1. A(n) **reduction** is a translation of an instance of one problem $P$ into an instance of another problem $P'$ such that the solution of $P'$ implies a solution to $P$.

2. True or False. The class of languages accepted by nondeterministic Turing machines in polynomial time is the same as the class of languages accepted by deterministic Turing machines in polynomial time.

   Probably False. A deterministic Turing machine may simulate a nondeterministic one by enumerating all of the possible ID's of the NTM in a breadth-first manner until one with a final state is found. This creates exponential time. There are many nondeterministic polynomial time languages which we do not believe but cannot prove can be recognized in deterministic polynomial time.

3. True or False. It is known that $P \subset NP$ but it can never be formally shown that the containment is proper or if $P = NP$.

   False. It probably cannot be shown, but if any NP-complete problem were shown to be in $P$, then $P=NP$.

4. List two (2) properties that are undecidable for recursive languages.

   - emptiness
   - finiteness
   - regularity
   - context-freeness

5. For each language below, indicate whether the language is: 1) in $P$, 2) in $NP$ but probably not in $P$, or 3) probably not in $NP$. Give a brief informal justification for your answer. Note that all languages are encoded over the alphabet $\{0, 1\}$.

   (a) $L = \{w \mid w$ is a Boolean formula which is *not* satisfiable\}.

      For example, $(x)(-x), (x+y)(-x+y)(x+y)(-x-y), \text{ etc.}$

      $L$ is probably not in $NP$. The obvious algorithm is to try all possible assignments to variables to verify that $w$ is not satisfiable, but trying all possibilities cannot be done in polynomial time. Also note that this is essentially the complement of SAT, which is NP-complete and NP-completeness is probably not closed under complementation.

   (b) $L = \{<x, y> \mid x$ is a list of integers and $y$ is the same list of integers sorted in ascending sequence\}.

      For example, $<(3, 1, 2), (1, 2, 3)>, <(7, 4, 2, 13), (2, 4, 7, 13)>, \text{ etc.}$

      $L$ is in $P$. Polynomial time sorting algorithms such as quick sort may be used to sort $x$ and then verify that $y$ is the same as that sorted list.

   (c) $L = \{<P, n> \mid P$ is an instance of Post’s Correspondence Problem which has a solution of maximum size $n\}$.

      For example, $<P_1, 0>$ is not in $L$ because any solution to $P_1$ would be at least of size 1, and $<P_2, 3>$ would not be in $L$ if $P_2$ has a solution of size larger than $n$ or has no solution at all, and would be in $L$ otherwise.

      $L$ is in $NP$ but probably not in $P$. It is in $NP$ because we may nondeterministically guess the solution and then verify it in polynomial time. It is probably not in $P$ because to solve this problem deterministically requires trying all possible solutions of size $n$ or less, of which there are an exponential number.
6. For each language below, indicate whether the language is: 1) recursive, 2) recursively enumerable but not recursive, or 3) not recursively enumerable. Give a brief informal justification for your answer. Note that all languages are encoded over the alphabet \{0, 1\}.

(a) \( L = \{x\#y^R\#z \mid x, y \text{ and } z \text{ are binary numbers such that } x + y = z\} \). Examples of strings in \( L \) include 1001\#011\#1111 and 111\#1000\#1000.

\( L \) is recursive. A Turing machine may copy \( x \) to a second tape and then add \( y \), moving right on tape 1 and left on tape 2, writing the sum on tape 2. Then \( z \) may be compared to tape 2.

(b) \( L = \{<M> \mid M \text{ is a Turing machine which accepts at least 4115 words}\} \).

\( L \) is recursively enumerable but not recursive. It is RE since a nondeterministic Turing machine may guess the 4115 words that \( M \) accepts. It is not recursive since \( M \) may not accept 4115 words and may not halt on words it does not accept. In that case, the Turing machine could not reject \( M \).

(c) \( L = \{<M> \mid M \text{ is a Turing machine which accepts no string of length 3}\} \).

\( L \) is not recursively enumerable. If it were, we could try \( M \) on all strings of length 3 and if all are rejected, then \( M \) should be accepted. However, if even one of these strings was rejected by \( M \) not halting, we would not know and hence would not be able to accept \( M \).