Problem 1

1. Give a detailed argument that for all decision problems D and E, if $D \leq_p E$ and $E \in NP$, then $D \in NP$.

   Proof. Assume D, E represented as formal languages. We wish to construct a polynomial-time verification algorithm for D. Since $D \leq_p E$, there exists a polynomial-time reduction function f such that for any $x \in \{0,1\}^*$, an instance $x \in D$ if and only if the transformed instance $f(x) \in E$. Also given $E \in NP$, there exists a polynomial time verification algorithm A such that

   $$E = \{x \in \{0,1\}^* : \exists \text{ certificate } y \text{ where } y \in O(|x|^c) \text{ such that } A(x,y) = 1\}$$

   So we have

   $$D = \{x \in \{0,1\}^* : \exists \text{ certificate } y \text{ where } y \in O(|f^{-1}(x)|^c) \text{ such that } A(f^{-1}(x),y) = 1\}$$

   Let $B(x,y) = A(f^{-1}(x),y)$, which is a composite of polynomial reduction function f and polynomial-time verification algorithm A, hence B is a polynomial-verification algorithm for D, so $D \in NP$ □

2. By analogy with the definition of NP-hardness, give a precise definition of what it means for a decision problem D to be coNP-hard.

   Solution. □

   D is coNP-hard if for all $L \in coNP$, $L \leq_p D$. In other words, D is coNP-hard if every problem in $coNP$ is polynomial-time reducible to D.

3. Show that if decision problem D is coNP-hard, then $D \in NP$ implies $NP = coNP$.

   Proof. 2 directions

   (a) Prove $coNP \subseteq NP$. Let arbitrary $L \in coNP$. Since D is coNP-hard, then $L \leq_p D$. Given D is NP, by result from part a of this question, we have $L \in NP$. So $L \in coNP \rightarrow L \in NP$ implies $coNP \subseteq NP$

   (b) Prove $NP \subseteq coNP$. Similarly, let arbitrary $L \in NP$, then $\overline{L} \in coNP$. Since D is coNP-hard, $\overline{L} \leq_p D$. Since D is NP, then by result from part a of this question, we have $\overline{L} \in NP$, this implies, $L \in coNP$. So $L \in NP \rightarrow L \in coNP$ implies $NP \subseteq coNP$

   Since $coNP \subseteq NP$ and $NP \subseteq coNP$, so $NP = coNP$ □

Problem 2

For each decision problem D below, state whether $D \in P$ or $D \in NP$, then justify your claim.

- For decision problems in P, describe an algorithm that **decides** the problem in polytime (including a brief argument that your decider is correct and runs in polytime).
- For decision problems in NP, describe an algorithm that **verifies** the problem in polytime (including a brief argument that your verifier is correct and runs in polytime), and give a detailed reduction to show that the decision problem is NP-hard for your reduction(s), you must use one of the problems shown to be NP-hard during lectures or tutorials.

1. ExactCycle
   - **Input** Undirected graph G and $k \in \mathbb{Z}$
   - **Question** Does G contain some simple cycle on **exactly** k vertices?

   Solution. □

   ExactCycle can be represented as

   $$\text{ExactCycle} = \{\langle G, k\rangle : G \text{ contains a simple cycle of size } k\}$$

   (a) Prove ExactCycle $\in NP$

   Proof. We prove this by finding an polynomial-time verification algorithm. Given an instance of ExactCycle $\langle G, k\rangle$. The certificate is a cycle of vertices $\langle v_0, \cdots, v_k\rangle$. The algorithm checks

   i. There are k unique vertex in the cycle, with $v_0 = v_k$
   ii. Each of (v_i, v_{i+1}) is a valid edge in E

   and outputs 1 (yes) if both checks are true and 0 (no) otherwise. Both steps can be done in polynomial time easily.

   i. We can check for uniqueness of vertices in the sequence by making $\binom{k}{2}$ pairwise comparison, which takes $O(k^2)$ time.
   ii. The second step is simply a look up in the graph and there are a total of k edges to verify, which takes a total of $O(k)$, assuming a constant time lookup in adjacency matrix representation of the graph.

   So the verification algorithm is a polynomial time algorithm. If the certificate is a simple cycle of size k, then the verification algorithm will output 1 accordingly as the checks the algorithm performs is equivalent in definition to a simple cycle of size k. If the certificate, either is not simple, does not contain a cycle, or contain invalid edges, the algorithm will output 0 accordingly. □

(b) Prove for all $L \in NP$, $L \leq_p$ ExactCycle (i.e. NP-hard)

Proof. By lemma in clrs, we can find a NP-complete problem Ham-Cycle and a polynoial time reduction algorithm mapping $x \in$ Ham-Cycle to $f(x) \in$ ExactCycle to prove that ExactCycle is NP-complete. Given an instance of Ham-Cycle $\langle G \rangle$, the reduction algorithm computes $k = |G.V|$ and outputs an instance of $\langle G' = G, k \rangle$ to ExactCycle. The transformation function f is polynomial, in fact constant as we are only computing the size of vertices in G. Now we prove that the transformation is a valid reduction

i. Suppose C is a hamiltonian cycle in G. Then we have $k = |G.V| = |C|$. We claim that C is a simple cycle of length k in G'. Indeed, we have $|C| = k$ by construction. Therefore there is a simple cycle of size k in G'

ii. Suppose there is a simple cycle of size k in G'. Let C be such simple cycle. We claim that C is hamiltonian cycle in G. There are k vertices in G by construction, the fact that C is a simple cycle of size k implies that C is a hamiltonian cycle, which is simply a simple cycle over every vertex (k of them).

□

2. LargeCycle

- **Input** Undirected graph G and $k \in \mathbb{Z}$
- **Question** Does G contain some simple cycle on **at least** k vertices?

Solution. □

LargeCycle can be represented as

$$\text{LargeCycle} = \{\langle G, k \rangle : G \text{ contains a simple cycle of size } \geq k\}$$

(a) Prove LargeCycle $\in NP$

Proof. We prove this by finding an polynomial-time verification algorithm. The algorithm is exactly that of the ExactCycle verification algorithm with one difference, we are checking if the certificate, a sequence of vertices, have length greater than or equal to k instead of testing if the length is equal to k. The complexity and correctness analysis follows similarly. □

(b) Prove for all $L \in NP$, $L \leq_p$ LargeCycle (i.e. NP-hard)

Proof. By lemma in clrs, we can find a NP-complete problem Ham-Cycle and a polynoial time reduction algorithm mapping $x \in$ Ham-Cycle to $f(x) \in$ LargeCycle to prove that LargeCycle is NP-complete. Given an instance of Ham-Cycle $\langle G \rangle$, the reduction algorithm computes $k = |G.V|$ and outputs an instance of $\langle G' = G, k\rangle$ to LargeCycle. The transformation function f is polynomial, in fact constant as we are only computing the size of vertices in G. Now we prove that the transformation is a valid reduction

i. Suppose C is a hamiltonian cycle in G. Then we have $k = |G.V| = |C|$. We claim that C is a simple cycle of length k in G'. Indeed, we have $|C| = k$ by construction. Therefore there is a simple cycle of size k in G', implying there is a simple cycle of size at least k in G'

ii. Suppose there is a simple cycle of size at least k in G'. Let C be such simple cycle. Since $|G'.V| = k$, the simple cycle has exactly size k. We claim that C is hamiltonian cycle in G. There are k vertices in G by construction, the fact that C is a simple cycle of size k implies that C is a hamiltonian cycle, which is simply a simple cycle over every vertex (k of them).

□

3. SmallCycle

- **Input** Undirected graph G and $k \in \mathbb{Z}$
- **Question** Does G contain some simple cycle on **at most** k vertices?

Solution. □

(a) Prove SmallCycle $\in NP$

Proof. We prove this by finding an polynomial-time verification algorithm. The algorithm is exactly that of the ExactCycle verification algorithm with one difference, we are checking if the certificate, a sequence of vertices, have length less than or equal to k instead of testing if the length is equal to k. The complexity and correctness analysis follows similarly. □

(b) Prove for all $L \in NP$, $L \leq_p$ SmallCycle (i.e. NP-hard)

Proof. .... □

Problem 3

Consider the following Partition search problem.

1. **Input** A set of integers $S = \{x_1, \cdots, x_n\}$ each integer can be positive, negative, or zero.

2. **Output** A partition of S into subsets S_1, S_2 with equal sum, if such a partition is possible; otherwise, return the special value *nil*. (S_1, S_2 is a partition of S if every element of S belongs to one of S_1 or S_2, but not to both.)

1. Give a precise definition for a decision problem PART related to the Partition search problem.

   Solution. $\square$

   Given a set of integers $S = \{x_1, \cdots, x_n\}$ where each integer x_i can be positive, negative, or zero. We want to find if there exists a partition S_1, S_2 with equal sums.

   $$\textsc{Part} = \left\{ \langle S_1, S_2 \rangle : \sum_{s \in S_1} s = \sum_{s \in S_2} s \quad \text{and} \quad S_1 \cup S_2 = S \text{ and } S_1 \cap C_2 = \emptyset \right\}$$

   Let PARTDECIDE be the algorithm that decides the decision problem PART, specifically

   $$\textsc{PartDecide}(S_1, S_2) = \begin{cases} 1 & \text{if } \langle S_1, S_2 \rangle \in \textsc{Part} \\ 0 & \text{otherwise} \end{cases}$$

2. Give a detailed argument to show that the PARTITION search problem is polynomial-time self-reducible. (Warning: Remember that the input to the decision problem does not contain any information about the partition if it even exists.)

   Solution. $\square$

   Note given a set of size n, there are $\binom{n}{2}$ possible different ways to separate the set into 2 non-empty subsets. Now we provide an efficient algorithm utilizing PARTDECIDE, assumed to be efficient, to solve for PARTITION

   ```
   Function Partition(S)
       for (S1, S2) be one of the (n choose 2) different partition of S do
           if PartDecide(S1, S2) then
               return (S1, S2)
       return nil
   ```

   (a) **Proof of correctness** There are $\binom{n}{2}$ possible different partition of S into S_1 and S_2 by Stirling number of second kind. In each iteration we test for if the currernt partition has equal sums by calling PARTDECIDE. Note that the algorithm returns (S_1, S_2) only if PARTDECIDE(S_1, S_2) evaluates to true and nil otherwise. Since PARTDECIDE returns 1 (true) if and only if the partitions (S_1, S_2) have equal sum, therefore the algorithm returns the correct output in this case. If we exhaust the for loop, then we have looked over every possible unique partition of S and have not found any that PARTDECIDE evaluates to 1 (true), therefore the algorithm returning nil is correct.

(b) **Runtime** The algorithm iterates over $\binom{n}{2}$ times, each iteration invovles calling PARTDECIDE, which we assume to have a worst case runtime of $O(T)$, and a constant time operation to assign the appropriate element to partition S_1 and S_2. The worst case complexity of the algorithm is therefore $O(n^2T)$

Given that the algorithm is correct and the runtime, $O(n^2T)$, is polynomial to the worst case runtime of PARTDECIDE. If T is polynomial, in other words, if there exists an efficient algorithm for solving PARTDECIDE, then we can solve PARTITION in (n^2T), which is still in polynomial time. Therefore the search problem PARTITION is **self-reducible**