1. For any integer $k$, the problem $k$-SAT is defined as follows:

- **INPUT:** A boolean formula $\Phi$ in conjunctive normal form, with exactly $k$ distinct literals in each clause.
- **OUTPUT:** True if $\Phi$ has a satisfying assignment, and False otherwise.

(a) Describe a polynomial-time reduction from $2$-SAT to $3$-SAT, and prove that your reduction is correct.

**Solution:** Let $\Phi$ be an arbitrary $2$-CNF boolean formula. We construct a $3$-CNF formula $\Phi'$ by splitting each clause $(a \lor b)$ in $\Phi$ into two clauses $(a \lor b \lor x) \land (a \lor b \lor \bar{x})$, where $x$ is a new variable. I claim that $\Phi$ is satisfiable if and only if $\Phi'$ is satisfiable.

- Suppose $\Phi$ is satisfiable. Fix an arbitrary satisfying assignment of $\Phi$. Consider an arbitrary clause $(a \lor b)$ in $\Phi$; at least one of the literals $a$ or $b$ must be true. Thus, no matter what value we assign to the new variable $x$, the new clauses $(a \lor b \lor x) \land (a \lor b \lor \bar{x})$ in $\Phi'$ each contain at least one true literal. We conclude that $\Phi'$ is satisfiable.

- Suppose $\Phi'$ is satisfiable. Fix an arbitrary satisfying assignment of $\Phi'$. Consider an arbitrary clause $(a \lor b)$ in $\Phi$. The corresponding pair of clauses $(a \lor b \lor x) \land (a \lor b \lor \bar{x})$ in $\Phi'$ each contain at least one true literal. No matter what value $x$ has, at least one of the literals $a$ or $b$ must be true, so the clause $(a \lor b)$ is satisfied. We conclude that $\Phi$ is satisfiable.

Transforming $\Phi$ into $\Phi'$ by brute force takes linear time.

Essentially the same algorithm reduces $k$-SAT to $(k + 1)$-SAT for any $k$. ■

**Solution (smartass):** See part (b). We don’t need to call the $3$-SAT algorithm. ■

**Rubric:** 5 points: standard poly-time reduction rubric (scaled). Yes, the smartass solution is worth full credit.

(b) Describe and analyze a polynomial-time algorithm for $2$-SAT. [Hint: This problem is strongly connected to topics covered earlier in the semester.]

**Solution:** Let $\Phi$ be an arbitrary $2$-CNF boolean formula. Suppose $\Phi$ has $n$ variables and $k$ clauses. We define a directed graph $G = (V, E)$ as follows:

- $V$ contains $2n$ vertices, which correspond to the possible literals in $\Phi$. Each literal becomes a single vertex, even if it appears in multiple clauses or no clauses (like the $3$-COLOR reduction shown in class, but not the INDEPENDENT-SET reduction).

- $E$ contains $2k$ edges: $\bar{a} \rightarrow b$ and $\bar{b} \rightarrow a$ for each clause $(a \lor b)$ in $\Phi$. Thus, the edges correspond to logical implications.

Once we construct the graph $G$, we compute its strong components using either of the algorithms in the lecture notes. Finally, return True if no variable vertex is in the same strong component as its negation; otherwise, return False. The entire algorithm runs in $O(V + E) = O(n + k)$ time.
With a bit more work, we can actually construct a satisfying assignment if one exists. Build the strong component graph of $G$, topologically order the strong components, and label each vertex by the position of its strong component in this topological order. For each variable $v$, there are three cases to consider.

- If $label(x) < label(\bar{x})$, set $x \leftarrow \text{FALSE}$.
- If $label(x) > label(\bar{x})$, set $x \leftarrow \text{TRUE}$.
- If $label(x) = label(\bar{x})$, then $x$ and $\bar{x}$ are in the same strong component, so there is no satisfying assignment. Abort immediately.

We can prove the algorithm correct as follows.

- Suppose no variable is strongly connected with its negation. Then we successfully assign a value to each variable. If some clause $(a \vee b)$ in $\Phi$ has no true literals, then we have a contradiction:

$$
label(a) < label(\bar{a}) \quad \text{because } a = \text{FALSE} \\
\leq label(b) \quad \text{because } \bar{a} \rightarrow b \in E \\
< label(\bar{b}) \quad \text{because } b = \text{FALSE} \\
\leq label(a) \quad \text{because } \bar{b} \rightarrow a \in E
$$

We conclude that every clause in $\Phi$ is satisfied, as claimed.

- On the other hand, if we find a variable and its negation in the same strong component, then no matter which value we assume for that variable, we can derive a contradiction, which implies that $\Phi$ has no satisfying assignment.

The entire algorithm runs in $O(V + E) = O(n + k)$ time.

**Rubric:** 4 points: standard graph-reduction rubric

(c) Why don’t these results imply a polynomial-time algorithm for 3SAT?

**Solution:** The reduction in part (a) goes in the wrong direction. If we want to prove that 2SAT is NP-hard, or that 3SAT can be solved in polynomial time, we need to describe a reduction from 3SAT to 2SAT.

**Rubric:** 1 point; all or nothing.
2. (a) Describe a polynomial-time reduction from \textsc{SubsetSum} to \textsc{Partition}.

**Solution (add two elements):** Let \( x \) denote the sum of the elements of \( X \).

- If \( k > x \), then \( \textsc{SubsetSum}(X, k) = \text{FALSE} \).
- Otherwise, if \( k = x/2 \), then \( \textsc{SubsetSum}(X, k) = \textsc{Partition}(X) \).
- Otherwise, \( \textsc{SubsetSum}(X, k) = \textsc{Partition}(X \cup \{2x + k, 3x - k\}) \).

The reduction is trivially correct when \( k > x \), and the proof of correctness when \( k = x/2 \) is the same as part (b), so let’s assume that \( k \leq x \) and \( k \neq x/2 \). Let \( Y = X \cup \{3x - k, 2x + k\} \). Observe that \( \sum Y = 6x \) and \( 2x + k \neq 3x - k \).

\( \implies \) Suppose \( X \) has a subset \( X' \) whose elements sum to \( k \). Let \( Y_1 = X' \cup \{3x - k\} \), and let \( Y_2 = Y \setminus Y_1 \). and \( \sum Y_1 = \sum X + 3x - k = 3x \), so \( \sum Y_2 = 3x \) as well. So \( Y \) can be partitioned into two subsets with the same sum.

\( \iff \) On the other hand, suppose \( Y \) can be partitioned into two subsets \( Y_1 \) and \( Y_2 \) with the same sum, which must be \( 3x \). Neither set can contain both \( 2x + k \) and \( 3x - k \), because \( (2x + k) + (3x - k) = 3x > x \). So suppose without loss of generality that \( 3x - k \in Y_1 \), and let \( X' = Y_1 \setminus \{3x - k\} \). Then \( X' \) is a subset of \( X \) whose elements sum to \( k \).

The reduction takes \( O(n) \) time (to compute \( x \)).

**Solution (add one element):** Let \( x \) denote the sum of the elements of \( X \).

- If \( k = x/2 \), then clearly \( \textsc{SubsetSum}(X, k) = \textsc{Partition}(X) \).
- Otherwise, \( \textsc{SubsetSum}(X, k) = \textsc{Partition}(X \cup \{|x - 2k|\}) \).

The reduction is trivially correct when \( k > x \), and the proof of correctness when \( k = x/2 \) is the same as part (b), so let’s assume that \( k \leq x \) and \( k \neq x/2 \). Let \( Y = X \cup \{|x - 2k|\} \), and let \( y = \sum Y \). If \( x > 2k \), we have \( y = 2x - 2k \); otherwise, we have \( y = 2k \).

\( \implies \) Suppose \( X \) has a subset \( X' \) whose elements sum to \( k \). If \( x > 2k \), let \( Y' = X' \cup \{x - 2k\} \); in this case, we have \( \sum Y' = x - k = y/2 \). On the other hand, if \( x < 2k \), let \( Y' = X' \); in this case, we have \( \sum Y' = k = y/2 \). In either case \( Y \) can be partitioned into two subsets \( (Y' \text{ and } Y \setminus Y') \) with the same sum.

\( \iff \) On the other hand, suppose \( Y \) can be partitioned into two subsets \( Y_1 \) and \( Y_2 \) with the same sum. Without loss of generality, suppose \( |x - 2k| \in Y_1 \). If \( x > 2k \), let \( X' = Y_1 \setminus \{x - 2k\} \); otherwise, let \( X' = Y_2 \). In both cases, \( X' \) is a subset of \( X \) whose elements sum to \( k \).

The reduction takes \( O(n) \) time (to compute \( x \)).

**Rubric:** 5 points, standard poly-time reduction rubric (scaled). These are not the only correct reductions. No penalty if the output of the reduction could be a multiset instead of a set. \(-\frac{1}{2} \) if the output of the reduction can contain negative integers.
(b) Describe a polynomial-time reduction from \textsc{Partition} to \textsc{SubsetSum}.

\textbf{Solution:} \textsc{Partition}(Y) = \textsc{SubsetSum}(Y, y/2), where \(y\) is the sum of the elements of \(Y\).

\(\implies\) Suppose \(Y\) can be partitioned into two subsets \(Y_1\) and \(Y_2\) with the same sum. Then \(\sum Y_1 = \sum Y_2 = y/2\). So \(Y\) has a subset whose elements sum to \(y/2\).

\(\impliedby\) On the other hand, suppose \(Y\) has a subset \(Z\) whose elements sum to \(y/2\). Then the elements of \(Y \setminus Z\) sum to \(y/2\) as well, which implies that \(\sum Z = \sum(Y \setminus Z)\). So \(Y\) can be partitioned into two subsets with the same sum.

The reduction requires \(O(n)\) time (to compute \(y\)).

\textbf{Rubric:} 5 points standard reduction rubric (scaled). This is not the only correct reduction.
3. **Pebbling** is a solitaire game played on an undirected graph $G$, where each vertex has zero or more pebbles. A single pebbling move removes two pebbles from some vertex $v$ and adds one pebble to an arbitrary neighbor of $v$. (Obviously, $v$ must have at least two pebbles before the move.) The PebbleClearing problem asks, given a graph $G = (V, E)$ and a pebble count $p(v)$ for each vertex $v$, whether there is a sequence of pebbling moves that removes all but one pebble. Prove that PebbleClearing is NP-hard.

**Solution:** We describe a reduction from the undirected Hamiltonian path problem. Let $G = (V, E)$ be an arbitrary undirected graph, and suppose $G$ has $n$ vertices. We construct a new graph $H$ by adding a new vertex $x$ to $G$, and then adding edges from $x$ to every vertex of $G$. (Vertex $x$ is sometimes called an apex.) Finally, we place two pebbles on $x$ and one pebble on every other vertex.

I claim that $G$ has a Hamiltonian path if and only if we can clear all but one of the pebbles from $H$.

$\Rightarrow$ Suppose $G$ has a Hamiltonian path $v_1 \to v_2 \to \cdots v_n$. Then we can clear all but one of the pebbles from $H$ by making pebble moves from $x$ to $v_1$, then from $v_i$ to $v_{i+1}$ for all $i$ in increasing order, and finally from $v_n$ back to $v_{n-1}$.

$\Leftarrow$ Suppose we can clear all the pebbles from $H$. Fix a sequence of pebble moves. We start with $n + 1$ pebbles and end with one pebble, so the sequence must contain exactly $n + 1$ pebble moves. For each index $i$, let $u_i \to v_i$ denote the $i$th pebble move, which removes two pebbles from $u_i$ and adds one pebble to $v_i$. The definition of pebble moves implies that $u_i, v_i$ is an edge in $H$, for every index $i$. Every vertex in $H$ contains at least one pebble, so there must be at least one pebble move away from each vertex; it follows that the vertices $u_i$ are distinct.

Because $x$ is the only vertex with two pebbles initially, we have $u_1 = x$. Now I need to prove the following claim:

**Claim:** For every index $1 \leq i \leq n$, we have $v_i = u_{i+1}$; moreover, the following conditions hold just after the $i$th pebble move: $u_1, \ldots, u_i$ have no pebbles; $u_{i+1}$ has two pebbles; and every other vertex has one pebble.

**Proof:** Fix an index $1 \leq i \leq n$, and consider the placement of pebbles just before the $i$th move $u_i \to v_i$. By the induction hypothesis, $u_1, \ldots, u_{i-1}$ are empty, $u_i$ has two pebbles, and all other vertices have one pebble each. We must have $v_i = u_j$ for some $j > i$, since otherwise no more moves would be available. Thus, after the $i$th move, vertices $u_1, \ldots, u_i$ are empty, vertex $v_i$ has two pebbles, and all other vertices have one pebble each. Because $v_i$ is the only vertex with more than one pebble, we must have $u_{i+1} = v_i$. □
We conclude that the sequence $u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_{n+1}$ is a walk in $H$ that visits each vertex exactly once, or in other words, a Hamiltonian path in $H$. Removing the first vertex $x = u_1$ gives us a Hamiltonian path in the original graph $G$.

Transforming the original graph $G$ into the pebbled graph $H$ obviously takes only $O(V)$ time. ■

**Rubric:** 10 points: standard polynomial-time reduction rubric. This is not the only correct solution; in particular, there is also a (many-one) reduction from undirected Hamiltonian cycle. This is more detail than necessary for full credit.