1. Prove that if P = NP, then every non-trivial problem in this class is NP-complete.

Let P be a non-trivial problem in $\mathrm{P} = \mathrm{NP}$. We want to show that P in NP-Complete, that is, every problem $Q \in \mathrm{NP}$ reduces to P in polynomial time. Let I_1 be a fixed instance for P for which the answer is *Yes*, and I_2 a fixed instance for P for which the answer is *No*. Finally, let I be an input instance for Q. Since $\mathrm{NP} = \mathrm{P}$, there exists a polynomial-time algorithm to solve Q. Invoke this algorithm to solve Q on I. If the answer is *Yes* (respectively, *No*), the reduction algorithm generates the instance I_1 (respectively, I_2) for P. Since I_1 and I_2 are fixed instances, their sizes are constant, that is, do not depend on the size of I. Therefore the running time of the reduction algorithm is the same as that to solve Q on I. This shows that $Q \leqslant P$.

2. Graph isomorphism problem: Two graphs G, H (both directed or both undirected) are called isomorphic if there exists a bijection $f : V(G) \to V(H)$ such that $(u,v) \in E(G)$ if and only if $(f(u),f(v)) \in E(H)$.

(a) Prove that GI is in NP.

The map f is a certificate.

(b) Is GI NP-complete?

GI is not known to be NP-complete. Laszlo Babai proposes a pseudo-polynomial-time algorithm for this problem. See:

https://en.wikipedia.org/wiki/Graph_isomorphism_problem

3. Subgraph isomorphism problem: Given two graphs G and H, decide whether there exists an injective function $f : V(H) \rightarrow V(G)$ such that $(u,v) \in E(H)$ if and only if $(f(u),f(v)) \in E(G)$. Prove that SGI is NP-complete.

First, I show that SUBGRAPH-ISOMORPHISM is in NP. For an input (G,H) of SUBGRAPH-ISOMORPHISM, guess a subset $V' \subseteq V(G)$ together with a bijective function $f : V' \rightarrow V(H)$. Then, check whether f preserves the adjacency relations of V' in $V(H)$.

In order to show the NP-Hardness, I reduce CLIQUE to SUBGRAPH-ISOMORPHISM. Let (G,r) be an input for CLIQUE. We construct the input (G,K_r) for SUBGRAPH-ISOMORPHISM, where K_r is the complete graph on r vertices. G has an r-clique if and only if G has a subgraph isomorphic to K_r. This reduction evidently runs in polynomial time.

4. Partition problem: You are given n positive integers $a_1, a_2, \cdots, a_n$ such that

$a_1 + a_2 + \cdots + a_n = A$ is even.

Decide whether the given integers can be partitioned into two subcollections such that the sum of the integers in each subcollection is A / 2. Prove that PARTITION is NP-complete.

Use reduction from SUBSET-SUM. Let S, t be an instance of SUBSET-SUM with $A = \sum_{a \in S} a$. Create the instance $S \cup \{2A - t, A + t\}$ for PARTITION.

5. Bin-packing problem: You are given n objects of weights $w_1, w_2, \cdots, w_n$. You are given an infinite supply of bins each with weight capacity C. You want to pack all the objects in m bins such that m is as small as possible. Assume that each $w_i \leq C$.

(a) Frame an equivalent decision problem, and prove that the decision problem can be solved in poly-time if and only if the optimization problem can be solved in poly-time.

Equivalent decision problem: Decide whether the objects can be packed in m bins for any given $m \geqslant 1$. A solution of the optimization problem clearly indicates whether m bins suffice. Conversely, if we have an oracle solving the decision problem, we can invoke the oracle with $m = 1, 2, \ldots, n$ until the decision is *yes*. The running time increases by a factor of n only. We can do binary search to increase the running time by a factor of $\log n$ only.

(b) Prove that the decision version is NP-Complete.

Use reduction from PARTITION (Exercise 6.17). Let $S = (a_1, a_2, \ldots, a_n)$ be an input instance for PARTITION with $\sum_{i=1}^{n} a_i = 2t$. Take n objects of weights $w_i = a_i$, bins each of capacity $C = t$, and $m = 2$. The partition problem has a solution if and only if two bins suffice.

6. Knapsack problem: A thief finds n objects of weights $w_1, w_2, \cdots, w_n$ and integer-valued profits $p_1, p_2, \cdots, p_n$. The thief has a knapsack of capacity C. The goal of the thief is to pack objects in the knapsack without exceeding the capacity so as to maximize the profit of packed objects.

0,1 variant: Each object can be packed or discarded.
Fractional variant: Any fraction of any object can be packed.

(a) Prove that the fractional knapsack problem can be solved in polynomial time.

Pack the objects in the decreasing order of p_i / w_i values.
This is a greedy algorithm. Prove its correctness.

(b) Formulate an equivalent decision version of the 0,1 knapsack problem.

Decision problem: Given w_1, w_2, $\cdots$, w_n, p_1, p_2, $\cdots$, p_n, C, and a (positive) integer P, decide whether a profit of $\geqslant$ P is achievable without exceeding the knapsack capacity C.

(a) [If] Let M be a polynomial-time algorithm for solving the maximization problem. Using M, we determine the maximum profit P^*, and return *true* if and only if $P^* \geqslant P$.

[Only if] Let D be a polynomial-time algorithm for solving the decision problem. We invoke D multiple times with separate profit bounds P in order to determine the maximum profit P^*. Initially, we start with $L = 0$ and $R = \sum_{i=1}^{n} p_i$, since we definitely know that P^* must lie between these two values. We compute $P = \lfloor (L+R)/2 \rfloor$, and call D with this profit bound P. If D returns *true*, we conclude that P^* is between P and R, so we set $L = P$. On the other hand, if D returns *false*, we set $R = P - 1$, since P^* must be smaller than P. This binary search procedure is repeated until we have $L = R$. We output this value ($L = R$) as P^*.

The total number of invocations of D is $O(\log \sum_{i=1}^{n} p_i)$ which is $O(\log(np_{\max}))$. Since each invocation runs in polynomial time, the total running time is polynomial in n and $\log p_{max}$.

(c) Prove that the decision version of the 0,1 knapsack problem is NP-complete.

(b) Clearly, the decision version of the knapsack problem is in NP.

In order to prove its NP-hardness, we reduce PARTITION to it. Let $a_1, a_2, \ldots, a_n$ be an input instance for PARTITION with $A = \sum_{i=1}^{n} a_i$.

We consider n objects $O_1, O_2, \ldots, O_n$ such that the weight of O_i is $w_i = 2a_i$ and the profit of O_i is $p_i = 2a_i$. Finally, we take the knapsack capacity $C = A$ and the profit bound $P = A$. Clearly, this reduction can be done in polynomial time.

Suppose that $\sum_{j=1}^{k} a_{i_j} = A/2$ for some subcollection $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ of $a_1, a_2, \ldots, a_n$. This implies that $\sum_{j=1}^{k} w_{i_j} = 2 \times (A/2) \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} = 2 \times (A/2) \geqslant P$, that is, the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy the capacity constraint and the profit bound.

Conversely, suppose that the objects $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ satisfy $\sum_{j=1}^{k} w_{i_j} \leqslant C$ and $\sum_{j=1}^{k} p_{i_j} \geqslant P$. These, in turn, imply that $\sum_{j=1}^{k} 2a_{i_j} \leqslant A$ and $\sum_{j=1}^{k} 2a_{i_j} \geqslant A$, that is, $\sum_{j=1}^{k} a_{i_j} = A/2$. Therefore, the integers $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ satisfy the requirement of the PARTITION problem.

A is assumed to be even. If so, you can take $w_i = p_i = a_i$, and $C = P = A/2$.

7. Let G = (V,E) be an undirected graph, and k a positive integer. You want to determine whether the removal of some k or fewer edges from E makes G bipartite. Prove that this problem is NP-complete.

The problem is clearly in NP, because the list of edges to remove from E in order to make it bipartite is a succinct certificate for the problem. It is easy to verify whether the list contains $\leqslant k$ edges, and their removal from G leaves a bipartite graph.

For NP-Hardness, we make a reduction from MAX-CUT. Let (G, r) be an instance for MAX-CUT. If m is the number of edges in G, generate the instance (G, k) of the given problem, where $k = m - r$. G has a cut of size $s \geqslant r$ if and only if the removal of the remaining $m - s$ edges makes G bipartite. The number of edges removed is $m - s \leqslant m - r = k$.

8. Weighted MAX-CUT problem: Let $G = (V,E)$ be an undirected graph with each edge e carrying a positive weight w_e (assume to be a positive integer). The weight of a cut X,Y of V is the sum of the weights of the edges connecting X and Y. Let k be a positive integer. Decide whether G has a cut of weight $\geqslant$ k. Prove that the weighted MAX-CUT problem is NP-complete.

Weighted MAX-CUT is a generalized version of the (unweighted) MAX-CUT problem. More specifically, reduce MAX-CUT to wt-MAX-CUT as follows. Let (G, k) be an instance of MAX-CUT. Take the weight of each edge of G as 1, and pass the same G and k along with these weights to the output.