, what is a search problem?

Search problem:

Given instance $I$ (e.g., graph $G$), we are asked to find a solution if one exists, and if none exists, we output NO.

For a search problem, if we are given a solution $S$, we are given a solution $S$, we can verify in time polynomial in $|I|$ that $S$ is a solution to $I$.

So when there is a solution, we can verify it is a solution efficiently.

If there are NO solutions, we do not need to do anything.

Need that: $|S| \leq \text{poly}(|I|)$ and there is an algorithm $A$ that takes $I$ & $S$ as input, and in polynomial-time verifies that $S$ is a solution to $I$. 
Examples of search problems:

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

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

**Output:** assignment of \( k \) colors to the vertices \( V \) so that adjacent vertices get different colors, and \( \text{NO} \) if no \( k\text{-coloring} \) of \( G \).

Given \( G \) & a \( k\text{-coloring} \), in \( O(n^k) \) time we can verify that it is a valid coloring.

Hence, coloring \( \in \text{NP} \).

\[ \text{SAT}: \]

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

**Output:** satisfying assignment, if one exists \( \text{NO} \) otherwise.

Given \( f \) and an assignment in \( O(nm) \) time we can verify that \( f \) is satisfied (spend \( O(n) \) time per clause).

Hence, SAT \( \in \text{NP} \).
Knapsock:

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.

Given instance \( \{w_1, \ldots, w_n; v_1, \ldots, v_n; B\} \) & subset \( S \), we can verify that the total weight is \( \leq B \) in \( O(n) \) time, but how do we verify that has maximum value? Not clear how to do it. It's an optimization problem.
Look at search version:

**Input:** \( w_1, \ldots, w_n, v_1, \ldots, v_n, B \) \& goal \( q \).

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

- Total weight \( \leq B \) \( (\sum_{i \in S} w_i \leq B) \)
- Total value \( \geq q \) \( (\sum_{i \in S} v_i \geq q) \)

& No if no such \( S \) exists.

Given \( S \) in \( O(n) \) time can check that it has total weight \( \leq B \) & total value \( \geq q \).

\[\Rightarrow\text{Knapsack-search \( \in \) NP.}\]

*Note:* If we can solve the search version in poly-time then we can solve the optimization version by doing binary search for max \( q \in \{1, \ldots, V\} \) which has a solution.
MST:
input: \( G = (V,E) \) with positive edge lengths
output: Tree \( T \) with min weight.

MST is a search problem.
Hence, MST \( \in \text{NP} \).

Why?
Given \( G \) \& \( T \), we can run Kruskal's or Prim's to verify that \( T \) has min weight.
Then run BFS or DFS to verify that \( T \) is a tree.