Problem 7.28

Show that if P = NP we can factor integers in polynomial time.

- Consider the language
  \[ L = \{ \langle n, a, b \rangle | n \text{ has a factor } p \text{ in the range } a \leq p \leq b \} \]
  - \( L \) is obviously in NP, since the factor can serve as the certificate.
  - Since we're assuming P=NP, there is a polynomial algorithm that decides the above language.
  - Repeated applications of the algorithm allow us to divide our search space in half each time by asking “Is there a factor in the range \((a, a + b/2)\)?” If there isn’t we know there is a factor in the other range.
  - The total number of times we have to apply the algorithm is equal to \( \log n \), or in other words \( O(k) \) if \( k \) is the number of bits of \( n \). So a polynomial number of applications of this algorithm allows us to isolate one factor.
  - Since there are at most \( O(k) \) factors as well (the maximum number of factors occurring when \( n \) is simply a product of 2s), we can find all the factors in polynomial time.

Problem 7.29

Show that if P = NP, a polynomial time algorithm exists that, given a boolean formula \( \phi \), actually produces a satisfying assignment for \( \phi \) if it is satisfiable.

- If P = NP, then there is a deterministic TM D that solves SAT in polynomial time.
- Consider the following algorithm:
  \( B = “ \text{On input } \phi, \text{ where } \phi \text{ is a boolean formula of variables } x_1, x_2, x_3, \ldots, x_k “ \)
  1. Run D on \( \phi \). If \( \phi \) is not satisfiable, reject. Otherwise
  2. For \( i \) from 1 to \( k \)
3. Replace all the $x_i$s in $\phi$ with 1, and simulate D on that.
4. If D accepts, permanently overwrite $x_i$ with 1, otherwise overwrite $x_i$ with 0.

- Notice that this algorithm is definitely in P, since $k$ (the number of variables), is of course $\leq n$. Thus the “for loop” and D make it $O(k) \cdot O($ time of D $) = \text{polynomial} \times \text{polynomial} = \text{polynomial}$

- Notice also that the algorithm is accurate. It only gets to the “for loop” if it knows that the formula is satisfiable. For each $x_i$, if D rejects, then it’s absolutely certain that it must accept some assignment of a value to $x_i$, and there only are two assignments...

**Problem: NP $\neq$ CoNP implies P $\neq$ NP**

This is easily shown by arguing the contrapositive. If $P = NP$, then CoNP = NP.

- If $P=NP$, then since $P$ is closed under complement, so is $NP$. That is, if we have a machine that decides a language in $NP$, there exists a $P$ machine that decides the same language. We can decide that complement of that language in polynomial time as well by switching the accept and reject states of the $P$ machine (Note: switching the accept and reject states of an $NP$ machine does not necessarily give you a machine that decides the complement).