Problems marked with **E** are graded on effort, which means that they are graded subjectively on the perceived effort you put into them, rather than on correctness. For bonus questions, we will not provide any insight during office hours or Piazza, and we do not guarantee anything about the difficulty of these questions. We strongly encourage you to typeset your solutions in LaTeX.

If you collaborated with someone, you must state their names. You must write your own solution and may not look at any other student's write-up.

1. Determine whether the following languages are always, sometimes or never decidable and provide justification. If they are always or never decidable, provide a proof. If they are sometimes decidable, provide an example for when they are decidable and when they are not.

   (a) $L = (L_1 \setminus L_2) \cap L_3$, where L_1, L_2 and L_3 are all decidable languages.

   Note: If A and B are sets, then $A \setminus B = \{x \in A : x \notin B\}$ (set difference of A and B).

   Solution: L is always decidable.
   First we will show that $L_1 \setminus L_2$ is always decidable if L_1 and L_2 are decidable languages. Let M_1 be a TM that decides L_1, and M_2 be a TM that decides L_2. We construct a TM M that decides $L_1 \setminus L_2$ as follows:

   M = "on input ($\langle x \rangle$):
   1: Run M_1 on input $\langle x \rangle$
   2: Run M_2 on input $\langle x \rangle$
   3: **if** M_1 accepted and M_2 rejected **then** accept
   4: **else** reject"

   Because M_1 and M_2 are deciders, they halt on all inputs, so it is clear that M halts on all inputs (i.e., it is a decider). We now show that M decides $L_1 \setminus L_2$.

   If $x \in L_1 \setminus L_2$, then $x \in L_1$ and $x \notin L_2$. Thus, M_1 will accept x, and M_2 will reject x. Thus M accepts x, as required.

   On the other hand, if $x \notin L_1 \setminus L_2$ then either $x \notin L_1$ or $x \in L_2$ (or both), so M_1 rejects x or M_2 accepts x (or both). In any of these cases, it is clear that M rejects x.

   Therefore, M decides $L_1 \setminus L_2$.

   Next, we will show that the intersection of two decidable languages is also always decidable.
   Let M_1, M_2 be deciders for L_1 and L_2 where both are decidable languages. We define a decider $M_{1\cap 2}$ for the language $L_1 \cap L_2$ as follows:

$M_{1\cap 2}$ = "on input ($\langle x \rangle$):
1: Run M_1 on input $\langle x \rangle$
2: Run M_2 on input $\langle x \rangle$
3: **if** M_1 or M_2 rejected **then** reject
4: **else** accept"

We show that $M_{1\cap 2}$ satisfies the requirements of a decider for $L_1 \cap L_2$. It is clear that $M_{1\cap 2}$ halts, because we only run M_1 and M_2 in $M_{1\cap 2}$, and they are both deciders and must halt.

If $x \in L_1 \cap L_2$, then $x \in L_1$ and $x \in L_2$, so $M_1(x)$ and $M_2(x)$ both accept, so $M_{1\cap 2}(x)$ accepts.

If $x \notin L_1 \cap L_2$, then $x \notin L_1$ or $x \notin L_2$ (or both). In the former case, $M_1(x)$ rejects, and in the latter case, $M_2(x)$ rejects. In either case, $M_{1\cap 2}(x)$ rejects.

Hence, as $L_1 \setminus L_2$ is decidable, L_3 is decidable, and the intersection of 2 decidable languages is always decidable, $(L_1 \setminus L_2) \cap L_3$ must also be decidable.

(b) $L = L_1 \cap L_2$, where L_1 and L_2 are both undecidable.

Solution: L is sometimes decidable.
Consider $L_1 = L_2 = L_{\text{HALT}}$. In this case $L = L_1 \cap L_2 = L_{\text{HALT}} \cap L_{\text{HALT}} = L_{\text{HALT}}$ which is undecidable.
Consider $L_1 = L_{\text{HALT}}$ and $L_2 = \overline{L_{\text{HALT}}}$, where both are undecidable. Then $L_1 \cap L_2 = \emptyset$ which is decidable.

(c) $L = L_1 \cup L_2$, where L_1 is decidable and L_2 is undecidable.

Solution: L is sometimes decidable.
Consider the case where $L_1 = \emptyset$ and $L_2 = L_{\text{HALT}}$. We know that L_1 is decidable, here is a decider E for it:
E on input $x \in \Sigma^*$:

i. Reject x.

In this case, we can see that $L = L_1 \cup L_2 = \emptyset \cup L_{\text{HALT}} = L_{\text{HALT}}$ which is undecidable.

Consider $L_1 = \Sigma^*$ and $L_2 = L_{\text{HALT}}$. In this case, L_1 is also decidable, here is a decider E for it:
E on input $x \in \Sigma^*$:

i. Accept x.

Then $L = L_1 \cup L_2 = \Sigma^* \cup L_{\text{HALT}} = \Sigma^*$ which is decidable.

2. For each of the following languages, state whether it is decidable or undecidable and prove your statement.

   (a) $L_{\text{MULTIPLE376}} = \{\langle M\rangle : \ |L(M)| \text{ is a multiple of 376, i.e. } |L(M)| = 376 \cdot n \text{ where } n \in \mathbb{N}_0\}$

Solution: $L_{\text{MULTIPLE376}}$ is undecidable, thus we are going to do a proof via Turing reduction from L_{ACC} to $L_{\text{MULTIPLE376}}$ ($L_{\text{ACC}} \leq_T L_{\text{MULTIPLE376}}$). Assume that there is some decider D of $L_{\text{MULTIPLE376}}$.

We are going to use the decider D to construct a decider T for L_{ACC}. Given a Turing Machine M and an input x as input, T works as follows:

T = "on input $\langle M\rangle, x$:
1: Construct Turing machine M_x as follows:

> M_x = "on input w:
> 1: **if** $w = x$ and $M(w)$ accepts **then accept**
> 2: **else reject"**

2: **if** $D(M_x)$ accepts **then reject**
3: **else accept** "

Note that

$$L(M_x) = \begin{cases} \{x\}, & \text{if } x \in L(M) \\ \emptyset, & \text{if } x \notin L(M), \end{cases}$$

and in particular $|L(M_x)|$ is a multiple of 376 (zero) if and only if $x \notin L(M_x)$. That is, $(\langle M\rangle, x) \in L_{\text{ACC}}$ if and only if $|L(M_x)|$ is not a multiple of 376. From this idea, if we have a $\langle M\rangle, x \in L_{\text{ACC}}$, then $|L(M_x)|$ will be 1, which is not a multiple of 376, and D will reject, so T will accept. If we have a $\langle M\rangle, x \notin L_{\text{ACC}}$, then $|L(M_x)|$ will be 0, which is a multiple of 376, and D will accept, so T will reject. Ultimately, we can see that T decides L_{ACC}, so $L_{\text{ACC}} \leq_T L_{\text{MULTIPLE376}}$. Since L_{ACC} is undecidable, we may conclude that $L_{\text{MULTIPLE376}}$ is undecidable. □

Alternatively, we can show that $L_{\varepsilon-\text{HALT}} \leq_T L_{\text{MULTIPLE376}}$ with a similar proof as above. We construct a decider E for $L_{\varepsilon-\text{HALT}}$ as follows:

E = "on input $\langle M\rangle$:
1: Construct Turing machine M' as follows:

> M' = "on input w:
> 1: **if** $w = \varepsilon$ and $M(w)$ halts (accepts or rejects) **then accept**
> 2: **else reject"**

2: **if** $D(M')$ accepts **then reject**
3: **else accept** "

Then we have

$$L(M') = \begin{cases} \{\varepsilon\}, & \text{if } M \text{ halts on } \varepsilon \\ \emptyset, & \text{if } M \text{ loops on } \varepsilon, \end{cases}$$

so that $|L(M')|$ is a multiple of 376 if and only if M loops on ε. The rest of the reasoning is as above. □

(b) Let $S = \{\langle \text{“pancakes”} \rangle, \langle \text{“waffles”} \rangle, \langle \text{“toast”} \rangle, \langle \text{“oatmeal”} \rangle\}$.
$L_{\text{ILoveBreakfast}} = \{\langle M \rangle : M \text{ accepts all strings in } S \text{ and loops on all other inputs}\}$

Solution: $L_{\text{ILoveBreakfast}}$ is undecidable. We show that $L_{\text{ACC}} \leq_{\text{T}} L_{\text{ILoveBreakfast}}$. Assume we have a decider B for $L_{\text{ILoveBreakfast}}$. We construct a decider A for L_{ACC}:

A = “on input $(\langle M, x \rangle)$:
1: Construct a machine M' as follows:

M' = “on input $\langle w \rangle$:
1: **if** $w \in S$ **then**
2: Run M on x
3: Accept w if M accepts x
4: Reject w if M rejects x
5: **else**
6: Loop on w

2: Run B on input $\langle M' \rangle$
3: **if** B accepted **then** accept $\langle M, x \rangle$
4: **else** reject $\langle M, x \rangle$”

Since B is a decider, A necessarily halts on any input. So all that remains is to show A decides L_{ACC}.

- If M accepts x, then M' accepts all strings in S ($L(M') = S$) and loops on all other inputs. So B accepts ($\langle M' \rangle$), and thus A accepts $\langle M, x \rangle$.
- If M rejects x, then M' rejects all inputs in S and loops on all other inputs. Hence B rejects ($\langle M' \rangle$), causing A to reject $\langle M, x \rangle$.
- If M loops on x, then M' loops on all inputs. Hence B rejects ($\langle M' \rangle$), causing A to reject $\langle M, x \rangle$.

So we have that A decides L_{ACC}.

(c) $\text{KNAPSACK} = \{\langle n, C, k, W, V \rangle : \exists S \subseteq \{1, \ldots, n\} \text{ s.t. } \sum_{i \in S} W[i] \leq C, \sum_{i \in S} V[i] \geq k\}$

In other words, KNAPSACK contains elements of the form $\langle n, C, k, W, V \rangle$ where:

- C is the capacity of the knapsack
- There are n items to choose from

- W is a list of item weights ($W[i]$ = weight of item i)
- V is a list of item values ($V[i]$ = value of item i)
- There exists a subset of items that have total weight $\leq C$ and total value $\geq k$ ($S \subseteq \{1, \ldots, n\}$ and $\sum_{i \in S} W[i] \leq C$ and $\sum_{i \in S} V[i] \geq k$)

Solution: KNAPSACK is decidable. The decider K, described below can be used to decide it. The strategy is to try all possible subsets of item numbers to see if any one of them satisfies the constraints.

K = "on input $(\langle n, C, k, W, V \rangle)$:
1: **for** each subset $S \in P(\{1, \ldots, n\})$ **do**
2: **if** $(\sum_{i \in S} W[i] \leq C) \wedge (\sum_{i \in S} V[i] \geq k)$ **then**
3: Accept $\langle n, C, k, W, V \rangle$
4: Reject $\langle n, C, k, W, V \rangle$

Now we must prove that K is a decider for KNAPSACK.

- If $\langle n, C, k, W, V \rangle \in$ KNAPSACK then there must exist a subset of items that has total weight less than or equal to C and total value greater than or equal to k. Hence, in this case there must be a set in the power set of $\{1, \ldots, n\}$ which makes K accept.
- If $\langle n, C, k, W, V \rangle \notin$ KNAPSACK then there exists no subset of items that has both, total weight less than or equal to C and total value greater than or equal to k. Hence, in this case there is no set in the power set of $\{1, \ldots, n\}$ which makes K accept. Once the for loop is done running and K has checked all possible subsets of items, it will reject $\langle n, C, k, W, V \rangle$.

(d) Let R(M) represent the set of strings that a Turing Machine M rejects.
$L_{\text{OPPOSITE}} = \{\langle M_1 \rangle, \langle M_2 \rangle : L(M_1) = \text{R}(M_2), \text{R}(M_1) = L(M_2)\}$

Solution: L_{OPPOSITE} is undecidable. We show that $L_{\text{ACC}} \leq_{\text{T}} L_{\text{OPPOSITE}}$. Assume we have a decider O for L_{OPPOSITE}. We construct a decider A for L_{ACC}:

A = "on input $(\langle M \rangle, x)$:
1: Construct a machine M' as follows:

> M' = "on input $\langle w \rangle$:
> 1: Reject w

2: Construct a machine M'' as follows:

> M'' = "on input $\langle w \rangle$:
> 1: Run M on x
> 2: If M accepts x, accept w
> 3: If M rejects x, reject w

3: Run O on input $(\langle M' \rangle, \langle M'' \rangle)$
4: **if** O accepted **then** accept
5: **else** reject"

Since O is a decider, A necessarily halts on any input. So all that remains is to show A decides L_{ACC}.

Notice that M' rejects all inputs. Therefore $L(M') = \emptyset$ and $R(M') = \Sigma^*$

- If M accepts x, then M'' accepts all strings in Σ^* (and rejects nothing). Hence $L(M'') = \Sigma^*$ and $R(M'') = \emptyset$. So we know that $L(M') = R(M'')$ and $R(M') = L(M'')$ which means that O will accept $\langle M', M'' \rangle$, causing A to accept $\langle M, x \rangle$.
- If M rejects x, then M'' rejects all strings in Σ^* (and accepts nothing). Hence $L(M'') = \emptyset$ and $R(M'') = \Sigma^*$. So we know that $L(M') \neq R(M'')$ and $R(M') \neq L(M'')$ which means that O will reject $\langle M', M'' \rangle$, causing A to reject $\langle M, x \rangle$.
- If M loops on x, then M'' loops on all strings in Σ^* (and thus accepts or rejects no input). Hence $L(M'') = \emptyset$ and $R(M'') = \emptyset$. So we know that $L(M') = R(M'')$ but $R(M') \neq L(M'')$ which means that O will reject $\langle M', M'' \rangle$, causing A to reject $\langle M, x \rangle$.

So we have that A decides L_{ACC}.

(e) $L_{\text{TRIPLETS}} = \{\langle M_1, M_2, M_3 \rangle : L(M_1) = L(M_2) = L(M_3)\}$

Solution: L_{TRIPLET} is undecidable. We show that $L_{\text{EQ}} \leq_{\text{T}} L_{\text{TRIPLET}}$. Assume we have a decider T for L_{TRIPLET}. We construct a decider E for L_{EQ}:

E = "on input $(\langle M_1, M_2 \rangle$:
1: Run T on input $\langle M_1, M_2, M_2 \rangle$
2: **if** T accepted **then** accept
3: **else** reject"

Since T is a decider, E necessarily halts on any input. So all that remains is to show T decides L_{EQ}.

- If $\langle M_1, M_2\rangle \in L_{\text{EQ}}$ then $L(M_1) = L(M_2)$, then $L(M_1) = L(M_2) = L(M_2)$. As a result, T will accept $\langle M_1, M_2, M_2\rangle$, causing E to accept $\langle M_1, M_2\rangle$.
- If $\langle M_1, M_2\rangle \notin L_{\text{EQ}}$ then $L(M_1) \neq L(M_2)$, then $L(M_1) \neq L(M_2) = L(M_2)$. As a result, T will reject $\langle M_1, M_2, M_2\rangle$, causing E to reject $\langle M_1, M_2\rangle$

So we have that E decides L_{EQ}.

3. Let L_1, L_2 be languages over $\Sigma = \{0, 1\}$. The *Hamming distance* between L_1 and L_2, written $d_H(L_1, L_2)$, is the number of elements by which L_1 and L_2 are different. In other words,

$$d_H(L_1, L_2) = |L_1 \triangle L_2|,$$

where $L_1 \triangle L_2$ is the *symmetric set difference*

$$L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1).$$

For example, $d_H(\{0, 1, 10\}, \{1, 10, 11\}) = 2$, since $|\{0, 1, 10\} \triangle \{1, 10, 11\}| = |\{0, 11\}| = 2$.

Let w_i be the i^{th} element in a list that enumerates all the elements in Σ^*.

Consider an infinite list of infinite languages represented in the following manner:

$$\begin{aligned} L_1 &= \{b_{11}, b_{12}, b_{13} \cdots\} \\ L_2 &= \{b_{21}, b_{22}, b_{23} \cdots\} \\ L_3 &= \{b_{31}, b_{32}, b_{33} \cdots\} \\ &\vdots \end{aligned}$$

where each $b_{ij} \in \{0, 1\}$. If b_{ij} is 1 this means that the i^{th} language (L_i) has the string w_j in it. On the other hand, if b_{ij} is 0 this means that the i^{th} language (L_i) does not have the string w_j in it.

For instance, L_1 in the example shown below represents a language which has the strings '0' and '00' but not 'ε' and '1'.

	w_1	w_2	w_3	w_4	...
	ε	'0'	'1'	'00'	...
L_1	0	1	0	1	...
L_2	1	0	0	0	...
L_3	0	1	1	1	...
L_4	0	0	1	1	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Cantor's diagonalization argument shows that the language represented by $\{\overline{b}_{11}, \overline{b}_{22}, \overline{b}_{33} \cdots\}$ has Hamming distance at least 1 from every language represented in the list, where $\overline{b}$ denotes the complement of the bit b.

E (a) Extend the argument by constructing, with justification, a language that has Hamming distance at least d from each language in the list, where d is some arbitrary given positive integer.

(b) *Optional Bonus:* Construct, with justification, a language that has *infinite* Hamming distance from each language in the list, i.e., it differs from each language in an infinite number of elements.

Solution:

(a) Using the same principle that we used to construct a language with Hamming distance 1 from every other language in the list, we can construct a language l that has Hamming distance d from every other language in the list in the following way:

$$l = \overline{b}_{1,1}, \overline{b}_{1,2}, ..., \overline{b}_{1,d}, \overline{b}_{2,d+1}, \overline{b}_{2,d+2}, ..., \overline{b}_{2,d+d}, ..., \overline{b}_{i,(i-1)d+1}, \overline{b}_{i,(i-1)d+2}, ..., \overline{b}_{i,(i-1)d+d}, ...$$

In order to better see how we build the language l, we re-write l

$$\begin{aligned} l = & \overline{b}_{1,1}, \overline{b}_{1,2}, ..., \overline{b}_{1,d}, \\ & \qquad \overline{b}_{2,d+1}, \overline{b}_{2,d+2}, ..., \overline{b}_{2,d+d}, \\ & \qquad\qquad \ddots \\ & \qquad\qquad \overline{b}_{i,d(i-1)+1}, \overline{b}_{i,d(i-1)+2}, ..., \overline{b}_{i,d(i-1)+d}, ... \end{aligned}$$

That is, we construct l by flipping elements 1 through d of the first language in the table: l_1, then by flipping bits $d+1$ through $2d$ of l_2, then by flipping bits $2d+1$ through $3d$ of l_3, and so on. l differs from l_i in bits $(i-1)d+1$ through id, which means that it differs from l_i in d bits. l may differ from l_i in more places than that, but we have guaranteed that at least d bits differ between l and l_n. So l has Hamming distance as least d from every language in the list. This is illustrated below for the case $d = 2$. The bold bits indicate the ones to flip, and we get the resulting language by taking the flipped bits first going rightwards, then downwards, starting at the top left corner.

$$\begin{aligned} L_1 &= \mathbf{b_{11}b_{12}}\ b_{13}\ b_{14}\ b_{15}\ b_{16}\ b_{17} \cdots \\ l_2 &= b_{21}\ b_{22}\ \mathbf{b_{23}b_{24}}\ b_{25}\ b_{26}\ b_{27} \cdots \\ l_3 &= b_{31}\ b_{32}\ b_{33}\ b_{34}\ \mathbf{b_{35}b_{36}}\ b_{37} \cdots \\ &\vdots \\ l &= \overline{b}_{11}\ \overline{b}_{12}\ \overline{b}_{23}\ \overline{b}_{24}\ \overline{b}_{35}\ \overline{b}_{36}\ \overline{b}_{47} \cdots \end{aligned}$$

(b) **There are many correct solutions.**

Correctness criteria: A correct solution must have the following properties:

i. Our method for choosing bits to flip must be well-defined: The value of l at the i^{th} position should be determined by flipping the i^{th} bit of exactly *one* of the l_i's.

ii. The Hamming Distance $d_H(l, l_i)$ must be infinite for each l_i: To construct l, we should flip infinitely-many bits of *each* language l_i in the table.

Solution: To construct l^* which has infinite Hamming Distance from each language l_i in the table, proceed as follows:

i. Flip every other bit of l_1, leaving the odd-indexed bits still undetermined (denote undetermined bits with ?).

index i	1	2	3	4	5	6	7	8	$\cdots$
i^{th} bit of l^*	?	$\overline{\mathbf{l_{1,2}}}$	?	$\overline{\mathbf{l_{1,4}}}$	?	$\overline{\mathbf{l_{1,6}}}$	?	$\overline{\mathbf{l_{1,8}}}$	$\cdots$

ii. Among the still-undetermined bits, flip every other bit of l_2: flip the (1 mod 4)-indexed bits, and leave the (3 mod 4)-indexed bits undetermined.

index i	1	2	3	4	5	6	7	8	$\cdots$
i^{th} bit of s^*	$\overline{\mathbf{l_{2,1}}}$	$\overline{l_{1,2}}$	?	$\overline{l_{1,4}}$	$\overline{\mathbf{l_{2,5}}}$	$\overline{l_{1,6}}$	?	$\overline{l_{1,8}}$	$\cdots$

iii. Among the still-undetermined bits, flip every other bit of l_3: flip the (3 mod 8)-indexed bits, and leave the (7 mod 8)-indexed bits undetermined.

index i	1	2	3	4	5	6	7	8	$\cdots$
i^{th} bit of l^*	$\overline{l_{2,1}}$	$\overline{l_{1,2}}$	$\overline{\mathbf{l_{3,1}}}$	$\overline{l_{1,4}}$	$\overline{l_{2,5}}$	$\overline{l_{1,6}}$	?	$\overline{l_{1,8}}$	$\cdots$

$\vdots$

iv. For the i^{th} iteration: among the still-undetermined bits, flip every other bit of l_i: flip the $(2^{i-1}-1 \bmod 2^i)$-indexed bits, and leave the $(2^i-1 \bmod 2^i)$-indexed bits undetermined

Correctness analysis:

i. This condition holds by construction: we only ever flip bits which were undetermined by previous iterations of the algorithm.

ii. For some $i \in \mathbb{N}$, there are infinitely-many $(2^{i-1}-1 \bmod 2^i)$-indexed bits of l_i which are flipped when constructing l^*. Thus the Hamming Distance $d_H(l_i, l^*)$ between l_i and l^* is infinite.

Another correct solution:

Let p_k denote the k^{th} prime number. So $p_1 = 2, p_2 = 3, p_3 = 5, p_4 = 7, \ldots$. If we produce a language l by flipping every p_k^n bit of every language l_k, then l will differ from every string in the list in an infinite number of places.

4. Let us define a *discoverable* set as a subset of natural numbers for which there exists an algorithm that outputs the numbers of this set in a finite number of steps, and an undiscoverable set as a subset of the natural numbers for which there exists no such algorithm. Prove that there exist uncountably many undiscoverable sets of natural numbers.

   Hint: The power set of a set A is the set of all subsets of A. For instance, $\mathcal{P}(\{0,1\}) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$. For a *countably* infinite set A, the power set $\mathcal{P}(A)$ is uncountably infinite.

   Solution: Since every discoverable set requires a Turing Machine that outputs its numbers, the set of discoverable sets can be no larger than the set of Turing Machines. In particular, we know that there are countably many Turing Machines, so this implies there are countably many discoverable sets. We also know that the set of all natural numbers is countably infinite, so we know that the powerset of natural numbers is uncountable. Therefore, if $\mathcal{C}$ is the set of all discoverable sets, this implies $\mathcal{P}(\{\mathbb{N}\}) \setminus \mathcal{C}$ is uncountable. This can be easily proved by contradiction. Suppose the set $\mathcal{K} = \mathcal{P}(\{\mathbb{N}\}) \setminus \mathcal{C}$ is countable, we also know that $\mathcal{C}$ is countable, then $\mathcal{P}(\{\mathbb{N}\}) = \mathcal{C} \cup \mathcal{K}$ is countable because we can *count* the union set by alternating between the elements of $\mathcal{C}$ and $\mathcal{K}$. This is a contradiction. Therefore the set of undiscoverable sets $\mathcal{K} = \mathcal{P}(\{\mathbb{N}\}) \setminus \mathcal{C}$ is uncountable, so there are uncountably many undiscoverable sets of natural numbers. □

   The reasoning above assumes that an algorithm for outputting a discoverable set takes no input. However, this doesn't actually restrict us in any way – given a program M that takes an input x, we can convert it into an inputless program M' by hardcoding both M and x into M'. Thus, there is a one-to-one mapping of program/input pairs to programs that don't take an input, and since the latter set is countable, so is the former.

   We can also directly show that the set of program/input pairs $\mathcal{M} \times \{0,1\}^*$ is countable; since the set of programs $\mathcal{M}$ is countable, and the set of inputs $\{0,1\}^*$ is countable, the set of pairs $\mathcal{M} \times \{0,1\}^*$ is also countable. We can demonstrate this with similar reasoning to how we showed $\mathbb{Q}^+$ is countable – construct a table with programs listed along the rows and inputs listed along the columns, then follow the diagonals to construct a list of the pairs. Each pair (M, x) can produce at most one discoverable set, so the set of discoverable sets is countable.

   Alternate method to show the number of discoverable sets is countable

   We can prove that the set of all discoverable sets is countably infinite using a different strategy. Our claim is that a subset S of $\mathbb{N}$ is discoverable if and only if it has finitely many elements.

   To prove this we need to prove the following:

   1. If S is a discoverable set then it has finitely many elements.
   2. If S has finitely many elements then it is a discoverable set.

   To prove the first direction let us first assume that a set S is discoverable. This means that there exists a Turing Machine that outputs the numbers in S in a finite number of steps. Assuming that S is an infinite set would lead to a contradiction as, in this case,

outputting all of its elements in finite time would be impossible. Hence, S must have a finite number of elements.

To prove the second direction let us first assume that S has finitely many elements. Then we can construct an algorithm to output all of its elements as follows:

```
ALG_S :
1: for each x ∈ S do
2:     Output x
```

Since every discoverable set is finite, we also know that the set of all discoverable sets is countably infinite. This is because we can create a list containing each discoverable set in the following manner:

For each natural number i, list the subsets of $\{0, 1, \ldots, i\}$.

Recall that the number of subsets of a set with i elements is 2^i. Hence for each step of this process, we will list 2^i subsets. This list will contain all possible finite subsets of $\mathbb{N}$. This can be understood better by thinking about when it will list an arbitrary finite subset S. The process will eventually list this subset S when i reaches $\max(S)$, the value of the maximal element in S. Since S is a finite set of natural numbers, it contains a unique maximal element.

5. Prove or disprove the following statements

   (a) If $L \leq_{\mathrm{T}} L'$ for some language L', then L is decidable.

   Solution: This statement is false.
   L is always decidable if $L \leq_{\mathrm{T}} L'$ and L' is decidable. However it is not necessarily decidable when L' is undecidable.
   If $L \leq_{\mathrm{T}} L'$ this just means that if there exists a blackbox decider for L' then it would be possible to construct a decider for L.
   Assuming we have a blackbox decider for an undecidable language can allow us to construct a decider for another undecidable language in many cases. That is, in many cases, an undecidable language is Turing reducible to another undecidable language.
   An example is shown in section notes 4 where we show that $L_{\mathrm{Acc}} \leq_{\mathrm{T}} \overline{L_\emptyset}$ (both L_{Acc} and $\overline{L_\emptyset}$ are undecidable).
   Also note that every language is Turing reducible to itself, even if it is undecidable.

   (b) For any decidable language L and an arbitrary language L', $L \leq_{\mathrm{T}} L'$

Solution: This statement is true. Because L is decidable, there is a TM B that decides it. Given a black box B' that decides L', a machine that decides L is simply... B itself.

The point is this: even though a black box that decides L' is available, the decider for L *is not required to use it* (and does not need to, in this case).

(c) For all $L, L \leq_{\mathrm{T}} \overline{L}$

Solution: This statement is true. Given a black box B' that decides $\overline{L}$, we can construct a machine B that decides L as follows: $B(x)$ just runs $B'(x)$ and outputs the opposite answer. Because B' halts on every input, so does B. And

$$x \in L \iff x \notin \overline{L} \iff B'(x) \text{ rejects} \iff B(x) \text{ accepts},$$

$$x \notin L \iff x \in \overline{L} \iff B'(x) \text{ accepts} \iff B(x) \text{ rejects},$$

so B decides L.