Worth: 8% **Due:** Thursday December 3

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing.

1. Show that A_{TM} is NP-hard. Hint: use the original definition of NP-hardness, not a reduction from other known NP-hard problems). What is the running time of your reduction function? (be more specific than just "polynomial").

2. A set of vertices S in a graph G is called "almost-independent-set" if by removing at most one edge to G, S becomes an independent set. Define

$$\text{ALMOST-IS} = \{\langle G, k\rangle \mid G \text{ is a graph on } n \text{ vertices that has an almost-independent set of size } k\}.$$

   Show that ALMOST-IS is NP-complete. Hint: reduce from INDEPENDENT-SET. Your reduction will add two vertices (you will have to decide which edges to add!) to the original graph.

3. Recall that coNP is the class of complements of languages in NP. By analogy with the definition of NP completeness, we call a language coNPcomplete if

   - B is in coNP, and
   - B is coNP-hard, i.e., for all $A \in$coNP, $A \leqslant_p B$.

   (a) Show that for any language A, A is coNP complete if and only if $\overline{A}$ is NP complete.

   (b) Show that if P = NP then P = coNP.

   (c) Show that if there is any language that is both NP complete and coNP complete, then NP = coNP.

   (d) Consider the following "Formula Equivalence" language:

   FE = $\{\langle F_1, F_2\rangle : F_1, F_2$ are propositional formulae that are logically equivalent, *i.e.*, *every* assignment of values to the variables of F_1 and F_2 gives both formulae the same value$\}$

   Write a detailed proof that FE is *coNP*-complete.

4. Show that

$$\text{PARTITION} = \{\langle S\rangle \ : \ S \text{ is a list of positive integers } x_1, \ldots, x_n, \text{ and}$$
$$\text{there exists } T \subseteq \{1, \ldots, n\} \text{ such that } \sum_{i\in T} x_i = \sum_{i\notin T} x_i\}$$

   is self reducible. Namely, if PARTITION is in P then there is a polynomial time algorithm to *find a subset* T such that $\sum_{i\in T} x_i = \sum_{i\notin T} x_i$.

   Show that your algorithm is correct and that it runs in the correct time bound.