Problem 1: Consider the following “Longest Increasing Sublist” problem.

**Input:** A list of integers \( L = [a_1, a_2, \ldots, a_n] \).

**Output:** A sublist \( L' = [a_{i_1}, a_{i_2}, \ldots, a_{i_k}] \) such that \( 1 \leq i_1 < i_2 < \cdots < i_k \leq n \) and \( a_{i_1} < a_{i_2} < \cdots < a_{i_k} \) and \( k \) is maximum.

For example, if \( L = [4, 1, 7, 3, 10, 2, 5, 9] \), then \( L_1 = [1, 3, 5, 9] \) and \( L_2 = [1, 2, 5, 9] \) are two optimal solutions, but \( [1, 2, 3, 4] \) is not a solution (it takes integers from \( L \) out of order), \( [1, 7, 3, 10] \) is not a solution (it is not increasing), and \( [4, 7, 10] \) is not an optimal solution (it is not as long as possible).

Give a dynamic programming algorithm to solve the Longest Increasing Sublist problem.

**Step 0:** *Describe the recursive structure of sub-problems.*

Any optimal solution for input \( [a_1, a_2, \ldots, a_n] \) either contains \( a_n \), or it does not. Consider sub-problems whose solutions have last element \( a_k \), for various values of \( k \).

**Step 1:** *Define an array (“semantic array”) that stores optimal values for arbitrary sub-problems.*

Let \( M[k] \) represent the length of a longest increasing sublist that ends with \( a_k \), for \( k = 1, 2, \ldots, n \).

**Step 2:** *Give a recurrence relation for the array values.*

\[
M[k] = \max\{ M[i] + 1 : 0 < i < k \land a_i < a_k \}, \text{ for } k = 1, 2, \ldots, n.
\]

\( \text{MAX} = \max_{k=1}^{n} M[k] \) is the optimal value.

**Step 3:** *Write a bottom-up algorithm to compute the array values, following the recurrence.*

```python
for k in [1, 2, ..., n]:
    M[k] := 1
    for i in [1, 2, ..., k-1]:
        if a_i < a_k and M[i] + 1 > M[k]:
            M[k] := M[i] + 1
```

**Step 4:** *Use the computed values to reconstruct an optimal solution.*

Use a second array \( N[k] \) to store the index of the second-last element in the longest sub-list that ends with \( a_k \).

```python
# Complete algorithm.
for k in [1, 2, ..., n]:
    M[k] := 1
    N[k] := k
    for i in [1, 2, ..., k-1]:
        if a_i < a_k and M[i] + 1 > M[k]:
            M[k] := M[i] + 1
            N[k] := i

# Figure out the last element in the longest increasing sub-list.
b := 1
for k in [2, 3, ..., n]:
    if M[k] > M[b]:
        b = k
```
Problem 2: Consider the problem of creating a weekly schedule of TA office hours. You are given a list of TA’s $t_1, t_2, \ldots, t_n$ and a list of time slots $s_1, s_2, \ldots, s_m$ for office hours. Each TA is available for some of the time slots and unavailable for others. Each time slot $s_j$ must be assigned at most one TA, and every week, each TA $t_i$ is responsible for some positive integer number of office hours $h_i$.

We want to know if there is a feasible schedule of office hours, i.e., if it is possible to assign time slots to TA’s to satisfy all of the problem constraints (each TA gets exactly $h_i$ time slots and each time slot gets at most one TA—some time slots may remain unfilled).

(a) Describe precisely how to model this problem as a network flow problem. (Don’t forget to specify all edge directions and capacities in your network.)

Solution: Create a network $N$ with

- vertices $V = \{s, s_1, \ldots, s_m, t_1, \ldots, t_n, t\}$,
- edges $E = \{(s, s_i) : 1 \leq i \leq m\} \cup \{(s_i, t_j) : 1 \leq i \leq m, 1 \leq j \leq n, \text{ and TA } t_j \text{ is available at time } s_i\} \cup \{(t_j, t) : 1 \leq j \leq n\}$, where $c(s, s_i) = 1$ and $c(s_i, t_j) = 1$ and $c(t_j, t) = h_j$ for $1 \leq i \leq m, 1 \leq j \leq n$.

(Note: It is also correct to do this with all edges directed in the opposite direction.)

(b) Explain clearly the correspondence between valid assignments of TAs to office hour time slots and valid integer flows in your network above.

Solution:

- Every valid assignment of TAs to time slots generates a valid flow in $N$ by setting $f(s_i, t_j) = 1$ iff $t_j$ is assigned to $s_i$, $f(s, s_i) = 1$ iff someone is assigned to time $s_i$, $f(t_j, t) = \text{the number of hours assigned to } t_j$.
- Every valid integer flow in $N$ corresponds to a valid assignment of TAs to time slots by assigning $t_j$ to $s_i$ for all edges with $f(s_i, t_j) = 1$, because no time can have more than one TA assigned and no TA $t_i$ can be assigned to more than $h_i$ times, by the capacity and conservation constraints.

Problem 3 [If you have time]:

Consider the following “teaching assignment” problem: We are given a set of profs $p_1, \ldots, p_n$ with teaching loads $L_1, \ldots, L_n$, and a set of courses $c_1, \ldots, c_m$ with number of sections $S_1, \ldots, S_m$, along with subsets of courses that each prof is available to teach. The goal is to assign profs to courses so that: (1) each prof $p_i$ assigned exactly $L_i$ courses, and (2) each course $c_j$ assigned exactly $S_j$ profs.

Show how to represent this problem as a network flow, and how to solve it using network flow algorithms. Justify carefully that your solution is correct and can be obtained in polytime.

Solution: Given input, create network with vertices $p_1, \ldots, p_n, c_1, \ldots, c_m$, source $s$, sink $t$, and edges $(s, p_i)$ of capacity $L_i$ for each $p_i$, edges $(c_j, t)$ of capacity $S_j$ for each $c_j$, edges $(p_i, c_j)$ of capacity 1 for each $p_i, c_j$ such that $p_i$ is available to teach $c_j$.

- Any assignment of profs to courses yields flow in network: set $f(p_i, c_j) = 1$ if $p_i$ assigned $c_j$, 0 otherwise; set $f(s, p_i) = \text{number of courses assigned to } p_i$; set $f(c_j, t) = \text{number of profs assigned to } c_j$. Value of this flow
= number of course sections assigned. This implies maximum flow in network at least as large as maximum number of course sections that can be assigned.

- Any integer flow in network yields assignment of profs to courses: assign \( p_i \) to \( c_j \) iff \( f(p_i, c_j) = 1 \). By capacity constraints, no prof can be assigned more than \( L_i \) courses, no course can be assigned more than \( S_j \) profs, and no prof will be assigned to a course they are unavailable to teach. This means the maximum number of courses sections that can be assigned is at least as large as the maximum flow value for the network.

In other words, maximum flow value = maximum number of course sections that can be assigned. So, find max flow \( f \) (in polytime). If \( |f| = L_1 + ... + L_n = S_1 + ... + S_m \), then it is possible to assign profs to courses to satisfy all constraints (as indicated above); otherwise it isn’t. If it is not possible, max flow yields max assignment possible. This could be used to determine set of courses that can be offered, or maximum teaching load for profs, for example.