1. For each of the following statements, first say whether or not it even "makes sense" — that is, whether it is a well-formed mathematical statement. If not, explain why not. If it is well formed, say whether it is known to be true, known to be false/wrong, or not yet known to be true or false, and explain why.

   (a) If $P = NP$, then there is a polynomial-time algorithm for the following decision problem: given a graph G and an integer k, are there k vertices that have no edges between any of them?

   > **Solution:** The statement makes sense and is true. First, the described problem is in NP, because there is an efficient verifier: the witness is a list of k vertices in the graph, and the verifier checks that all the graph's edges involve at least one vertex that is not in the list. Second, since we are assuming $P = NP$, the described problem (which is in NP) is in P, which means it has a polynomial-time algorithm.

   (b) Dijkstra's algorithm is in P, so it is in NP as well.

   > **Solution:** The statement does not make sense: P is a collection of (decision) *problems*, whereas Dijkstra's algorithm is an *algorithm*. An algorithm cannot be in P.
   > (Since "Dijkstra's algorithm is in P" can also be interpreted as a false statement, it would also be acceptable to say that the whole statement is false. The point here is to notice that P is a collection of *problems*, not algorithms.)

   (c) If NP is polynomial time, then $P = NP$.

   > **Solution:** The statement does not make sense: NP is a collection of (decision) *problems*; it cannot "be" polynomial time. An *algorithm* can be polynomial time (meaning, its runtime is polynomial in the input length).

   (d) 3SAT cannot be solved in polynomial time.

   > **Solution:** The statement is well formed, but it is unknown whether it is true or false. Nobody has yet found a polynomial-time algorithm for 3SAT (which would mean that $P = NP$), and nobody has yet proved that no such algorithm exists (which would mean that $P \neq NP$). In other words, determining whether this statement is true or false *is exactly the P versus NP question.*

   (e) There might be a polynomial-time algorithm for HAM-CYCLE, but no polynomial-time algorithm for 3SAT.

   > **Solution:** The statement is well formed, and is known to be false. Both HAM-CYCLE and 3SAT are NP-complete. We know that either *all* NP-complete problems have polynomial-time algorithms (if $P = NP$), or *none* of them do (if $P \neq NP$). So whether or not $P = NP$, the statement is false.

(f) 3SAT is NP-complete because it takes $O(2^n)$ time to try all possible assignments to the n variables of a 3SAT instance.

> **Solution:** The statement is well formed, but it is wrong: while 3SAT is indeed NP-complete, and it does take $O(2^n)$ time to try all possible assignments, these two facts have little to do with each other (and the latter is certainly not the reason for the former). First, trying all assignments is just *one possible* algorithm for solving 3SAT; there might be very different, more clever algorithms that run much faster (maybe even in polynomial time!). But the main point is that "NP-complete" has nothing to do with "trying all possible solutions." What makes a problem NP-complete is that it is in NP, and every NP problem reduces to it (i.e., it is "at least as hard" as every NP problem).

2. Do at least one (your choice) of Exercise 34.5-1, 34.5-2, or 34.5-5.

> **Solution:** We give a full solution to 34.5-1, and sketch the key ideas behind the other two exercises.
>
> Exercise 34.5-1: the main insight here is that a Hamiltonian cycle in a graph $G = (V, E)$ is merely a subgraph of G that cycles through all $|V|$ vertices in some order. In other words, the "cycle graph" on $|V|$ vertices is isomorphic to some subgraph of G (namely, a subgraph corresponding to a Hamiltonian cycle). So any algorithm that solves the subgraph-isomorphism problem on two *arbitrary* graphs can in particular be used to detect the existence of a Hamiltonian cycle in a given graph. We now proceed more formally.
>
> It is routine to show that the subgraph-isomorphism problem is in NP: a witness is just a bijection $\pi\colon V_1 \to V_2'$ between the vertices V_1 of G_1 and some subset $V_2' \subseteq V_2$ of the vertices in G_2. The verifier, given G_1, G_2, and the witness π, checks that π is indeed a bijection, and also checks that $(\pi(u), \pi(v)) \in E_2$ is an edge in G_2 for every edge $(u, v) \in E_1$. These are the exact conditions demonstrating an isomorphism between G_1 and a subgraph of G_2, so the verifier accepts only when given a "yes" instance with a valid witness (which is guaranteed to exist).
>
> We now show that subgraph isomorphism is NP-hard; we prove this by exhibiting a reduction from the NP-complete HAM-CYCLE problem to the subgraph-isomorphism problem. Our reduction is given an instance of HAM-CYCLE, i.e., a graph $G = (V, E)$. It outputs an instance (G_1, G_2) of subgraph-isomorphism, where $G_2 = G$ is the input graph and G_1 is simply a cycle graph on $n = |V|$ vertices (i.e., a graph with vertices labelled $1, 2, \ldots, n$ whose edges are $(1, 2), (2, 3), \ldots, (n - 1, n), (n, 1)$). This reduction clearly runs in polynomial (in fact, linear) time.
>
> To finish the proof, we claim that G is a "yes" instance of HAM-CYCLE if and only if $(G_1, G_2 = G)$ is a "yes" instance of subgraph isomorphism. Indeed, as we argued at the beginning, G has a Hamiltonian cycle if and only if $G_2 = G$ has a subgraph that is isomorphic to the cycle graph G_1.
>
> For Exercise 34.5-2, we reduce from 3SAT to 0-1 integer programming using the following main idea: we let true and false literals correspond to 1 and 0 respectively, and let each clause correspond to the sum of its literals. So, a clause is satisfied if and only if the sum of its literals is ≥ 1 (because its literals cannot all be false/0). In a little more detail: our reduction is given a 3CNF formula with m clauses on n variables z_i, and transforms it into an instance of 0-1 integer programming with m constraints (i.e., rows of A) on n corresponding 0-1 variables x_i. The reduction transforms

each given clause into a corresponding linear constraint, e.g., the clause $(z_i \vee \overline{z_j} \vee z_k)$ becomes the constraint $x_i + (1 - x_j) + x_k \geq 1$, which we rewrite as $-x_i + x_j - x_k \leq 0$. Collecting these m constraints immediately yields a linear system $Ax \leq b$, which the reduction outputs. It is routine to check that if the original formula is satisfiable by some true/false assignment to the z_is, then the linear system is 0-1 satisfiable by setting $x_i = 1$ if z_i is true and $x_i = 0$ otherwise. Conversely, if the linear system is satisfiable by some 0-1 assignment to the x_is, then the original formula is satisfiable by setting $z_i =$ true if $x_i = 1$ and $z_i =$ false otherwise.

For Exercise 34.5-5, we reduce from the subset-sum problem to the set-partition problem. Given integer values $s_1, \ldots, s_n$ and a target value t (i.e., an instance of subset-sum), our reduction computes $S = \sum_i s_i$ and outputs a set-partition instance consisting of all the values s_i along with $s_{n+1} = 2t - S$.* Now, if there is a subset of $s_1, \ldots, s_n$ summing to t, then the same subset of $s_1, \ldots, s_{n+1}$ sums to t, while the remaining numbers sum to $S - t + (2t - S) = t$ as well, yielding a partition. Conversely, if there is a valid partition of $s_1, \ldots, s_{n+1}$, then the two subsets of the partition must sum to t, because the sum of all the values is $S + (2t - S) = 2t$. Therefore, the subset in the partition that does *not* contain $2t - S$ is a subset of $s_1, \ldots, s_n$ that sums to t. So, the original input is a "yes" instance of subset-sum if and only if our reduction's output is a "yes" instance of set-partition.

[*Technically, $S - 2t$ might be equal to one of the s_i, in which case our output is actually a "multiset" (a set where duplicate values are allowed). A simple variation on this reduction can ensure that we create a true set.]