1. If A and B are NP-Complete problems, then show that $A \preceq_{\mathsf{P}} B$ and $B \preceq_{\mathsf{P}} A$.

2. Recall that, for a graph G, its complement graph $\overline{G}$ is a graph such that $V(\overline{G}) = V(G)$ and $E(\overline{G}) = \{(u, v) : (u, v) \notin E(G)\}$; that is, its the graph formed by picking exactly those edges that are not present in G.
   (a) Let S be an independent set of a graph $G(V, E)$. Then, show that S is a clique in $\overline{G}$.
   (b) Suppose that a set $S \subseteq V(G)$ is *not* an independent set in $G(V, E)$. Then, show that S is not a clique in $\overline{G}$.
   (c) Suppose that a set $C \subseteq V(G)$ is a clique in $\overline{G}$. Show that C is an independent set in $\overline{G}$.
   (d) Show that INDEPENDENTSET $\preceq_{\mathsf{P}}$ CLIQUE.
   (e) Show that CLIQUE $\preceq_{\mathsf{P}}$ INDEPENDENTSET.

3. Show that $\leq$3SAT $\preceq_{\mathsf{P}}$ 3SAT. Note that $\leq$3SAT is a variant of SAT in which every clause contains *at most three* literals.

   Hint: Decide how you will handle clauses containing only 1 or 2 literals.

4. For showing that SAT $\preceq_{\mathsf{P}}$ 3SAT, we described what the algorithm would do in each step (or, equivalently, during each intermediate reduction, if using transitivity). It should also be argued that the number of steps (or, equivalently, the number of intermediate reductions) is polynomial, as otherwise the reduction algorithm would not be in P.

   Show that this is indeed the case with our algorithm.

   Hint: While the exact number of steps required can be computed fairly easily, its easier (and sufficient) to define a polynomial *measure* function that is always an upper bound on the number of steps and whose value is guaranteed to decrement in each step.

5. Justify why we *cannot* extend the proof of SAT $\preceq_{\mathsf{P}}$ 3SAT to show that SAT $\preceq_{\mathsf{P}}$ 2SAT. Note that 2SAT is a variant of SAT in which every clause contains *exactly two* literals.

6. Show that each of the problems listed below are NP-Complete. To begin with, you may assume that SAT $\in$ NP-Complete. Recall that, to show that a problem $A \in$ NP-Complete, it is sufficient to show that (1) $A \in$ NP and (2) that $B \preceq_{\mathsf{P}} A$, for some $B \in$ NP-Complete. Furthermore, to show that $B \preceq_{\mathsf{P}} A$, its necessary to show that an instance I_B of B is an YES-instance if and only if the reduced instance I_A is an YES-instance of A.
   1. $\leq$3SAT
   2. 3SAT
   3. INDEPENDENTSET
   4. VERTEXCOVER
   5. CLIQUE
   6. SETCOVER
   7. HITTINGSET
   8. ILP (The Integer Linear Programming problem)
   9. CUBICVC (This is a variant of VERTEXCOVER, in which the input graph is a *cubic graph*; that is, a graph in which every vertex has degree exactly 3.)
   10. HAMPATH
   11. HAMCYCLE
   12. TSP
   13. THREECOLORING (This is a variant of GRAPHCOLORING in which we determine if the graph can be colored with 3 colors.)
   14. GRAPHCOLORING

7. Given a graph G, an integer k and a vertex $v \in V(G)$, prove the following statement.

$$(G, k) \textit{ is an YES-instance of } \textsc{VertexCover} \textit{ if and only if}$$
$$(G \setminus v, k-1) \textit{ or } (G \setminus N(v), k - |N(v)|), \textit{ is an YES-instance of } \textsc{VertexCover}.$$

8. Assuming that you are given algorithms to solve each of the problems in Question 6, use them to design algorithms for their corresponding OPT and Search versions (if they exist).

9. For each of the following problems, show how to reduce them to the ILP problem. Note that such reductions are referred to as the ILP *formulations* of the corresponding problem. While most problems have a natural ILP formulation, they can have other formulations as well depending on how you characterize them.
   1. IndependentSet
   2. CubicVC
   3. Clique
   4. SetCover
   5. HittingSet
   6. MaxSAT
      Hint: Think about what you are trying to maximize here and choose your variables accordingly. Next, think about what the constraints on the underlying problem are and how to express them mathematically.
   7. SAT
      Hint: SAT is not a minimization/maximization problem; as such, to formulate an ILP instance, its useful to determine what the problem is that SAT minimizes/maximizes.
   8. HamCycle
   9. TSP
   10. GraphColoring

10. co-NP is the class of decision problems that have polynomial time verifiers for NO instances. Show that
    (a) UNSATISFIABILITY $\in$ co-NP.
    (b) $\overline{\textsc{VertexCover}} \in$ co-NP.
    (c) $\overline{\text{TSP}} \in$ co-NP.
    (d) MST $\in$ NP and MST $\in$ co-NP.
    (e) P $\subseteq$ co-NP.
    (f) $A \in$ co-NP if and only if $\overline{A} \in$ NP.

11. Let $(\mathcal{F})$ be a NO-instance of SAT. In class, we informally argued that a verifier to check whether $(\mathcal{F})$ is a NO-instance cannot run in polynomial time, as a certificate certifying this fact would possibly have to list all possible truth assignments.
    (a) Think about why this type of argument is informal.
    (b) Can you think of some other problems for which this type of argument would not only be informal but also incorrect?

    Lets try to resolve these issues!
    (c) Suppose that an NP-Complete problem had a polynomial time verifier for NO instances. Then, show that NP $\subseteq$ co-NP.
    (d) Prove: If NP $\subseteq$ co-NP, then co-NP $\subseteq$ NP. Note that this further implies NP $=$ co-NP.
    (e) As with the P versus NP problem, its generally *believed* that NP $\neq$ co-NP. Using this assumption, argue that SAT is *unlikely* to have a polynomial time verifier for NO instances.