Main Steps

To prove a problem X is NP-complete, you need to show that it is both in NP and that it is at least as hard as any other problem in NP. This last step is typically done by showing that $Y \leq_P X$ for some problem Y already known to be NP-Complete. This is described in steps 2 through 5.

Step 1: Show that X is in NP. Argue that there is an efficient verifier for X. In other words, describe a verifier such that for any yes instance of X, there exists a certificate that the verifier will accept, and for any no instance of X, no such certificate exists. The running time of the verifier (and hence the size of the certificate) must be polynomial. Typically, a solution to the given problem is a sufficient certificate. This step is very brief, but necessary.

Step 2: Pick a known NP-complete problem. State what problem Y you are reducing to X; you need to show that $Y \leq_P X$. You may use any problem Y which we have proved in class or on homework assignments to be NP-complete. By the very definition of NP-complete, if X is NP-complete, then you can technically use any other NP-complete problem Y to show this. However, *some problems will be far easier to use than others in your proof.* It is often useful to spend some time thinking about which of the problems you know to be NP-complete would be most natural to use for a given reduction.

Step 3: Construct an algorithm to solve Y given an algorithm to solve X. Show that a instance of Y can be solved using a polynomial number of operations, and a polynomial number of calls to a black box that can solve X. **Note:** It is very easy to get mixed up and instead prove that $X \leq_p Y$. Unfortunately, this is not what you want to show (we already know that Y is NP-complete).

Typically, you will show how to solve Y by constructing a single input to the black box for X, and typically, your algorithm will output the same answer as is given by the black box. However, this is not always the case (in which case, define the correspondence between the output of the black box for X and the desired solution for Y.)

Step 4: Prove the correctness of your algorithm. This requires proving an if and only if statement. It is always trivial to come up with an algorithm that satisfies just one of these two conditions. We want something that satisfies both.

Step 4a: Prove the solution you find is correct. Show that *if* your algorithm returns "yes," *then* the given input is indeed a yes instance of Y.

Step 4b: Prove that you find all solutions . Show that *if* you are given a yes instance of Y, *then* your algorithm returns "yes".

Step 5: Running time and wrap-up. Make sure that your construction to be used with the black box for X is poly-size and poly-time. This should allow you to conclude that since your algorithm runs in polynomial time, $Y \leq_P X$. Since Y is NP-complete, and since we also have shown that X is in NP, X is NP-complete.

An Example: INDEPENDENT SET (IS)

An input to independent set is given by $G = (V, E)$ and an integer $k \geq 0$. The goal is to determine whether there is a subset S of k vertices such that no two vertices of S are adjacent in G.

Lemma 1. *Independent Set is in NP.*

Proof. Given any set S, our verifier will check that S indeed contains at least k vertices (in $O(k)$ time), and that no two vertices of S are adjacent (in $O(km)$ time). For a yes instance (G, k), our certificate is any independent set of size k in G; our verifier will accept such a set. For a no instance (G, k), it is clear that no set of k vertices of G can make our verifier accept. □

Lemma 2. *Independent set is NP-Hard.*

Proof. We show that 3SAT $\leq_P$ IS. Given an instance of 3SAT (a formula $\varphi = C_1 \wedge \cdots \wedge C_m$, where each C_i is the disjunction of 3 terms drawn from variables $x_1, x_2, \ldots, x_n$ and their negations $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$), we construct an instance (G, k) of the Independent Set problem as follows.

Let V be the set of terms in φ; that is, for each term t_j in each clause, create a node t_j. Note that there are multiple nodes with the label x_ℓ or $\overline{x_\ell}$ for a variable x_ℓ that appears in multiple clauses. For the edges E, connect the three (distinct) nodes that correspond to the three terms of each clause; we call these three nodes and three edges a "clause gadget". Finally, for all variables x_ℓ, add an edge between every pair of nodes with one labeled x_ℓ and the other labeled $\overline{x_\ell}$. We now run our black box for Independent Set on (G, m) and return the same result it gives. (This construction is poly-size and poly-time because G has $O(m)$ nodes and $k \in O(m)$.)

To prove this answer is correct, we simply need to show that the original instance φ of 3SAT is a yes instance (i.e. it is satisfiable) if and only if the the algorithm returns "yes".

Claim. *If there is an independent set of size m in G then there is a satisfying assignment for φ.*

Proof. Suppose our algorithm returns "yes," that is, that there is an independent set S of size m in G. We show there is a satisfying assignment A of φ by constructing one: set $A(x_\ell) = T$ if $x_\ell \in S$ and $A(x_\ell) = F$ otherwise. We just need to show that this is a valid truth assignment, and that it is satisfying. Since S is independent, no pair of nodes x_ℓ and $\overline{x_\ell}$ are ever both selected for S because there is an edge between them in G; in this way, each variable is assigned either T or F, but not both, and the assignment A is valid. Moreover, at most one node in each triangle clause gadget is in S. Since there are exactly $m = |S|$ clause gadgets, S must contain exactly one node from each clause gadget; thus, at least one term of each clause evaluates to true. □

Claim. *If there is a satisfying assignment for φ then there is an independent set of size m in G.*

Proof. Suppose φ has a satisfying assignment A. We show that our algorithm returns "yes" by constructing an independent set S of size m in G. Let S be such that for each clause gadget, we select one term node that is satisfied by A to be in S; since A is a satisfying assignment, such a node exists, and $|S| = m$. Furthermore, since we only select a single node for each gadget, independence is not violated within any clause gadget. Finally, independence is not violated between clauses, since the only such edges go between nodes with labels x_ℓ and $\overline{x_\ell}$, and A cannot satisfy both of these, so we never would have selected both. Therefore S is an independent set of size m. □

Theorem 3. *Independent Set is NP-Complete.*

Proof. We showed that $IS \in NP$ and $3SAT \leq_P IS$; therefore IS is NP-Complete. □