COMPSCI330 Design and Analysis of Algorithms
Assignment 2

Due Date: Wednesday, February 5, 2020

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit the problems to GradeScope. You will be asked to **label your solution for individual problems**. Failing to label your solution can cost you 5% of the total points (3 points out of 60 for this homework).
- LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you need more time for your homework please use this form and submit a STINF.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Clarifications on Grading Policy:

1. Whenever you are asked to describe/design an algorithm, you always need to analyze its running time.

2. (Just for dynamic programming problems) You don't need to write a proof if you were not explicitly asked. If the problem is not dynamic programming, you always need to write a formal proof.

3. If the problem asks you to describe the states for dynamic programming, you need to have a sentence that describes the states clearly in English, similar to the one that we provided in Problem 1. **You will receive no credits for the sub-problem if the English description is missing.**

4. If the transition function you give has a minor bug (such as typo, incorrect base case or off-by-one error), then you will receive partial credit and this will not effect the credit for the algorithm/running time/proof of correctness. However, if your transition function has a **major flaw**, **you will receive no credits for the subsequent sub-problems.**

5. When you are designing a dynamic programming algorithm (with states, transition function and base cases already specified), we look for (a) the ordering of the states (in which you fill up the dynamic programming table); (b) how to get the output given the dynamic programming table. You can write a pseudo-code but it is not required.

Problem 1 (Dynamic Programming Table)**.** (16 points) In the Longest Common Subsequence (LCS) problem, we are given two sequences $a[1..4] = ATAC$ and $b[1..4] = ATCG$. Recall the state for the LCS problem is defined as:

Let $f[i, j]$ be the longest common subsequence for $a[1..i]$ and $b[1..j]$.

The base case is given as $f[0, j] = f[i, 0] = 0$, and the transition function is

$$f[i,j] = \max \begin{cases} f[i-1,j] \\ f[i,j-1] \\ f[i-1,j-1]+1 & \text{if } a[i]=b[j] \end{cases}$$

(a) (10 points) fill in the DP table.

i\ j	0	1	2	3	4
0					
1					
2					
3					
4					

(If you are using latex, you can create the table by copying the following code:)

```
\begin{tabular}{|c|c|c|c|c|c|}
\hline
i$\backslash$ j & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & & & & & \\
\hline
```

```
1 & & & & & \\
\hline
2 & & & & & \\
\hline
3 & & & & & \\
\hline
4 & & & & & \\
\hline
\end{tabular}
```

(b) (6 points) List the sequence of states that leads to the optimal solution. (That is, starting from $(4, 4)$, keep track of which option in the transition function is taken.)

Problem 2 (Diet Options)**.** (20 points) Rong decides to go on a diet. Now he is at a cafeteria which has n items. Item i has c_i calories. For each item, Rong also has a (very subjective) rating of its tastiness t_i. Rong is looking for some items to order, such that the meal will have at least A calories and at most B calories, while the sum of tastiness is as large as possible. Please design an algorithm to help Rong. More precisely, given n, A, B, c_i, t_i (all numbers are integers), output the maximum sum of tastiness for every item ordered. (Note: Rong will not order the same menu item twice.)

Note: For this problem you can use everything we learned in class and recitation, but you need to follow the instructions and write down the details, instead of just saying "it's the same as xxx".

(a) (10 points) Define the state (sub-problems), write the transition function, and specify the base cases.

(b) (10 points) Design an algorithm that outputs the maximum sum of tastiness for every item ordered. Analyze its running time. (You do not need to prove the correctness for this problem.)

Problem 3 (Rollercoaster Sequences)**.** (24 points) You are given a sequence of numbers $a[1..n]$. A rollercoaster sequence of level k $b[1..m]$ is a subsequence of a that can be decomposed into **at most** k segments (where neighboring segments share exactly one number), where the segments alternate between increasing and decreasing. For example, if $a[] = \{1, 5, 4, 7, 2, 8, 3, 9, 10, 6\}$, a rollercoaster sequence of level 3 can be $b[] = \{1, 4, 7, 2, 9, 10\}$, it can be decomposed into three segments $\{1, 4, 7\}, \{7, 2\}, \{2, 9, 10\}$, which are increasing, decreasing and increasing respectively. For this problem, the segments are allowed to have length 1, so that $\{5, 4, 7, 9, 10\}$ is also a valid rollercoaster sequence (its decomposition is $\{5\}, \{5, 4\}, \{4, 7, 9, 10\}$). **The first component should always be increasing.** Given a seuqnce of numbers $a[1..n]$ and k, find the length of the longest rollercoaster sequence of level k.

(a) (10 points) Define the state (sub-problems), write the transition function, and specify the base cases.

(b) (8 points) Design an algorithm for the Rollercoaster Sequences problem.

(c) (6 points) Prove the correctness of your algorithm.