1. Show that if $L \in P$ and $L' \leq_m L$, then $L' \in P$.

A. Since $L' \leq_m L$, there is a totally computable function f such that $\forall x, x \in L'$ iff $f(x) \in L$. Build a DTM M to accept L' as follows:

> input x
> compute $f(x)$
> if $f(x) \in L$ then M accepts x
> otherwise M rejects x

By definition, $f(x)$ can be computed in deterministic polynomial time. Since $L \in P$, $f(x) \in L$ can also be computed in deterministic polynomial time, given the fact that the length of $f(x)$ is bounded by a polynomial (this polynomial is the polynomial time for computing f). Therefore DTM M runs in polynomial time and $L' \in P$.

2. Show that if $L' \leq_m L$, then $\bar{L}' \leq_m \bar{L}$.

A. Since $L' \leq_m L$, there is a totally computable function f such that $\forall x, x \in L'$ iff $f(x) \in L$. From this, we can determine the following:

$$\forall x' \in \bar{L}' \rightarrow x' \notin L' \rightarrow f(x') \notin L \rightarrow f(x') \in \bar{L}$$

$$\forall f(x') \in \bar{L} \rightarrow f(x') \notin L \rightarrow x' \notin L' \rightarrow x' \in \bar{L}'$$

So $x' \in \bar{L}'$ iff $f(x') \in \bar{L}$. Therefore $\bar{L}' \leq_m \bar{L}$.

3. Show that there is a recursive language that is not in NP.

A. Remember that $NP \subseteq \cup_k DTIME(2^{n^k})$. Now we construct a language L as follows: $L = \{0^i : 0^i \text{ is not accepted by } M_i \text{ in } 2^{n^i} steps\}$. Using diagonization (see your notes8), you can show that $L \notin DTIME(2^{n^k})$. Hence, $L \notin NP$. Clearly, L is recursive. Done.

4. Show that there are two languages L_1 and L_2 such that $L_1 \subseteq L_2$, $L_1 \leq_m L_2$, L_1 is NP-complete, and L_2 is NOT in NP.

A. L is defined as in the previous problem. Let $L_1 = SAT$ and $L_2 = L_1 \cup L_2$. Notice that here L_2 and L_1 are on different alphabet. Clearly, L_1 is NP-complete, and L_2 is NOT in NP.

5. Problem Q is defined as follows:

   - Given: a finite collection C of subsets of a finite set S, a number k.
   - Question: is there a set $S' \subseteq S$ such that $|S'| \leq k$ and S' contains at least one element from each subset in C?

   Show that Problem Q is NP-complete.

A. To show Problem Q is NP-complete, show that Problem Q is in NP and Problem Q is NP-hard.

   - $Q \in$ NP: Guess S' and check that $S' \subseteq S$, $|S'| \leq k$, and S' contains at least one element from each subset in C. Guessing S' takes $O(n)$ time and checking S' takes $O(n^2)$ time, so $Q \in$ NP.
   - VC $\leq_m Q$: The Vertex Cover problem is NP-complete, so if VC can be transformed into Q in polynomial time, then Q is NP-hard. Transform the Vertex Cover problem to Q as follows:
     1. Start with a graph $G = (V, E)$ and value k.
     2. Set $S = V$.

3. For each $(v, v') \in E$, add the subset $\{v, v'\}$ to C.

From S and C, form the set S'. The set S' is a vertex cover for G, where $|S'| \leq k$. In the original VC problem, the size of the VC is k. If $|S'| < k$, then $\exists S''$, where $S' \subset S'' \subseteq S$, and $|S''| = k$. The set S'' is formed by setting $S'' = S'$ and adding elements of S until $|S''| = k$. S'' is still a VC for G. The transformation of VC to Q can be performed in deterministic polynomial time, so VC $\leq_m Q$. VC is NP-complete, so Q is NP-hard.

6. Define co-NP $= \{L : \bar{L} \in \text{NP}\}$. Show that if there is L such that L is both NP-complete and co-NP-complete, then NP = co-NP.

A. If $L \in$ NP-complete, then $L \in$ NP, and if $L \in$ co-NP-complete, then $L \in$ co-NP.

- NP $\subseteq$ co-NP: $\forall L' \in$ NP, $L' \leq_m L$, since $L \in$ NP-complete. Using a construction similar to problem 1, we can show that since $L \in$ co-NP, $L' \in$ co-NP.
- co-NP $\subseteq$ NP: $\forall L' \in$ co-NP, $L' \leq_m L$. Since $L \in$ NP, $L' \in$ NP.

NP $\subseteq$ co-NP and co-NP $\subseteq$ NP, so NP = co-NP.

7. Problem R is defined as follows:

- Given: a graph $G = (V, E)$, and a number k.
- Question: is there a spanning tree for G where no vertex has degree more than k.

Show that Problem R is NP-complete.

A. To show Problem R is NP-complete, show that Problem R is NP and Problem R is NP-hard.

- $R \in$ NP: Guess spanning tree T and check that T is a spanning tree. To check that T is a spanning tree, check that T contains all vertices in G, that the degree of each vertex is no more than k, and that every edge in T is in G. Guessing T takes $O(n)$ time and checking T takes $O(n^2)$ time, so $R \in$ NP.
- HP $\leq_m R$: The Hamiltonian Path problem is concerned with finding a path through a graph that visits each vertex once. Transform the HP problem to Q as follows:
  1. Start with a graph $G = (V, E)$.
  2. Set $k = 2$.

  From G, find a spanning tree T where the degree of each vertex in T is $\leq k$. The spanning tree T is a HP for G. The transformation of HP to R can be performed in deterministic polynomial time, so HP $\leq_m R$. HP is NP-complete, so R is NP-hard.

8. Show that 2-SAT is in P.

A. The 2-SAT problem is defined as follows:

- Given: A boolean formula F in Conjunctive Normal Form (CNF), where $F = \bigwedge F_i$ and $F_i = (a_{i1} \vee a_{i2})$.
- Question: Is there an assignment of values $a_{ij} = \{0, 1\}$ such that F is satisfied?

The 2-SAT problem is similar to the 3-SAT problem, except each clause can only contain one or two literals. 2-SAT can be solved using the following steps:

1. Eliminate clauses with a single literal. The key observation is that each clause must be true to make the conjunction true. For each clause with a a single non-negated literal, replace all instances of the literal with 1. For each clause with a a single negated literal, replace all instances of the literal with 0. If any literal must be set to both 0 and 1, then the boolean formula is not satisfiable. It takes one pass through the formula to find all disjunctions with one literal, and one pass to replace all instances with 0 or 1, so this step can be done in $O(n)$ time.

2. A clause of the form $(a_i \vee a_j)$ is equivalent to $(\neg a_i \Rightarrow a_j)$ and $(\neg a_j \Rightarrow a_i)$, so the boolean formula can be rewritten as a set of implications. Because implication is transitive, a series of clauses such as $(\neg a_i \vee a_j) \wedge (\neg a_j \vee a_k) \wedge (\neg a_k \vee a_l)$ can be rewritten as $a_i \Rightarrow a_j \Rightarrow a_k \Rightarrow a_l$.

   Given a boolean formula F, a directed graph of these implications can be built as follows:

   $G = (V, E)$

   $V = \{a_1, \neg a_1, a_2, \neg a_2, \cdots, a_n, \neg a_n\}$

   $E = \{(a_i, a_j) : (\neg a_i \vee a_j) \in F\} \cup \{(a_i, a_j) : (a_i \vee \neg a_j) \in F\}$, $E \subseteq V \times V$

   This graph can be constructed in $O(n)$ time.

3. If there is a string of implications $a_i \Rightarrow \cdots \Rightarrow \neg a_i$ and another string of implications $\neg a_i \Rightarrow \cdots \Rightarrow a_i$, then the boolean formula is unsatisfiable. This is equivalent to having a path from $\neg a_i$ to a_i and a path from a_i to $\neg a_i$ in G. In general, we can assume that $a_i = 1$ and $\neg a_i = 0$. In this case, there is an edge (a_j, a_k) on the path from a_i to $\neg a_i$ such that $a_j = 1$ and $a_k = 0$. Since there is an edge (a_j, a_k), then either the clause $(a_j, \neg a_k)$ or the clause $(\neg a_j, a_k)$ is in F. In either case, the clause is 0, so F is unsatisfiable.

4. If there is are no strings of implications $a_i \Rightarrow \cdots \Rightarrow \neg a_i$ and $\neg a_i \Rightarrow \cdots \Rightarrow a_i$, then the boolean formula is satisfiable. In this case, G has no a_i such that there is a path from $\neg a_i$ to a_i and a path from a_i to $\neg a_i$. This means that there is no edge (a_j, a_k) such that $a_j = 1$ and $a_k = 0$ or $a_j = 0$ and $a_k = 1$. So either $a_j = a_k = 0$ or $a_j = a_k = 1$. In both cases, $(a_j \vee \neg a_k) = 1$ and $(\neg a_j \vee a_k) = 1$, so F is satisfiable.

   To actually assign the values, do the following: For each vertex a_i where there is no path from a_i to $\neg a_i$ in G, pick an edge (a_i, a_j) and assign $a_i = a_j = 1$ and $\neg a_j = 0$. From this, all additional connected vertices in the graph can be assigned a value. All remaining vertices can be assigned either 0 or 1. They are not connected to any other vertex, so the final truth value of F does not depend on their value.

5. Checking for strings of implications can be done in polynomial time by computing the transitive closure of G using Warshall's algorithm. This algorithm runs on $O(n^3)$. Running this algorithm on all of the n vertices results in an $O(n^4)$ algorithm to check for strings of implications.

Computing 2-SAT can be done in $O(n^4)$ time, so 2-SAT is in P.