Solutions to HW2

1. To show that $L^+ \in NP$ we will show that there is a TM $M$ and a polynomial $p$:

$$\forall y \in \{0,1\}^*: \ y \in L^+ \iff \exists u \in \{0,1\}^p|y|: M(u,y) = 1.$$

Suppose that $y = y_1y_2...y_m \in \{0,1\}^*$ and each $y_i \in \{0,1\}^*$, $1 \leq i \leq m$. $M$ expects its certificate $u$ to have the form

$$i_1, i_2, ..., i_k, c_1, c_2, ..., c_{k+1}$$

and interprets $i_1, i_2, ..., i_k$ as positions where its input $y$ needs to be split into substrings $y_1, y_1y_2, ..., y_{k+1}$. Each $c_j$, $1 \leq j \leq k+1$, is taken to be the string certifying membership of $y_j$ in $L$. Since $L \in NP$ we know that there is a TM $M_L$ and a polynomial $p_L$ such that $
exists x \in \{0,1\}^*: x \in L \iff \exists c \in \{0,1\}^p|x|: M_L(x,c) = 1$.

Now we describe the poly-time algo for $M$.

**INPUT:** $y = y_1y_2...y_m \in \{0,1\}^*$

**CERTIFICATE:** $u = (i_1, i_2, ..., i_k, c_1, c_2, ..., c_{k+1})$

1. Split $y$ according to the positions $i_1, i_2, ..., i_k$ and obtain substrings

$$y_1 = y_1y_2...y_{i_1}, \quad y_2 = y_{i_1}y_{i_1+1}y_{i_1+2}...y_{i_2}, \quad ..., \quad y_{k+1} = y_{i_{k+1}}y_{i_{k+1}+1}...y_m$$

2. For $j = 1, 2, ..., k+1$ do
   * If $M_L(y_j, c_j) = 0$ then return 0


It is easy to see that $M$ runs in time that is polynomial in $m = |y|$. Also note that size of certificate $u$ is at most $O(m \log m) + m \cdot p_L(m)$. □
2. **HALT is NP-hard.**

   **Proof:** We show this by proving \( \text{SAT} \leq_p \text{HALT} \).

   Consider an algorithm that transforms a given boolean formula \( \varphi \) in CNF into \( \langle M_\varphi, 0 \rangle \), where \( M_\varphi \) is the TM described below:

   **Turing Machine \( M_\varphi \)**
   **Input:** \( w \in \{0, 1\}^* \)

   1. Run through all possible truth assignments to the variables in \( \varphi \) to determine if \( \varphi \) is satisfiable.

   2. If \( \varphi \) is satisfiable, then halt; else go into an infinite loop.

   Note that on any input \( w \in \{0, 1\}^* \), \( M_\varphi \) halts iff \( \varphi \) is satisfiable.

   Also note that the algo. that takes \( \varphi \) and transforms it into \( \langle M_\varphi, 0 \rangle \) takes constant time (i.e., time independent of \( |\varphi| \)).

3. Suppose that \( P = NP \). Recall that INDSET is the following decision problem:

   **INDSET**
   **Input:** Graph \( G = (V, E) \), positive integer \( k \)
   **Question:** Does \( G \) have an independent set of size \( \geq k \)?

   Since INDSET \( \in NP \) & since \( P = NP \), INDSET has a
polynomial-time algorithm. Let \( A_{IS} \) be an algorithm that solves INDSET in polynomial time for some polynomial \( P(n) \). Using \( A_{IS} \), we can design the following algorithm for the Maximum Independent Set problem.

**Maximum Independent Set**

**Input:** \( G = (V, E) \)

**Output:** Independent set \( I \subseteq V \) of maximum size.

1. Find largest \( k \) such that \( G \) has an independent set of size \( k \).

   (This can be done by calling \( A_{IS}(G, k) \) for \( k = |V|, \ |
   V| - 1, |V| - 2, \ldots \). If \( n = |V| \), the running time of
   Step 1 is at most \( n \cdot P(n) \).)

2. for each \( v \in V \) do
   - Let \( H \) be the graph obtained by deleting \( v \)
     and its neighbors \( N(v) \).
   - If \( A_{IS}(H, k-1) = \text{"yes"} \) then add \( v \) to
     the solution and return \( H \).

After Step 2 we know that \( G \) has an independent set
of size \( k \). Hence for some \( v \in V \), \( H = G \setminus \{v\} \cup N(v) \)
contains an independent set of size \( k-1 \). We use
Step 2 above to discover such a \( v \in V \). Note that
Step 2 takes at most \( n \cdot P(n) \) time.

We can then recurse \( \Box \) on \( H \) to find the remaining
\( k-1 \) elements of the size-\( k \) independent set in \( G \). The total running time of this is \( k \cdot n \cdot P(n) \leq n^2 \cdot P(n) \).
Let us use the notation $\leq_c$ to denote poly-time Cook reducibility.

CLAIM: If $L \leq_c L'$ & $L' \leq_c L''$ then $L \leq_c L''$.

Proof: Suppose that $M$ is a poly-time TM that decides $L$, given an oracle for deciding $L'$. Let $M$ run in time $p(n)$. Suppose that $M'$ is a poly-time TM that decides $L'$, given an oracle for deciding $L''$. Let $M'$ run in time $q(n)$.

We now modify $M$ so that it can decide $L$ in poly-time, given an oracle for $L''$. This will show that $L \leq_c L''$.

The modification to $M$ is as follows:

1. When $M$ writes on its magical extra tape and goes into its special "invocation" state, instead of calling an oracle, $M$ simply initiates the execution of $M'$.

2. $M$ uses $M'$'s magical extra tape as its input tape & uses a completely new set of tapes (separate from $M$'s tapes) along with its own magical tape needed for invoking an oracle to decide $L''$.

The number of tapes used by the modified $M$ equals (roughly) the number of tapes it was using initially plus the number of tapes being used by $M'$. Also, the running time of $M$ is the polynomial $P(q(n))$. Finally, note that $M$ is a machine that decides $L$ given an oracle for $L''$.

CLAIM: 3SAT $\leq_c$ TAUTOLOGY

Proof: Given an instance $\phi$ of 3SAT, we can construct $\neg \phi$ and write it onto its magical extra tape & go into the "invocation" state for deciding TAUTOLOGY. If the oracle returns 1 then $M$ outputs 0; if the oracle returns 0, then $M$ outputs 1. It is easy to see that $M$ runs in poly-time. \qed
5. Let \( \text{coNP}_2 \) denote the class of languages defined in Def. 2.19. Let \( \text{coNP}_2 \) denote the class of languages defined in Def. 2.20.

**Claim:** \( L \in \text{coNP}_1 \Rightarrow L \in \text{coNP}_2 \).

**Proof:** If \( L \in \text{coNP}_2 \), then \( L \in \text{NP} \). By Def. 2.1, there exists a polynomial \( p \) and a poly-time TM \( M \) such that \( \forall x \in \Sigma^*, x \in L \iff \exists u \in \Sigma^* \text{ s.t. } M(x,u) = 1 \).

This is equivalent to: \( \forall x \in \Sigma^*, x \in L \iff \forall u \in \Sigma^* \text{ s.t. } M(x,u) = 0 \). Construct a TM \( \overline{M} \) by modifying \( M \) to simply change its output from 1 to 0 and vice versa. Then, for a polynomial \( p \) and a poly-time TM \( \overline{M} \), \( \forall x \in \Sigma^*, x \in L \iff \forall u \in \Sigma^* \text{ s.t. } \overline{M}(x,u) = 1 \). Hence, by Def. 2.20, \( L \in \text{coNP}_2 \).

**Claim:** \( L \in \text{coNP}_2 \Rightarrow L \in \text{coNP}_1 \).

**Proof:** If \( L \in \text{coNP}_2 \), then by Def. 2.20, there exists a polynomial \( p \) and a poly-time TM \( M \):

\[
\forall x \in \Sigma^*, x \in L \iff \forall u \in \Sigma^* \text{ s.t. } M(x,u) = 1.
\]

Now construct a TM \( \overline{M} \) from \( M \) that outputs 1 when \( M \) outputs 0 and outputs 0 when \( M \) outputs 1. Then:

\[
\forall x \in \Sigma^*, x \in L \iff \forall u \in \Sigma^* \text{ s.t. } \overline{M}(x,u) = 0 \iff \forall x \in \Sigma^*, x \in L \iff \forall u \in \Sigma^* \text{ s.t. } \overline{M}(x,u) = 1.
\]

By Def. 2.1, \( L \in \text{NP} \) and \( \vdash L \in \text{coNP}_1 \).

The two claims above imply that \( \text{coNP}_1 = \text{coNP}_2 \).

6. Suppose there is a language \( L \subseteq 1^* \) that is \( \text{NP} \)-complete. Then \( \text{SAT} \leq_p L \). Suppose that \( A \) is an algorithm promised by \( \text{SAT} \leq_p L \). In other words, \( \forall x \in \Sigma^*, x \in \Sigma \text{ s.t. } A(x) \in L \). Furthermore, suppose that \( A \) runs in time \( n^c \). Hence, if \( |x| = n \),
then $|A(x)| \leq n^c$.

Now consider the following inefficient algorithm for SAT. Suppose the input is $\Phi(x_1, x_2, \ldots, x_m)$ and suppose that $|L(\Phi)| = n$. To determine if $\Phi$ is satisfiable, we create two new, smaller instances of SAT, namely $\Phi^T$ and $\Phi^F$ where $\Phi^T$ ($\Phi^F$) is obtained from $\Phi$ by setting $x_1 = \text{TRUE}$ ($x_1 = \text{FALSE}$). Then we process $\Phi^T$ and $\Phi^F$ in a similar manner (by setting $x_2 = \text{TRUE}$ & $x_2 = \text{FALSE}$) to get a collection of $4^n$ SAT instances, each of size $\leq n$ and each containing $m-2$ variables. Continuing in this manner would yield an exponential algorithm.

However, we can use $A$ to speed up this algorithm. Define the relation $\sim$ on SAT instances as follows: $\Phi \sim \Phi'$ iff $A(\Phi) = A(\Phi')$. It is easy to see that $\sim$ is an equivalence relation. Now note that any instance $\Phi'$ of SAT of size $\leq n$ is mapped by $A$ to a unary string of length $\leq n^c$. Since there are at most $n^c$ distinct unary strings of length $\leq n^c$, it means that SAT instances of size $\leq n$ can be partitioned into $\leq n^c$ equivalence classes by $\sim$. Furthermore, note that all SAT instances in the same equivalence class are either all satisfiable or all unsatisfiable. Thus, we can use $A$ to prune the collection of SAT instances so that it never exceeds size $n^c$. This yields a poly-time algo. for SAT, implying that $P = NP$. \hfill \Box
7. **Claim:** If $P = NP$ then $\Sigma_2^{\text{SAT}} \in P$.

**Proof:** If $P = NP$, then $\text{coNP} = P$.

Now fix $x \in \{0,1\}^*$ and let $\phi_x(y)$ denote the CNF formula $\psi(x,y)$. Then determining if $\phi_x(y) = 1$ for all $y \in \{0,1\}^m$ is TAUTOLOGY $\in \text{coNP} = P$. Hence this problem has a polynomial time solution. Let $T$ denote the algorithm that solves this problem in polynomial time.

To solve $\Sigma_2^{\text{SAT}}$ we design a non-deterministic TM $T$ that first "guesses" the bits $x_1, x_2, \ldots, x_n$ and then calls $T$ to solve the rest of the problem. This establishes that $\Sigma_2^{\text{SAT}} \in \text{NP}$. But since $NP = P$, $\Sigma_2^{\text{SAT}} \in P$. \qed