Read the instructions carefully.

More instructions specific to this exam:

1. Answers must be accompanied by rigorous proofs. Unlike the previous exams, vague ideas might not get marks.
2. While describing the behavior of a Turing machine, it is sufficient to give a high level description, that is, a COL106 or COL351 style algorithm.
3. We have assumed or proved the NP-hardness of only the following problems in class and in Homework 5: `CircuitSAT`, `3SAT`, `Strict3SAT`, `IndSet`, `Clique`, `SubgraphIsomorphism`, `VertexCover`, `SetCover`, `HittingSet`, `DominatingSet`, and the crazy COL352 instructor problem. Only these problems are to be assumed to be NP-hard for this exam.

1. **[1 mark each]** State whether each of the following statements is true or false. Substantiate each answer with a short (ideally, at most 3 sentences of) justification. (Answers with incorrect and insufficient justifications get 0 marks.)
   1. If L_1, L_2, L_3 are languages on the same alphabet such that each of $\sim_{L_1}, \sim_{L_2}, \sim_{L_3}$ has 7 equivalence classes, then $\sim_{(L_1 \cup L_2 \cup L_3)}$ has at most 343 equivalence classes. (Recall: $x \sim_L y$ if for all z, either both xz and yz are in L or both are not in L.)
   2. There exists a language that can be recognized by some NPDA with 4 stacks but cannot be recognized by any NPDA with 3 stacks.
   3. A language is Turing-recognisable if and only if it is mapping-reducible to A_{TM}, the membership language of Turing machines.
   4. If L_1 and L_2 are languages on the same alphabet Σ, L_1 is in P, and L_2 is neither empty nor Σ^*, then L_1 is Karp-reducible to L_2 necessarily.

2. **[6 marks]** Let Σ be a finite alphabet and let $L_0, L_1 \subseteq \Sigma^*$ be languages such that $L_0 \cap L_1 = \emptyset$. Think of L_0 and L_1 as the sets of "bad" and "good" strings respectively. We wish to design an automaton which is guaranteed to reject every string in L_0 and accept every string in L_1. Our automaton is allowed to behave arbitrarily on strings that are neither in L_0 nor in L_1. In other words, our automaton is required to recognize some language $L \subseteq \Sigma^*$ such that $L \cap L_0 = \emptyset$ and $L \supseteq L_1$.

   For concreteness, let us say that we are looking for a DFA over the alphabet $\Sigma = \{0, 1\}$. Also, let $L_0 = \{x \in \{0,1\}^* \mid \text{the number of 0's and 1's in } x \text{ are unequal}\}$ and $L_1 = \{0^n 1^n \mid n \in \mathbb{N} \cup \{0\}\}$. Observe that $L_0 \cap L_1 = \emptyset$. Does there exist such a DFA which accepts every string in L_1 and rejects every string in L_0? Prove your answer.

3. Consider a graph $G = (V, E)$. Recall that if G is connected, then G contains a spanning tree, i.e., a tree that connects all the vertices to one another, and that every such spanning tree has exactly $|V| - 1$ edges. Now, instead of connecting all vertices, suppose our task is to build a tree and connect a given subset $S \subseteq V$ of vertices to one another, possibly through vertices not in S. Obviously, we will need at least $|S| - 1$ edges. However, depending on the graph, we might need many more. An extreme case is where if G is a path and S contains only the two endpoints, in which case our tree must include all the edges. Thus, it makes sense to find a tree in G which connects vertices in S to one another and which has as few edges as possible.

   The decision version of this problem, which we will call `SetSpanningTree`, is as follows. The input is a graph $G = (V, E)$, a set $S \subseteq V$, and an integer k. We want to accept such an input if and only if G contains a tree which connects vertices in S to one another, and which has at most k edges. Our goal here is to prove that `SetSpanningTree` is NP-complete.

1. **[2 marks]** Prove that `SetSpanningTree` is in NP by giving a non-deterministic polynomial-time algorithm.
2. **[8 marks]** Prove that `SetSpanningTree` is NP-hard by giving an appropriate Karp-reduction, and proving that it is correct and it runs in polynomial time.