NOTE TO STUDENTS: This file contains sample solutions to the term test together with the marking scheme and comments for each question. Please read the solutions and the marking schemes and comments carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Remember that although you may not agree completely with the marking scheme given here it was followed the same way for all students. We will remark your test only if you clearly demonstrate that the marking scheme was not followed correctly.

For all remarking requests, please submit your request **in writing** directly to your instructor. For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

Question 1. [10 MARKS]

A thief breaks into a store holding a knapsack that can carry up to a maximum weight $W > 0$. The store contains items $1, 2, \ldots, n$, where item i has value $v_i > 0$ and weight $w_i \geqslant 0$. The thief can steal some amount x_i of item i, where $0 \leqslant x_i \leqslant w_i$, and its value is $(x_i/w_i)v_i$, i.e., the fraction of the item's weight stolen times the item's value. The thief must decide what fraction of each item to steal, so as to maximize the total value of the stolen goods, subject to the constraint that their total weight must not exceed the knapsack's capacity W. Write a greedy algorithm to solve this problem. Prove the correctness of your algorithm, and compute its complexity.

SAMPLE SOLUTION:

```
1: procedure FractionalKnapsack(V = ⟨v_1, ..., v_n⟩, W = ⟨w_1, ..., w_n⟩)
2:     Sort the items so that v_1/w_1 ⩾ v_2/w_2 ⩾ ··· ⩾ v_n/w_n
3:     Set t := 0 # total weight of stolen items in knapsack so far
4:     Set i := 1 # next item to be considered
5:     while (i ⩽ n and t + w_i ⩽ W) do
6:         x_i = w_i
7:         t = t + w_i
8:         i = i + 1
9:     if (i ⩽ n) then
10:        x_i = W − t
11:        for (j = i + 1; j ⩽ n; j++) do
12:            x_j = 0
13:    return (x_1, ..., x_n)
14: Complexity: Θ(n lg n)
```

Proof. We now show that the greedy algorithm produces an optimal solution. Let $O = (y_1, \ldots, y_n)$ be an optimal solution, where $0 \leqslant y_i \leqslant 1$ for $i = 1, \ldots, n$. Let $G = (x_1, \ldots, x_n)$ be the greedy solution, where $x_1 = \cdots = x_k = 1$ for some $k \geqslant 0$, $0 \leqslant x_{k+1} \leqslant 1$, and $x_j = 0$ for all $j > k + 1$. Assume, G and O agree till stage i, i.e., $x_j = y_j$ for $1 \leqslant j \leqslant i$. Now, consider stage $i + 1$. Two things can happen:

- $x_{i+1} = y_{i+1}$: In this case, G and O agree till stage $i + 1$, and there is nothing to do.
- $x_{i+1} \neq y_{i+1}$: In this case, we have $x_{i+1} > y_{i+1}$ because of the way the greedy algorithm works. Let $w := x_{i+1} - y_{i+1}$. Now create a new solution $O' = (y'_1, \ldots, y'_n)$ as follows: set $y'_j = y_j$ for $j \leqslant i$, set $y'_{i+1} = x_{i+1}$, and decrease some or all of the y_j's, for $j = i + 2, \ldots, n$, so that the total weight in the knapsack remains the same. In other words, $(y'_{i+1} - y_{i+1})w_{i+1} = \sum_{k=i+2}^{n}(y_k - y'_k)w_k$.

  Now, the total value of the new solution O' is

$$\begin{aligned}
\sum_{i=1}^{n} v_i y'_i &= \sum_{i=1}^{n} v_i y_i + v_{i+1}(y'_{i+1} - y_{i+1}) - \sum_{k=i+2}^{n} v_k(y_k - y'_k) \\
&= \sum_{i=1}^{n} v_i y_i + \frac{v_{i+1}}{w_{i+1}}(y'_{i+1} - y_{i+1})w_{i+1} - \sum_{k=i+2}^{n} \frac{v_k}{w_k}(y_k - y'_k)w_k \\
&\geqslant \sum_{i=1}^{n} v_i y_i + \frac{v_{i+1}}{w_{i+1}}(y'_{i+1} - y_{i+1})w_{i+1} - \sum_{k=i+2}^{n} \frac{v_{i+1}}{w_{i+1}}(y_k - y'_k)w_k \\
&= \sum_{i=1}^{n} v_i y_i + \frac{v_{i+1}}{w_{i+1}}[(y'_{i+1} - y_{i+1})w_{i+1} - \sum_{k=i+2}^{n} (y_k - y'_k)w_k] \\
&= \sum_{i=1}^{n} v_i y_i \text{ (since, the term within square brackets is zero)}
\end{aligned}$$

  Since O is already optimal, the total value of O' cannot be greater than that of O. Hence, they must be equal. In other words, O' is also an optimal solution. Moreover, O' and G agree till stage $i + 1$. By induction, G will be equal to some optimal solution. In particular, G is optimal.

MARKING SCHEME:

A. 2 marks: correct format/structure
B. 3 marks: correct algorithm (aka pseudocode)
C. 4 marks: correct idea (part marks for incorrect proof with some correct ideas)
D. 1 mark: correct complexity

Question 2. [15 MARKS]

Given two sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, find the length of a longest subsequence common to both X and Y, and also find one such subsequence. Write a 5-step DP algorithm to solve this problem. In particular, describe carefully the recursive structure of the problem in order to justify the correctness of your recurrence. Also, compute the complexity of your algorithm.

SAMPLE SOLUTION:

Step I. Define $OPT_{i,j}$ as the length of a longest common subsequence of $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$. Now, two things can happen:

1. $x_i = y_j$: In this case, we already found a common element, and the problem reduces to finding a longest common subsequence of X_{i-1} and Y_{j-1}. Hence, $OPT_{i,j} = OPT_{i-1,j-1} + 1$.
2. $x_i \neq y_j$: In this case, we didn't find any common element yet, and the problem reduces to finding a longest common subsequence of X_{i-1} and Y_j or a longest common subsequence of X_i and Y_{j-1}. Thus, $OPT_{i,j} = \max\{OPT_{i-1,j}, OPT_{i,j-1}\}$

Step II. Define a 2-D array M for memoization, where $M[i, j]$ stands for the length of the longest common subsequence of $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$.

Step III. Writing the recurrence in terms of the entries of M, we get

$$M[i,j] = \begin{cases} M[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max\{M[i-1, j], M[i, j-1]\} & \text{if } x_i \neq y_j. \end{cases}$$

Also, $M[0, k] = M[l, 0] = 0$ for all $0 \leqslant k \leqslant n, 0 \leqslant l \leqslant m$.

Step IV. Write a bottom-up iterative algorithm to compute M.

```
1: procedure LongestCommonSubsequenceLength(X, Y)
2:     Define M[0...m, 0...n]
3:     for (k = 0; k ⩽ n; k++) do
4:         M[0, k] = 0
5:     for (l = 0; l ⩽ m; l++) do
6:         M[l, 0] = 0
7:     for (i = 1; i ⩽ m; i++) do
8:         for (j = 1; j ⩽ n; j++) do
9:             if (X[i] == Y[j]) then
10:                M[i, j] = M[i − 1, j − 1] + 1
11:            else
12:                M[i, j] = max{M[i − 1, j], M[i, j − 1]}
13:    return M[m, n]
14: Complexity: Θ(mn)
```

Step V. Finally, find a longest common subsequence.

```
1: procedure LongestCommonSubsequence(X, Y, M)
2:     S := ∅
3:     Set i := m, j := n
4:     for (k = 1; k ⩽ max{m, n}; k++) do
5:         if (X[i] == Y[j]) then
6:             S.prepend(X[i])
7:             i = i − 1, j = j − 1
8:         else
9:             if (M[i − 1, j] > M[i, j − 1]) then
10:                i = i − 1
11:            else
12:                j = j − 1
13:    return S
14: Complexity: Θ(max{m, n})
```

Marking scheme:

A. 3 marks: **Structure:** clear attempt to define an array indexed by subproblems, to give a recurrence for the array values (with justification), and to write an iterative algorithm based on the recurrence — even if these are done incorrectly

B. 4 marks: **Idea:** answer contains the two cases when corresponding elements of X and Y match or they don't and write the recurrence accordingly — even if this is poorly structured or written up

C. 8 marks: **Details:** correct array definition, correct recurrence with appropriate base cases and justification, correct algorithm (even if based on wrong recurrence), and correct runtime (even if algorithm is wrong). It is important that there are algorithms to compute both the length of a longest common subsequence and an example of one such subsequence. It is also important that the algorithms are written as psuedocodes.