Consider the decision problem (or language) Composite defined by:

\[ \{ x \mid x \text{ is not prime} \} \]

This is the complement of the language Prime.

Is Composite $\in P$?

Clearly, the answer is yes if, and only if, Prime $\in P$. 
Hamiltonian Graphs

Given a graph $G = (V, E)$, a Hamiltonian cycle in $G$ is a path in the graph, starting and ending at the same node, such that every node in $V$ appears on the cycle exactly once.

A graph is called Hamiltonian if it contains a Hamiltonian cycle.

The language $\text{HAM}$ is the set of encodings of Hamiltonian graphs.

Is $\text{HAM} \in P$?
Examples

The first of these graphs is not Hamiltonian, but the second one is.
Graph Isomorphism

Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, is there a bijection

$$\iota : V_1 \rightarrow V_2$$

such that for every $u, v \in V_1$, $$(u, v) \in E_1 \quad \text{if, and only if,} \quad (\iota(u), \iota(v)) \in E_2.$$ 

Is Graph Isomorphism $\in \mathbb{P}$?
The problems Composite, SAT, HAM and Graph Isomorphism have something in common.

In each case, there is a search space of possible solutions.

the numbers less than x; truth assignments to the variables of $\phi$; lists of the vertices of $G$; a bijection between $V_1$ and $V_2$.

The size of the search is exponential in the length of the input.

Given a potential solution in the search space, it is easy to check whether or not it is a solution.
A verifier $V$ for a language $L$ is an algorithm such that

$$L = \{ x \mid (x, c) \text{ is accepted by } V \text{ for some } c \}$$

If $V$ runs in time polynomial in the length of $x$, then we say that $L$ is polynomially verifiable.

Many natural examples arise, whenever we have to construct a solution to some design constraints or specifications.
If, in the definition of a Turing machine, we relax the condition on $\delta$ being a function and instead allow an arbitrary relation, we obtain a nondeterministic Turing machine.

$$\delta \subseteq (Q \times \Sigma) \times ((Q \cup \{\text{acc, rej}\}) \times \Sigma \times \{R, L, S\}).$$

The yields relation $\rightarrow^*_M$ is also no longer functional.

We still define the language accepted by $M$ by:

$$\{x \mid (s, \triangleright, x) \rightarrow^*_M (\text{acc, w, u}) \text{ for some w and u}\}$$

though, for some $x$, there may be computations leading to accepting as well as rejecting states.
With a nondeterministic machine, each configuration gives rise to a tree of successive configurations.
We have already defined \( \text{TIME}(f) \) and \( \text{SPACE}(f) \).

\( \text{NTIME}(f) \) is defined as the class of those languages \( L \) which are accepted by a \textit{nondeterministic} Turing machine \( M \), such that for every \( x \in L \), there is an accepting computation of \( M \) on \( x \) of length \( O(f(n)) \), where \( n \) is the length of \( x \).

\[
\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)
\]
For a language in $\text{NTIME}(f)$, the height of the tree can be bounded by $f(n)$ when the input is of length $n$. 
A language $L$ is polynomially verifiable if, and only if, it is in $\text{NP}$.  

To prove this, suppose $L$ is a language, which has a verifier $V$, which runs in time $p(n)$. 

The following describes a *nondeterministic algorithm* that accepts $L$

1. input $x$ of length $n$
2. nondeterministically guess $c$ of length $\leq p(n)$
3. run $V$ on $(x, c)$
In the other direction, suppose $M$ is a nondeterministic machine that accepts a language $L$ in time $n^k$.

We define the *deterministic algorithm* $V$ which on input $(x, c)$ simulates $M$ on input $x$.

At the $i^{\text{th}}$ nondeterministic choice point, $V$ looks at the $i^{\text{th}}$ character in $c$ to decide which branch to follow.

If $M$ accepts then $V$ accepts, otherwise it rejects.

$V$ is a polynomial verifier for $L$. 
We can think of nondeterministic algorithms in the generate-and-test paradigm:

Where the *generate* component is nondeterministic and the *verify* component is deterministic.
Given two languages $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$,

A *reduction* of $L_1$ to $L_2$ is a *computable* function

$$f : \Sigma_1^* \to \Sigma_2^*$$

such that for every string $x \in \Sigma_1^*$,

$$f(x) \in L_2 \text{ if, and only if, } x \in L_1$$