Question 1. State the definition of NP-completeness (by which, as is standard, we mean with respect to $\leq_p^m$ reductions).

Answer:
A set $L$ is NP-complete if both of the following hold:

- $L \in NP$
- For all $L' \in NP$, $L' \leq^p_m L$

Question 2. One sometimes hears it said that “If $P$ equals $NP$, then all $NP$ sets are NP-complete.” I’m not asking you to discuss the logical truth or falseness of that statement (as there is a subtle issue regarding that, namely, if $P \neq NP$, then the statement is logically true (that is, in that case it becomes a vacuous truth); however, I’m more interested in the fact that if the hypothesis of that statement happens to be true, then the statement itself will overall be false. So, your task on this problem is to do the work on that. More specifically, your task is to do this: Give a specific finite set $A$ such that $A$ is an NP set that is not NP-complete (and prove both those facts: that it is in NP and that it is not NP-complete).

Answer:
$\emptyset$ is such a set (and, at least with the “finite” twist above, is the only such set you can prove this about today without winning a Turing Award; if the problem had been stated without “finite,” $\Sigma^*$ would also give you a path to resolving the issue without winning a Turing Award along the way).

This set is trivially in NP. We can decide it with an NTM by just rejecting every input.

No NP-complete set can reduce to this set. If a set $A$ reduces to $\emptyset$, there exists a function $\sigma$ such that $x \in A \iff \sigma(x) \in \emptyset$. However, since there are no elements in $\emptyset$, there is nothing that a function $\sigma$ can map to from elements that are in $A$. Therefore the only set that can many-one reduce to $\emptyset$ is $\emptyset$ itself, so it cannot be NP-complete.

Question 3. Let $L_{log} = \{F \mid F$ is a satisfiable formula in 3CNF form and the number of times a “negation” appears with a symbol in $F$ is at most $\log_2$ of the number of bits in $F\}$. Prove that $L_{log} \in P$.
Answer:
We can decide this language in polynomial time as follows. First, check if the input $F$ is in CNF format with the appropriately bounded number of negations and reject if it is not. Next, assign all variables which are never appear negated in $F$ to true. For every other variable, iterate through all possible assignments to them and check if each assignment causes the formula to evaluate to true. If an assignment does so, accept. If no assignment evaluates to true, reject.

This algorithm will run in polynomial time. Since there are a logarithmically bounded number of negations, there are a similarly bounded number of variables which are ever negated, and so we only have to check $2^{\log |F|} = |F|$ assignments, which is clearly polynomial. Also, we can check each assignment in polynomial time, so the entire process will take polynomial time.