1. Given as input integers $a, b, p$ (in binary representation), we would like to compute $a^b \mod p$.

One way to do it, is to multiply $a$ by itself (modulo $p$) $b$ times.

A second way to do it is as follows. Initialize $v = 1$. Go through $b$ bit by bit, starting from the most significant bit: if the bit is 0, let $v = v^2 \mod p$; else (if bit is 1), let $v = av^2 \mod p$. Output $v$.

Both these are (very high level descriptions of) correct algorithms (the first one works by the definition of exponentiation function, and the second one can be proved to work by noticing that given an integer $b$ in binary representation, the string $b0$ represents the number $2^b$, and the string $b1$ represents the number $2^b + 1$).

For simplicity, for the rest of the problem we assume that $a, b$, and all numbers mod $p$ are represented with the same number of bits as $p$ is. (Note that a number modulo $p$ is always an integer between 0 and $p - 1$, so it will never require more bits than are required to represent $p$; if it requires fewer bits, we can just pad it with 0’s on the left).

(a) Assume that computing modular multiplication (given $x, y, p$ output $xy \mod p$) takes time $O(f(n))$ for some $f$. Analyze the running time of each of these algorithms (up to constants, namely using big-oh notation).

(b) In fact, modular multiplication can be done in polynomial time (namely you can use $f(n) = n^k$ for some $k$). Use that to determine for each of the above algorithms whether it is polynomial time or not (justify your answer).

2. Given two languages $L_1, L_2$, we define $L_1 \odot L_2 = \{xy : |x| = |y|, x \in L_1, y \in L_2\}$.

Prove that NP is closed under $\odot$.

3. Given a boolean formula $\Phi = C_1 \land \ldots \land C_m$ in CNF format, we say that $\Phi$ is almost satisfiable if there exists an assignment that satisfies at least $m - 1$ of the clauses (namely, there exists an assignment where at most one of the clauses evaluates to false). Define $\text{ALMOSTSAT} = \{\Phi : \Phi$ is a boolean formula in CNF format that is almost satisfiable$\}$

Prove that $\text{ALMOSTSAT}$ is NP-complete.

4. Let $L_{NPC}$ be any NP-complete language (e.g., SAT), and $L_{UD}$ be any undecidable language (e.g., $A_{TM}$). Define the language

$\text{HARD} = \{\langle x, y \rangle : x \in L_{NPC}, y \in L_{UD}\}$.

Prove that $\text{HARD}$ is NP-hard but not NP-complete.
An aside about NP-hard but not NP-complete languages: Another example of an NP-hard but not NP-complete problem is $A_{TM}$ (can you prove it? not for submission). There are also decidable problems that are NP-hard but not NP-complete, and natural NP-hard problems that are conjectured not to be in NP (hence not NP-complete). In particular, PSPACE – the set of all languages decidable in polynomial space – contains NP, and is conjectured to be strictly larger (although we don’t even have a proof that it is not equal to P). Therefore, any language that is PSPACE-complete is a conjectured NP-hard but not NP-complete language. This includes, for example, the problem of minimizing NFAs or regular expressions (the decision versions: given a DFA/NFA/RE and a number $k$, is there an equivalent NFA/RE with at most $k$ states/symbols).