Reducibility, P and NP

Preliminary definition: A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Key Definition (mapping reducible): Language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$ such that: $w \in A \iff f(w) \in B$.

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable.

Proof: Let M be the decider for B. Construct N for A like so: On input w, compute $f(w)$ and run M on it, outputting whatever M does.

Key Definition (TIME): Let TIME(t(n)), the time complexity class, be:
TIME(t(n)) = { L | L is a language decided by an O(t(n)) Turing machine}.

Key Definition (NTIME): Let NTIME(t(n)), the time complexity class, be:
NTIME(t(n)) = { L | L is a language decided by an O(t(n)) nondeterministic Turing machine}.

Key definition (P): The class of languages P are decidable in polynomial time on a deterministic single-tape Turing machine, or:
$P = \cup_k \text{TIME}(n^k)$.

Key definition (NP): The class of languages NP are decidable in polynomial time on a nondeterministic Turing machine, or:
$NP = \cup_k \text{NTIME}(n^k)$.

Theorem: Every language in P is also in NP.

Key definition (NP): The class of languages NP have polynomial time verifiers.

Theorem (not proven here): The two definitions given for NP are equivalent (assuming one can prove the other, and vice-versa).

Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$, where p denotes a polynomial time mapping reduction function.

Theorem: If $A \leq_p B$ and $B \in NP$, then $A \in NP$.

Key definition (NP-complete): A language B is NP-complete if it is in NP and every other language A in NP is polynomial time reducible to B.

Theorem: If B is NP-complete and $B \in P$, then P = NP.

Theorem: If B is NP-complete and $B \leq_p C$ for C in NP, then C is NP-complete.