P and NP
We say a deterministic TM has time-complexity $T(n)$ if for every input $w$ with length $|w| = n$ the TM halts (whether or not it accepts $w$) after $T(n)$ steps. The class $\mathcal{P}$ is \{ $L$ | $L$ is a language accepted by some TM with polynomial time complexity \}
We say that a non-deterministic TM has time-complexity $T(n)$ if for every input $w$ with length $n$ the TM can halt after $T(n)$ steps, in an Accept state if the TM accepts $w$. The class $\mathcal{NP}$ is $\{ L \mid L$ is a language accepted by some non-deterministic TM with polynomial time complexity\}
While you can ask if any language is in $\mathcal{P}$ or $\mathcal{NP}$ we are often interested in algorithmic questions such as "Find the shortest path from node $q_1$ to node $q_2$ in this weighted graph." That translates to a $\mathcal{P}$ or $\mathcal{NP}$ question by looking at the language \{g1110^n | g is an encoding of a weighted graph and the graph has a path of length $n$ or less from node $q_1$ to node $q_2$\}. A TM might determine if g1110^n for a particular graph g and a particular n is in this language by finding a path from $q_1$ to $q_2$ with length n.
Note that a non-deterministic TM can solve this by guessing the sequence of nodes on the shortest path from $q_1$ to $q_2$ and then verifying in polynomial time that these nodes do form a path from $q_1$ to $q_2$ and that the sum of the lengths of the edges on this path is no more than $n$. 
Many people describe $P$ as the set of problems that can be solved in polynomial time while $NP$ is the set of problems for which a solution can be verified in polynomial time.

It is obvious that $P$ is a subset of $NP$. Perhaps the most important unsolved question in CS is: Is $P = NP$? This question arises from Cook's (or Cook-Levin) Theorem, which says that if one specific language $L$ is in $P$ then $P=NP$. 
Let $L$ be a language in $NP$. We say $L$ is NP-complete if for every language $A$ in $NP$ there is a polynomial time reduction of $A$ to $L$ in the sense that we can covert any string $w$ in polynomial time to a string $w'$ so that $w$ is in $A$ if and only if $w'$ is in $L$. A polynomial-time decider for $L$ then gives us a polynomial-time decider for every language $A$ in $NP$. 
In other words, if L is NP-complete and L is in $\mathcal{P}$, then every problem that can be verified in polynomial-time could actually be solved in polynomial-time. That would have enormous ramifications.
We say a language $L$ is NP-hard if every language $A$ in $\mathcal{NP}$ reduces to $L$. So to be NP-complete a language must be

a) In $\mathcal{NP}$

b) NP-hard
Boolean expressions.
We will use $\land$, $\lor$, and $\neg$ to represent the Boolean operators and, or, and not.

Definition: A Boolean expression is
a) A variable that can have value T or F
b) $e \land f$, $e \lor f$, $\neg e$, or $(e)$ where $e$ and $f$ are Boolean expressions

For example, $x \land \neg(y \lor z)$ is a Boolean expression.
Given values of the variables we can find the value of this expression: build a parse tree for it (linear time) and pass the Boolean values up the tree from the leaves to the root:
Given a Boolean expression we can find if there is a set of assignments to its variables for which the expression evaluates to T. We say such an expression is *satisfiable*. For example, we could build a truth table for it:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>( x \land \sim(y \lor z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Unfortunately, a truth table with $k$ variables has $2^k$ lines so it can't be completed in polynomial time.
SAT is the language of satisfiable Boolean expressions.

Ex: \( x \land \sim(y \lor z) \) is in SAT: take \( x=T, \ y=F, \ z=F \)
Ex: \( x \land \sim y \land (y \lor \sim x) \) is not in SAT

It is easy to see that SAT is in $\mathcal{NP}$: Guess the right values of the variables and verify them by evaluating a parse tree for the expression. This takes linear time.

To prove Cook's Theorem we need to show that every $\mathcal{NP}$ problem reduces in polynomial time to SAT.
Let L be any language in NP. This means there is a non-deterministic TM M that accepts L and M halts on any input w in time p(|w|) for some polynomial p.

To prove Cook's Theorem we will produce from M and w a Boolean expression that is satisfiable if and only if M accepts w.

Suppose w is any string with |w| = n and M is any TM. If M accepts w there is a sequence of configurations $\alpha_0 \alpha_1 \ldots \alpha_{p(n)}$ so that

a) $\alpha_0$ is the initial configuration for the computation of M on w
b) Each $\alpha_i \Rightarrow \alpha_{i+1}$
c) $\alpha_{p(n)}$ is a configuration in an accept state.
We will create a Boolean expression $B$ that is satisfiable if and only if such a sequence of configurations is possible. So if SAT is in $P$ we can show $L$ is in $P$:

a) Start with a nondeterministic TM that accepts $L$

b) For any string $w$ construct $B$ in polynomial time

c) determine if $B$ is in SAT in polynomial time

d) $B$ is in SAT if and only if $w$ is in $L$
Note that we need to construct $B$ in polynomial time, so it is important that $|B|$ be a polynomial function of $|w|$.

In $k$ steps we can write at most $|w| + k$ symbols on the tape so we'll assume the non-blank portion of the tape is no longer than $p(n)$.

Also, we'll assume the TM runs exactly $p(n)$ steps for any input $w$ with $|w| = n$. 
Here is some notation we'll use:

\( X_{ij} \) is the \( j^{th} \) symbol of the \( i^{th} \) configuration. If the 4\(^{th} \) configuration is 11q\(_2\)00 then \( X_{30} = 1, X_{31} = 1, X_{32} = q_2, X_{33} = 0, \) and \( X_{34} = 0 \)

For any tape symbol or state \( A \), \( Y_{ijA} \) is a Boolean variable whose intuitive meaning is "\( X_{ij} = A \)"

We will assume the start state of any TM is \( q_1 \).
The Boolean expression we will construct is $B = S \land N \land F$ where

- $S$ says the first configuration is $q_1w$ (where $q_1$ is the start state of the TM)
- $N$ says each configuration is derived from the previous one.
- $F$ says that in the $p(n)^{th}$ configuration the TM is in a final state

$S$ and $F$ are easy; $N$ takes some work.
Step 1: If input $w$ is $a_1a_2...a_n$ then 
$$S = Y_{00q_1} \land Y_{01a_1} \land Y_{02a_2}... \land Y_{0na_n}$$

Step 2: Let $q_{f_1}..q_{f_k}$ be all of the final states of $M$.
Let $F_{ji}$ be $Y_{p(n)j}q_{fi}$ This says the $j^{th}$ symbol of the last configuration is $q_{fi}$
Let $F_j$ be $F_{j1} \lor F_{j2} \lor .. \lor F_{jk}$ This says the $j$th symbol of the last configuration is a final state.
Finally, $F$ is $F_0 \lor F_1 \lor ... \lor F_{p(n)}$ this says the TM accepts $w$.

Note that $|F_j|$ is independent of $w$, so $|S|$ and $|F|$ are both $O( p(n) )$
Step 3: We only need $N$, which says that each configuration is derived from the previous one. In fact, we'll make

$$N = N_0 \land N_1 \land \ldots \land N_{p(n)-1}$$

where $N_i$ says that configuration $i+1$ is derived from configuration $i$. 
To make $N_i$ we need two kinds of subexpressions:

$A_{ij}$ will say that the state symbol of the $i$th configuration is at position $j$ and also that the $j-1^{st}$, $j^{th}$, and $j+1^{st}$ symbols of the $i+1^{st}$ configuration are correct for the corresponding transition of $M$.

$B_{ij}$ will say that either the state symbol of the $i^{th}$ configuration is at position $j-1$ or $j+1$ (and so symbol $j$ is covered by $A_{ij}$) or else position $j$ has a tape symbol that is copied correctly from configuration $i$ to configuration $i+1$.

Given these, $N_i = (A_{i0} \lor B_{i0}) \land (A_{i1} \lor B_{i1}) \land ... \land (A_{ip(n)} \lor B_{ip(n)})$
Let's pause for an example. Suppose the $i^{th}$ configuration is $010q_110$ and $M$ has transition $\delta(q_1,1) = (q_2,1,R)$. We want the $i+1^{st}$ configuration to be $0101q_20$.

$B_{i0}$ will say the initial 0 is copied correctly
$B_{i1}$ will say the 1 is copied correctly
$B_{i2}$ will say T
$B_{i3}$ will say F
$A_{i3}$ will say $0q_11$ is changed to $01q_2$
$B_{i4}$ will say T
$B_{i5}$ will say the final 0 is copied correctly
To make $B_{ij}$, let $t_1...t_k$ be all of the tape symbols and $q_1..q_m$ all of the states.

$$B_{ij} = (Y_{i(j-1)q_1} \lor Y_{i(j-1)q_2} \lor ... \lor Y_{i(j-1)q_m}) \lor (Y_{i(j+1)q_1} \lor Y_{i(j+1)q_2} \lor ... \lor Y_{i(j+1)q_m})$$

$$\lor [ (Y_{i(j)jt_1} \land Y_{(i+1)(j+1)t_1}) \lor (Y_{i(j)t_2} \land Y_{(i+1)(j+1)t_2}) \lor ... \lor (Y_{i(j)t_k} \land Y_{(i+1)(j+1)t_k}) ]$$

Note that $|B_{ij}|$ has nothing to do with the input $w$. 
$A_{ij}$ describes the legal transitions.

Suppose we have a move to the right: $\delta(q_s,a) = (q_t,b,R)$

If the $i^{th}$ configuration is $\alpha c q_s a \beta$ with $q_s$ at position $j$, we want the $i+1^{st}$ configuration to be $\alpha c b q_t \beta$

The phrase of $A_{ij}$ for this is $p = Y_{ijq_s} \land Y_{i(j+1)a} \land Y_{(i+1)j} \land Y_{(i+1)(j+1)q_t} \land [(Y_{i(j-1)t1} \land Y_{(i+1)(j-1)t1}) \lor \ldots \lor (Y_{i(j-1)tk} \land Y_{(i+1)(j-1)tk})]$
On the other hand suppose we have a move left: \( \delta(q_s,a) = (q_t,b,L) \)

If the \( i^{th} \) configuration is \( \alpha cq_s a \beta \) with \( q_s \) at position \( j \), we want the \( i+1^{st} \) configuration to be \( \alpha q_t cb \beta \). The phrase of \( A_{ij} \) for this is

\[
p = Y_{ij}q_s \land Y_{(i+1)(j-1)}q_t \land Y_{(i+1)(j+1)}a \land Y_{(i+1)(j+1)b} \\
\land [(Y_{i(j-1)t1} \land Y_{(i+1)j1t}) \lor \ldots \lor (Y_{i(j-1)tk} \land Y_{(i+1)j1tk})]
\]

If \( M \) has \( L \) transitions and \( p_{ijt} \) is the corresponding \( A_{ij} \) phrase for transition \( t \) then

\[
A_{ij} = p_{ij1} \lor p_{ij2} \lor \ldots \lor p_{ijL}
\]
This completes the construction. Note that this seamlessly incorporates the nondeterminism of the TM: SAT's question about whether some assignment of variables satisfies B corresponds to the nondeterministic question of whether there is some valid sequence of configurations that gets to a terminal state.

Now, how big is B? \( B = S \land N \land F \)

\[ |S| = O(n) \]
\[ |F| = O(p(n)) \]
\[ |N| = O(p^2(n)) \]

This completes the proof that SAT is NP-complete.