1. For any integer k, the problem kSAT is defined as follows:

- INPUT: A boolean formula Φ in conjunctive normal form, with exactly k distinct literals in each clause.
- OUTPUT: TRUE if Φ has a satisfying assignment, and FALSE otherwise.

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

Solution: One such reduction (of infinitely many possible ones) is as follows. Let

$$\Phi = \bigwedge_{i=1}^{n} \ell_{i,1} \vee \ell_{i,2}$$

be the instance to 2SAT; in the above description of Φ, $\ell_{i,1}$ and $\ell_{i,2}$ are *literals* for all $1 \le i \le n$, not variables. Construct 3CNF formula

$$\Phi' = \bigwedge_{i=1}^{n} (\ell_{i,1} \vee \ell_{i,2} \vee x) \wedge (\ell_{i,1} \vee \ell_{i,2} \vee \overline{x});$$

here, x is a variable *not* in Φ. Input Φ' into the black box algorithm $\mathscr{A}$ for 3SAT, and feed the output of $\mathscr{A}$ as the output of the constructed algorithm for 2SAT. Φ' has exactly twice the number of clauses as Φ and there are at most $2n$ variables. Thus, Φ' can be constructed by brute force in ***time $O(n)$*** by a scanning through once Φ. The reduction is linear-time and thus polynomial-time.

We now prove the correctness of this reduction by proving the following claim: Φ has a satisfying assignment $\iff$ Φ' has a satisfying assignment.

- $\Rightarrow$ Suppose there is an assignment A of the variables in Φ that makes Φ evaluate to TRUE. Fix $1 \le i \le n$. By the definition of $\wedge$, we have that $\ell_{i,1} \vee \ell_{i,2}$ evaluates to TRUE under A. By the definition of $\vee$, this gives either $\ell_{i,1} = \text{TRUE}$ or $\ell_{i,2} = \text{TRUE}$ under A. Define the assignment A' as one that coincides with A for variables in Φ and assigns *any* truth value to x. By the definition of $\vee$, both $\ell_{i,1} \vee \ell_{i,2} \vee x$ and $\ell_{i,1} \vee \ell_{i,2} \vee \overline{x}$ evaluate to TRUE under A'. Since this analysis holds for all $1 \le i \le n$, by the definition of $\wedge$, we have that $\bigwedge_{i=1}^{n}(\ell_{i,1} \vee \ell_{i,2} \vee x) \wedge (\ell_{i,1} \vee \ell_{i,2} \vee \overline{x})$ evaluates to TRUE under A'. But $\bigwedge_{i=1}^{n}(\ell_{i,1} \vee \ell_{i,2} \vee x) \wedge (\ell_{i,1} \vee \ell_{i,2} \vee \overline{x}) = \Phi'$, which implies Φ' has a satisfying assignment.
- $\Leftarrow$ Suppose there is an assignment A' of the variables in Φ' that makes Φ' evaluate to TRUE. Fix $1 \le i \le n$. By the definition of $\wedge$, $(\ell_{i,1} \vee \ell_{i,2} \vee x) \wedge (\ell_{i,1} \vee \ell_{i,2} \vee \overline{x})$ evaluates to TRUE under A'. By the definition of $\wedge$ again, $\ell_{i,1} \vee \ell_{i,2} \vee x$ and $\ell_{i,1} \vee \ell_{i,2} \vee \overline{x}$ both evaluate to TRUE under A'. It can easily be seen that both x and $\overline{x}$ cannot be TRUE under A'. Assume that x is TRUE under A' without loss of generality. Then $\overline{x}$ evaluates to FALSE, which implies that either $\ell_{i,1}$ or $\ell_{i,2}$ must evaluate to TRUE. We prove this by contradiction. Suppose both $\ell_{i,1}$ and $\ell_{i,2}$ evaluate to FALSE. Then $\ell_{i,1} \vee \ell_{i,2} \vee \overline{x}$ evaluates to FALSE, a contradiction. By the definition of $\vee$, then $\ell_{i,1} \vee \ell_{i,2}$ evaluates to TRUE under A' (and the restriction of the assignment A of A' to variables in Φ). Since this analysis holds for all $1 \le i \le n$, by the definition of $\wedge$, we have that

$\bigwedge_{i=1}^{n} \ell_{i,1} \vee \ell_{i,2}$ evaluates to TRUE under A. But $\bigwedge_{i=1}^{n} \ell_{i,1} \vee \ell_{i,2} = \Phi$, which implies Φ has a satisfying assignment. ■

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

Solution: Let

$$\Phi = \bigwedge_{i=1}^{n} \ell_{i,1} \vee \ell_{i,2}$$

be the instance to 2SAT; in the above description of Φ, $\ell_{i,1}$ and $\ell_{i,2}$ are *literals* for all $1 \leq i \leq n$, not variables. Construct a *directed* graph $G = (V, E)$ as follows:

- x is a variable in $\Phi \iff x, \overline{x} \in V$
- $\ell_1 \vee \ell_2$ is a clause for some *literals* ℓ_1 and ℓ_2 in $\Phi \iff \overline{\ell}_1 \to \ell_2, \overline{\ell}_2 \to \ell_1 \in E$

Compute the strong components of G using Kosaraju's algorithm and check if, for any variable x, x and $\overline{x}$ are in the same strong component. If so, return FALSE. Otherwise, return TRUE. Kosaraju's algorithm and checking the above condition combined require time $O(V+E)$ *in terms of the graph* G. Since $V \leq 2n$ and $E \leq 2n$ where n is the number of clauses in Φ, in terms of the original input Φ, this algorithm requires ***time*** $\boldsymbol{O(n)}$. This verifies that the algorithm is indeed polynomial-time. ■

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

Solution: We do not have enough information. It's worth noting that either of the following changes to the prompts of parts (a) and (b) would imply a polynomial-time algorithm for 3SAT:

- Part (a) asks for polynomial-time reduction from 3SAT to 2SAT instead of from 2SAT to 3SAT.
- Part (b) asks for a polynomial-time algorithm for 3SAT instead of 2SAT.

Also, just because you can use a harder problem (in this case 3SAT) to solve an easier one (in this case 2SAT) doesn't mean that is the *only* way to solve 2SAT (as you can see in part (b)). This is a subtle but very important distinction that is at the core of reductions. ■

2. Prove the following problems are NP-hard.

(a) Given an *undirected* graph G, does G contain a simple path that visits all but 17 vertices?

Solution: We prove this problem is NP-hard by a reduction from the undirected Hamiltonian path problem. Given an arbitrary graph G, let H be the graph obtained from G by adding 17 isolated vertices. Call a path in H ***almost-Hamiltonian*** if it visits all but 17 vertices. We claim that G contains a Hamiltonian path if and only if H contains an almost-Hamiltonian path.

- $\Rightarrow$ Suppose G has a Hamiltonian path P. Then P is an almost-Hamiltonian path in H, because it misses only the 17 isolated vertices.
- $\Leftarrow$ Suppose H has an almost-Hamiltonian path P. This path must miss all 17 isolated vertices in H, and therefore must visit every vertex in G. Since every edge in P is also in G, we conclude that P is a Hamiltonian path in G.

Constructing H can be done by brute force in ***time*** $\boldsymbol{O(V+E)}$, implying the reduction is polynomial-time. ■

(b) Given an *undirected* graph G with *weighted* edges, compute a *maximum-diameter* spanning tree of G. (The diameter of a tree T is the length of a longest path in T. (Don't use LONGEST-PATH for your reduction))

Solution: We prove this problem is NP-hard by a reduction from the undirected Hamiltonian path problem. Given an arbitrary undirected graph G, let H be the graph obtained from G by only assigning weight 1 to all edges. We claim that G contains a Hamiltonian path if and only a maximum-diameter spanning tree in H is a Hamiltonian path.

- $\Rightarrow$ Suppose G has a Hamiltonian path P in G. Since a path in an undirected graph is connected, undirected and acyclic, P is a tree by definition. It is spanning as P goes through every vertex by the definition of Hamiltonian. Because P is a path of length $V-1$ in H, the diameter of P (considering P as a spanning tree in H) is at least $V-1$. However, the diameter of P in H cannot be more than $V-1$ as no path in H has length more than $V-1$. Thus, P is a maximum-diameter spanning tree in H. This implies that a maximum-diameter spanning tree in H is necessarily a Hamiltonian path. Suppose otherwise. Then a maximum-diameter spanning tree T in H is not a Hamiltonian path. In other words, there is a vertex v such that $\deg_T(v) > 2$. In this case, there is no path in H that goes through every vertex, contradicting the existence of P.
- $\Leftarrow$ Suppose the maximum-diameter spanning tree T in H is a Hamiltonian path. Then T is a Hamiltonian path in G.

Checking if the maximum-diameter spanning tree T in H is a Hamiltonian path can be done in time $O(V+E)$. This is by checking that $\deg_T(v) \leq 2$ for every vertex v in H by scanning its adjacency list, returning TRUE if so and FALSE otherwise. Because the construction of H can also be done in time $O(V+E)$ by brute force, the reduction requires ***time*** $\boldsymbol{O(V+E)}$. This implies that the reduction is polynomial-time. ■

3. Let M be a Turing machine, let w be an arbitrary input string, and let s and t be positive integers. We say that M accepts w *in space* s if M accepts w after accessing at most the first s cells on its tape, and M accepts w *in time* t if M accepts w after at most t transitions. Prove that the following languages are decidable or undecidable:

   (a) $\{\langle M, w\rangle \mid M \text{ accepts } w \text{ in time } |w|^2\}$

   Solution: Define $L = \{\langle M, w\rangle \mid M \text{ accepts } w \text{ in time } |w|^2\}$. We can construct a Turing machine M' to decide L as follows. Given any $\langle M, w\rangle$, M' runs M on w for $|w|^2$ steps. If M' accepts w in that time, M' accepts $\langle M, w\rangle$. Otherwise, M' rejects $\langle M, w\rangle$. M' decides L so L is decidable. ■

   (b) $\{\langle M\rangle \mid M \text{ accepts at least one string } w \text{ in time } |w|^2\}$

   Solution: Define $L = \{\langle M\rangle \mid M \text{ accepts at least one string } w \text{ in time } |w|^2\}$. For the sake of argument, suppose there is an algorithm there exist an algorithm DecideL that decides the language L. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M, w⟩):
  Encode the following Turing machine M′:
      M′(x):
        run M on input w
        return True
  return DecideL(⟨M′⟩)
```

   Note that if M halts on w, M' accepts *every* input string using the *same* number of cells on its tape as its behavior does not depend on its input string x. Call this number k. Let w' be any string such that $|w'|^2 \geq k$; such a string exists as k is a fixed constant. We prove this reduction correct as follows:

   $\Longrightarrow$ Suppose $\langle M, w\rangle \in$ Halt.
   Then M halts on input w.
   Then M' accepts *every* input string x in k steps.
   Then M' accepts w' in time $|w'|^2$.
   So $\langle M'\rangle$ is in L.
   So DecideL accepts $\langle M'\rangle$.
   So DecideHalt accepts $\langle M, w\rangle$.

   $\Longleftarrow$ Suppose $\langle M, w\rangle \notin$ Halt.
   Then M does *not* halt on input w.
   Then M' diverges on *every* input string x.
   Then M' accepts *no* string.
   So $\langle M'\rangle$ is *not* in L.
   So DecideL rejects $\langle M'\rangle$.
   So DecideHalt rejects $\langle M, w\rangle$.

In both cases, DecideHalt is correct. But that's impossible, because Halt is undecidable. We conclude that the algorithm DecideL cannot not exist. So L must be undecidable. ■

(c) $\{\langle M, w\rangle \mid M \text{ accepts } w \text{ in space } |w|^2\}$

Solution: Define $L = \{\langle M, w\rangle \mid M \text{ accepts } w \text{ in space } |w|^2\}$. We can construct a Turing machine M' to decide L as follows. Suppose M' has $\langle M, w\rangle$ as its input. We assume M has the states Q and tape alphabet Γ. M' runs M on w for $k \triangleq |Q||w|^2|\Gamma|^{|w|^2}$ steps. If M accepts w in k steps *while accessing only the first* $|w|^2$ *cells on its tape*, M' accepts $\langle M, w\rangle$. Otherwise, M' rejects $\langle M, w\rangle$. M' decides L and so L is decidable.

The reasoning for the choice of M' is as follows. By definition, $|Q|$ is number of states of M, $|w|^2$ is number of possible tape head positions if the tape head is within the first $|w|^2$ cells of M and $|\Gamma|^{|w|^2}$ is the maximum number of possible strings that can be on the first $|w|^2$ cells of M. Thus, k is an *upper bound* on the number of possible configurations of M if M only ever accesses the first $|w|^2$ cells. This implies that if M doesn't accept w in k steps while accessing only the first $|w|^2$ cells, M would *never* accept input w after accessing only the first $|w|^2$ cells. ■

(d) $\{\langle M\rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2\}$

Solution: Define $L = \{\langle M\rangle \mid M \text{ accepts at least one string } w \text{ in space } |w|^2\}$. For the sake of argument, suppose there is an algorithm there exist an algorithm DecideL that decides the language L. Then we can solve the halting problem as follows:

```
DecideHalt(⟨M, w⟩):
  Encode the following Turing machine M′:
      M′(x):
        run M on input w
        return True
  return DecideL(⟨M′, w⟩)
```

Note that if M halts on w, M' accepts *every* input string using the *same* cells on its tape as its behavior does not depend on its input string x. Let k be the number cells M' uses on its tape when it accepts its input string x. Also, let w' be any string such that $|w'|^2 \geq k$; such a string exists as k is a fixed constant. We prove this reduction correct as follows:

$\Longrightarrow$ Suppose $\langle M, w\rangle \in$ HALT.

Then M halts on input w.

Then M' accepts *every* input string x using the first k cells of its tape.

Then M' accepts w' in space $|w'|^2$.

So $\langle M', w\rangle$ is in L.

So DECIDEL accepts $\langle M', w\rangle$.

So DECIDEHALT accepts $\langle M, w\rangle$.

$\Longleftarrow$ Suppose $\langle M, w\rangle \notin$ HALT.

Then M does *not* halt on input w.

Then M' diverges on *every* input string x.

Then M' accepts *no* string.

Then M' accepts *no* string w in space $|w|^2$.

So $\langle M', w\rangle$ is *not* in L.

So DECIDEL rejects $\langle M', w\rangle$.

So DECIDEHALT rejects $\langle M, w\rangle$.

In both cases, DECIDEHALT is correct. But that's impossible, because HALT is undecidable. We conclude that the algorithm DECIDEL cannot not exist. So L must be undecidable. ■

4. Let ($\Sigma = \{0, 1\}$):

$$X = \left\{ \begin{array}{ll} 0w & w \in A_{TM} \\ 1w & w \in \bar{A}_{TM} \end{array} \right\}$$

Show that neither X nor $\bar{X}$ is recursively-enumerable.

Solution: First let's show that X is not recursively enumerable. We know that the language $NA = \left\{\langle M, w\rangle | M \text{ is a TM and } M \text{ does not accept } w\right\}$ is not recursively enumerable (see lecture). In this case, the reduction is to create new yes instances of X by saying $\{1w|w \in NA\}$. Since the reduction is computable then we know that X is not recursively enumerable.

To show $\bar{X}$ is not recursively enumerable, we can reduce $\bar{A}_{TM}$ to $\bar{X}$. In this case the reduction would simply be $i = \{0w|w \in \bar{A}_{TM}\}$. Hence, i is only in $\bar{X}$ if $w \in \bar{A}_{TM}$. Since the reduction is computable, the language is not recursively enumerable. ■