**Problem 1:** You are about to go backpacking with a friend and you have to decide how to spread the load among your two backpacks. You have \( n \) items, with item \( i \) having weight \( w_i \geq 0 \). Your goal is to assign each item to backpack 1 or backpack 2 to minimize the maximum load. Consider the following algorithm:

**Greedy:** Consider items in order \( i = 1, \ldots, n \). At step \( i \) assign item \( i \) to the backpack whose current load is lightest.

Let \( w^*(X) \) and \( w(X) \) be the optimal solution and **Greedy**'s solution for the item set \( X \).

(a) Prove that \( w(X) \leq 1.5 \cdot w^*(X) \) for each instance \( X \). (b) Show that there is an instance \( X \) for which the bound in (a) is tight.

**Solution 1:** (a) We need to find some relation between the loads of **Greedy**'s backpacks and the optimum load. The idea of the proof is to show two different ways of lower-bounding the optimum load by the loads of **Greedy**'s backpacks. Then we choose an appropriate linear combination of these lower bounds to get the desired result.

Consider a fixed instance \( X \). Denote by \( L_1 \) and \( L_2 \) the final loads of the first and second backpack, respectively, after running **Greedy** on this instance. Without loss of generality, we can assume that \( L_1 \leq L_2 \), for otherwise we can simply renumber the backpacks. With this assumption we have \( w(X) = L_2 \).

The total weight of all items is \( L_1 + L_2 \). In any assignment, including the optimal one, the maximum backpack load cannot be smaller than the average backpack load, which equals \( \frac{1}{2}(L_1+L_2) \). So we have our first lower bound on the optimum value:

\[
w^*(X) \geq \frac{1}{2}(L_1 + L_2).
\]  

We now derive the second bound. Let \( j \) be the last item added to backpack 2. When **Greedy** was about to add item \( j \), the load of backpack 2 was equal \( L_2 - w_j \). By the definition of **Greedy**, this load must have been lower than (or equal to) the load of backpack 1 at that time, which, in turn, was at most \( L_1 \). We can thus conclude that \( L_2 - w_j \leq L_1 \), that is \( w_j \geq L_2 - L_1 \). But also, the optimum load must be at least \( w_j \), that is \( w^*(X) \geq w_j \). Putting it together, we obtain our second bound:

\[
w^*(X) \geq L_2 - L_1.
\]  

We now take a linear combination of the two bounds. Adding inequality (1) to inequality (2) multiplied by \( \frac{1}{2} \), we get

\[
\frac{3}{2} w^*(X) \geq L_2 = w(X).
\]

and part (a) follows.
(b) We use an instance $X$ with three items of weights 1, 1, and 2, in this order. Greedy will put the first two items one in each backpack, so its final load will be 3, as in the figure below (the numbers show sizes):

In the optimum solution the load is 2, because we can put the two items of weight 1 in one backpack and the third one in the other. Thus the ratio is 1.5.

---

**Problem 2:** Design a Turing Machine for the language given below. Give a high-level description of the algorithm (preferably in pseudocode), and the complete transition function.

$$L = \{0^i1^i : i \geq 0\}$$

Assume the input alphabet is \{0, 1\}.

**Solution 2:** The idea is this. Suppose first the input has the form 00..011..1, a sequence of 0’s followed by a sequence of 1’s. The computation will consist of a number of phases, where in each phase we mark one 0 and one 1. If we run out of 0’s before running out of 1’s, or vice versa, we reject. Otherwise we accept. We also need to detect inputs that have a wrong format, that is they don’t consist of a block of 0’s followed by a block of 1’s.

There are many ways to make it more concrete. In the solution below, we always mark the first 0 and the last 1. We mark the 1’s by changing them to blanks. In pseudo-code:
Turing Machine $M$:

loop: if symbol = $B$ then accept
else
    if symbol = 1 then reject
else    // symbol = 0
        change 0 to $X$
        walk right till you find first $B$
        move one step left
        if symbol = 0 then reject
else    // symbol = 1
        change 1 to $B$
        move left till you reach an $X$
        move one step right
        goto loop

And a complete description, in the TM format is given below. To simplify notation, we use letters $s$, $a$ and $r$ for the start state, the accepting state and the rejecting state. Each transition $\delta(q,x) = (p,b,m)$ is written as a quintuple $q x \rightarrow p b m$. The states are $Q = \{s,a,r,q,p,u\}$, the alphabet is $\{0,1,X,B\}$.

---------------
Turing Machine $M$.
---------------

s B -> a B R    // accept
s 1 -> r 1 R    // reject
s 0 -> q X R    // mark and move right
q 0 -> q 0 R    // keep moving right ...
q 1 -> q 1 R
q B -> p B L    // first blank, move back
p 0 -> r 0 R    // reject
p 1 -> u B L    // mark, move left
u 1 -> u 1 L    // keep moving left ...
u 0 -> u 0 L
u X -> s X R    // left end, new phase
---------------

Problem 3: Prove that the following decision problem is NP-complete.

HalfHC:

Instance: A graph $G = (V,E)$;
Query: Is there a simple cycle in $G$ of length at least $n/2$, where $n = |V|$?
Solution 3: First, we show that HALFHC is in \( \mathbb{NP} \). This is quite simple: a non-deterministic Turing Machine can guess a sequence \( S = v_1v_2...v_k \) of \( k = \lceil n/2 \rceil \) vertices. Then it verifies whether all these vertices are different, and whether \( G \) has edges \((v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k) \) and \((v_k, v_1)\).

Next, we show that the Hamiltonian cycle problem, \( \text{HamCycle} \), reduces to HALFHC in polynomial time. The reduction is as follows: Suppose that \( G = (V, E) \) is an instance of \( \text{HamCycle} \), and let \( n = |V| \). We construct another graph \( G' = (V', E) \) from \( G \) by adding to \( G \) \( n \) isolated vertices:

Thus \( G' \) has \( n' = 2n \) vertices. The reduction can be clearly done in polynomial time. A Turing Machine that realizes the reduction simply copies \( G \) and adds \( n \) vertices to the vertex list.

We now prove the correctness of the reduction. We need to show the following claim: \( G \) has a Hamiltonian cycle if and only if \( G' \) has a cycle of length at least \( n'/2 \). We prove the "if" and "only if" parts separately.

\[ (\Rightarrow) \] Suppose that \( G \) has a Hamiltonian cycle \( H \). Then \( H \) is also a cycle in \( G' \) and its length is \( n = n'/2 \). Therefore \( G' \) has a cycle of length \( n'/2 \).

\[ (\Leftarrow) \] Suppose that \( G' \) has a cycle \( S \) of length at least \( n'/2 = n \). \( S \) cannot contain any of the vertices that are not in \( G \), since these vertices have no edges. Therefore \( S \) may contain only the vertices in \( G \). This means that \( S \) has exactly \( n \) vertices, and therefore \( S \) is a Hamiltonian cycle in \( G \).

Summarizing, we proved that the HALFHC is in \( \mathbb{NP} \) and that \( \text{HamCycle} \), which is known to be \( \mathbb{NP} \)-complete, reduces to HALFHC in polynomial time. Therefore HALFHC is \( \mathbb{NP} \)-complete.