Solutions to Homework 7

We may grade a **subset of the assigned questions**, to be determined after the deadline, so that we can provide better feedback on the graded questions.
Unless otherwise stated, each question requires sufficient justification to convince the reader of the correctness of your answer.
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 name(s). You must write your own solution for all problems and may not look at any other student's write-up.

0. If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

1. In this question we examine the validity of verifiers and deciders for showing membership in NP or P.

(a) Recall

$$L_{\text{ACC}} = \{(\langle M \rangle, x) : M \text{ is a TM that accepts } x\}$$

Is the program V a verifier for L_{ACC}, where the certificate $C \geq 1$ is an integer represented in binary? (Hint: Is V efficient?) Show why or why not.

```
$V$ = "On input $(\langle M \rangle, x, C)$:
1. Run $M$ on $x$ for $C$ steps
2. If $M$ accepts $x$ in those $C$ steps, ACCEPT. Else, REJECT."
```

Solution: V does not prove that $L_{\text{ACC}} \in \mathsf{NP}$. While V is a correct verifier for L_{ACC}, it may not run in polynomial time. The runtime of V is at least $O(C)$ because of step 1, but C is not necessarily a polynomial function of $|\langle M \rangle|$ and $|x|$. Thus V fails the requirement for a verifier of an NP language to run in polynomial time with respect to the input from the language.

(b) Define

$$\textsf{First-Digit-Factorial} = \{n \in \mathbb{N} : \text{the first digit of the decimal representation of } n! \text{ is } 2\},$$

where the first digit is the most significant digit. Does F show that First-Digit-Factorial $\in$ P? Show why or why not.

```
$F$ = "On input $n$:
1. $product \leftarrow 1$
2. For $i$ from 2 to $n$
3.     $product \leftarrow product \cdot i$
4. $digit \leftarrow product$
5. While $digit \geq 10$
6.     $digit \leftarrow \lfloor digit/10 \rfloor$
7. If $digit = 2$, ACCEPT. Else REJECT."
```

Solution: F does not prove that First-Digit-Factorial $\in$ P. While F is a correct decider for First-Digit-Factorial, it does not run in polynomial time with respect to the input size $|n|$. F performs at least n steps, which is approximately equal to $2^{\log n}$. Therefore, F actually runs in exponential time with respect to the input size $\log n$, and thus does not show that First-Digit-Factorial $\in$ P.

2. Show that the following languages are in the class NP.

   Note: To prove that a language is in NP, you must provide a verifier and prove that your verifier satisfies correctness and efficiency.

   (a) Ham-Cycle $= \{\langle G \rangle : G$ is an undirected graph with a Hamiltonian cycle$\}$.
   Note: a Hamiltonian cycle for a graph is a path that visits each node exactly once and ends at the starting node.

Solution: We construct an efficient verifier for Ham-Cycle that expects a Hamiltonian cycle in the graph as a certificate. The verifier W works as follows:

```
1: function $W(G = (V, E), c)$
2:     if $c$ is not a permutation of the vertex set $V$, then
3:         reject
4:     for each vertex $v_i$ in the list $c = [v_1, v_2, \dots, v_n]$ do
5:         if $(v_i, v_{i+1}) \notin E$ (where $v_{n+1} = v_1$) then
6:             reject
7:     accept
```

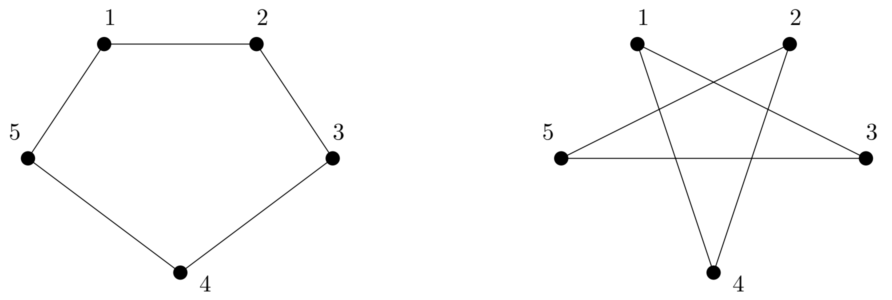
By construction, the verifier accepts exactly when $v_1 \to v_2 \to \cdots \to v_n \to v_1$ is a Hamiltonian cycle in G. So when a Hamiltonian cycle exists in G, there is a certificate $c = [v_1, \dots, v_n]$ that makes $V(G, c)$ accept. If there doesn't exist a Hamiltonian cycle in G, any candidate certificate c will either not have each vertex appearing exactly once, or will attempt to use edges that do not exist, so $V(G, c)$ will reject.

Finally, the verifier runs in polynomial time with respect to the length of the first input: we can perform the first check in $O(|V|)$ time by seeing if c has the proper number of vertices (with no duplicates), and we can perform the checks in the for-loop by looking up edges in E.

We have described an efficient verifier for Ham-Cycle, so Ham-Cycle $\in$ NP.

   (b) For two graphs $G = (V, E)$, $G' = (V', E')$, an *isomorphism* between them is a bijection $\gamma\colon V \to V'$ such that $(a, b) \in E$ if and only if $(\gamma(a), \gamma(b)) \in E'$. Similarly, G and G' are said to be *isomorphic* if there exists an isomorphism between them.

   For example, the following two graphs are isomorphic, by the bijection γ where $\gamma(1) = 1, \gamma(2) = 3, \gamma(3) = 5, \gamma(4) = 2, \gamma(5) = 4$.



Define Graph-Isomorphism $= \{(\langle G\rangle, \langle H\rangle) : G, H$ are isomorphic graphs$\}$.

Solution: We construct an efficient verifier for Graph-Isomorphism that expects an isomorphism γ (represented as a list of input-output pairs) between the two graphs as a certificate. The verifier W works as follows:

```
function W($G = (V, E), H = (V', E'), c$)
    if $c$ does not represent a bijection $\gamma\colon V \to V'$ then
        reject
    for each $(v, u) \in E$ do
        if $(\gamma(v), \gamma(u)) \notin E'$ then
            reject
    for each $(u', v') \in E'$ do
        if $(\gamma^{-1}(u'), \gamma^{-1}(v')) \notin E$ then
            reject
    accept
```

This algorithm first checks that the certificate represents a bijection $V \to V'$, which can be done efficiently by checking the set of all values $\gamma(v)$ for $v \in V$ against V' itself, and similarly checking all values $\gamma^{-1}(v')$ for $v' \in V'$ against V. It then does two loops which iterate through E and E' and check existence of edges. This all can be implemented straightforwardly in time polynomial in $|G|$ and $|H|$, so the verifier is efficient in the size of the input graphs G, H.

We now argue that the verifier is correct. First suppose that $G = (V, E), H = (V', E')$ are isomorphic, so there is an isomorphism γ between G and H. Then, by definition, γ is a bijection between V and V', and $(u, v) \in E$ if and only if $(\gamma(u), \gamma(v)) \in E'$. By construction, the verifier accepts $G, H, c = \gamma$.

Now suppose that G, H are not isomorphic. By construction of the verifier, it accepts G, H, c only if c represents an isomorphism between G and H. Because no such isomorphism exists, no certificate c can make $W(G, H, c)$ accept, as required.

We conclude that W is an efficient verifier for Graph-Isomorphism, and therefore Graph-Isomorphism $\in$ NP.

(c) Define Not-Prime $= \{n \in \mathbb{N} :$ n is not a prime number$\}$

Solution: In order to show that Not-Prime $\in$ NP, we need to build an efficient verifier. The certificate for our verifier will be an integer C.

$D =$ "V = On input (n, C):
1. Check whether C is an integer greater than 1 and less than n. If not, $REJECT$
2. Check whether $n \bmod C \equiv 0$. If so, $ACCEPT$. Else, $REJECT$."

$n \in \textsf{Not-Prime} \implies$ There exists some integer C where $1 < C < n$ that divides n $\implies V$ ACCEPTS given C
$n \notin \textsf{Not-Prime} \implies$ There exists no integer C where $1 < C < n$ that divides n $\implies V$ REJECTS for all C

This algorithm runs in polynomial time as comparing the value of a number to 1 and n is polynomial in time with respect to the size of the integers. The mod operation is also polynomial time with respect to the size of n. Thus our verifier is efficient with respect to our inputs.

3. (a) Let A, B be two languages in P. Then show the language $A \cup B$ is also in P.

Solution: Let P_A be an efficient decider for A and P_B be an efficient decider for B. Consider the following decider $P_{A \cup B}$ for $A \cup B$:

```
function $P_{A\cup B}(x)$
    Run $P_A(x)$
    Run $P_B(x)$
    if either accepted then
        accept
    else
        reject
```

If $x \in A \cup B$, then at least one of $x \in A$ or $x \in B$ must hold, so at least one of $P_A(x)$ or $P_B(x)$ accepts, and $P_{A\cup B}(x)$ accepts. If $x \notin A \cup B$, then $x \notin A$ and $x \notin B$, so both $P_A(x)$ and $P_B(x)$ reject, and $P_{A\cup B}(x)$ rejects.
This is efficient because both A and B are efficiently decidable. $P_A(x)$ runs in time $O(|x|^k)$ for some constant k and $P_B(x)$ runs in time $O(|x|^m)$ for some constant m. Thus, $P_{A\cup B}(x)$ runs in time $O(|x|^{k+m})$, which is polynomial with respect to $|x|$ since $k + m$ is a constant.

(b) Let A, B be two languages in NP. Then show the language $A \cup B$ is also in NP.

Solution: Let V_A be an efficient verifier for A and V_B be an efficient verifier for B. Consider the following verifier $V_{A\cup B}$ for $A \cup B$:

```
function $V_{A\cup B}(x,c)$
    Run $V_A(x,c)$
    Run $V_B(x,c)$
    if either accepted then
        accept
    else
        reject
```

Suppose $x \in A \cup B$. Then either $x \in A$ or $x \in B$. So there is some certificate c_A showing that $x \in A$ or some certificate c_B showing that c_B showing that $x \in B$. Either of c_A or c_B certifies that $x \in A \cup B$, so $V_{A\cup B}(x,c)$ accepts for some c.

If $x \notin A \cup B$, then there is no choice of c_A such that $V_A(x,c)$ accepts and similarly for c_B. So no certificate will cause $V_{A\cup B}(x,c)$ to accept.

Efficiency follows from the efficiency of V_A and V_B, using the same reasoning as the previous part.

(c) Let $A, B \in \mathsf{P}$. Define $AB = \{ab : a \in A, b \in B\}$ where ab denotes the concatenation of a and b. For example, if $a = 100111$ and $b = 0011$ are bitstrings then $ab = 1001110011$. Show that $AB \in \mathsf{P}$.

Solution: Consider the following decider for AB:

Let P_A be an efficient decider for A and P_B be an efficient decider for B.

Consider the following decider for AB which takes input $x = x_1, x_2, \ldots, x_n$ where x_i is the i-th character of the string:

```
function $P_{AB}(x = x_1x_2x_3\ldots x_n)$
    if ($P_A(\varepsilon)$ and $P_B(x)$) or ($P_A(x)$ and $P_B(\varepsilon)$) accept then
        accept
    for $i = 1, 2, \ldots, n-1$ do
        if $P_A(x_1 \ldots x_i)$ and $P_B(x_{i+1} \ldots x_n)$ accept then
            accept
    reject
```

Observe that $x \in AB$ if and only if there is some way to partition x into a prefix p and suffix s such that $p \in A$ and $s \in B$. The decider simply brute-forces over all possible partitions of x so it always gives a correct result.

The main issue is efficiency. There are $O(|x|)$ different partitions of x, so the time taken to brute-force is

$$\begin{aligned}\sum_{i=0}^{n} T_A(i) + T_B(n-i) &= \sum_{i=0}^{n} T_A(i) + T_B(i) && \text{(by symmetry of the sum)}\\ &\le \sum_{i=0}^{n} T_A(n) + T_B(n)\end{aligned}$$

$$
\begin{aligned}
&= O(n)(T_A(n) + T_B(n)) \\
&= O(n)(T_A + T_B)(n) \\
&= O(n)\,\mathrm{poly}(n) = \mathrm{poly}(n)
\end{aligned}
$$

where T_A is the time complexity of P_A and T_B is the time complexity of P_B. Note that we use closure of polynomials under addition as both T_A and T_B are both polynomials in n.

4. This question explores the complexity class $\mathsf{coNP} = \{\overline{L} \mid L \in \mathsf{NP}\}$. Conceptually, NP contains the languages whose "yes" instances can be verified efficiently, whereas coNP contains the languages whose "no" instances can be verified efficiently.

   (a) Prove that P is closed under set complement. That is, for any $L \in \mathsf{P}$, we have $\overline{L} \in \mathsf{P}$.

   Solution: If $L \in P$, there exists an efficient decider M_L for L. We can construct an efficient decider $M_{\overline{L}}$ that, on input x, simply runs $M(x)$ and outputs the opposite answer. (Equivalently, we can just swap the accept and reject states of M.) It is obvious that $M_{\overline{L}}$ is an efficient decider, and $M_{\overline{L}}(x)$ accepts if and only if $M(x)$ rejects, which happens if and only if $x \notin L$. Therefore, $M_{\overline{L}}$ decides $\overline{L}$, as desired.

   (b) Use part (a) to prove that if $\mathsf{P} = \mathsf{NP}$, the following set inclusions hold: (i) $\mathsf{NP} \subseteq \mathsf{coNP}$ (that is, for any $L \in \mathsf{NP}$, we have $L \in \mathsf{coNP}$), and (ii) $\mathsf{coNP} \subseteq \mathsf{NP}$.

   Solution: Observe that if $\mathsf{P} = \mathsf{NP}$, then because P is closed under set complement by part (a), so is NP. That is, if $L \in \mathsf{NP}$, then $\overline{L} \in \mathsf{NP}$ (which also means $\overline{L} \in \mathsf{NP} \implies L \in \mathsf{NP}$ by definition of complement). Then we have

   $$
   \begin{aligned}
   L \in \mathsf{NP} &\iff \overline{L} \in \mathsf{NP} && \text{(by set complement closure of } \mathsf{NP}\text{)} \\
   &\iff L = \overline{\overline{L}} \in \mathsf{coNP} && \text{(by definition of } \mathsf{coNP}\text{)}
   \end{aligned}
   $$

   Hence $\mathsf{NP} \subseteq \mathsf{coNP}$ and $\mathsf{coNP} \subseteq \mathsf{NP}$.

   Grading Note: Students may also show the two set inclusions separately, by showing the two implication directions separately.

   (c) Conclude from part (b) that if $\mathsf{NP} \neq \mathsf{coNP}$, then $\mathsf{P} \neq \mathsf{NP}$.

   Solution: We prove the contrapositive: if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{NP} = \mathsf{coNP}$. By part (b), $\mathsf{P} = \mathsf{NP}$ implies $\mathsf{coNP} \subseteq \mathsf{NP}$ and $\mathsf{NP} \subseteq \mathsf{coNP}$, so $\mathsf{NP} = \mathsf{coNP}$.