COMPSCI330 Design and Analysis of Algorithms
Assignment 2

Due Date: Wednesday, September 20, 2017

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, and analyze its overall running time. If the problem requires you to prove the correctness of the algorithm, you need to provide a proof. 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 each problem as an *individual* pdf file for the correct problem on Sakai. 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 submit within one week of the deadline you will get half credit. Any submission after that will not receive any credit.

- **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. If the running time you get is worse than the suggested running time, you will get partial credit.

2. If the problem asks you to prove the correctness, you need to provide 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 description given in Problem 1b). **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-problem.**

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 (Pokemon Healer)**.** (20 points) Bob is playing Pokemon Go. After each battle, he needs to heal the Pokemon to full HP so that it can battle again. Suppose the Pokemon is missing m HP (m is an integer), Bob has n different potions that he can use, where each potion can recover $w_i < m$ points of HP (w_i's are integers, Bob has only one of each potion). If you use a subset $S \subset \{1, 2, ..., n\}$ of the potions, the total amount of healing is going to be the sum $\sum_{i \in S} w_i$. The Pokemon is fully healed if the total amount of healing $w = \sum_{i \in S} w_i$ is at least m. Bob really hates wasting his potions, so he would like to find a way to fully heal his Pokemon, while the total amount of healing is minimized. You can assume $\sum_{i=1}^{n} w_i \geq m$(the sum of all potions heal for at least m).

(a) (4 points) Prove that there is always a solution whose amount of healing is at most $2m - 1$ (if you can prove it for at most $2m - 2$ it is even better).

(b) (8 points) Let $a[i, j] = true$ if using a subset of first i potions (potions $\{1, 2, 3, ..., i\}$), it is possible to produce exactly j points of healing ($a[i, j] = false$ otherwise). Write the transition function for $a[i, j]$, and specify the base cases.

(c) (8 points) Design an algorithm to compute the smallest $w \geq m$ that can be achieved using his potions. Your algorithm should run in time $O(mn)$. Prove the correctness of the algorithm.

Problem 2 (Optimal Path)**.** (20 points) Alice is playing a game where she controls a character to walk on a $m \times n$ table. The character starts at the bottom-left corner (coordinate (1,1)) and wants to go to the top-right corner (coordinate (m, n)). The character can only move to the right (from (i, j) to $(i, j + 1)$) or up (from (i, j) to $(i + 1, j)$). Each square of the table has a non-negative value $v[i, j]$. The goal is to find a path from the starting point to the end point that maximizes the sum of $v[i, j]$ for all the squares (i, j) on the path.

(a) (6 points) Define the state (sub-problems), write the transition function, and specify the base cases.

(b) (6 points) Design an algorithm that outputs the maximum possible value. Your algorithm should run in time $O(mn)$.

(c) (8 points) Suppose the character learned two new moves that allows it to move from (i, j) to $(i+2, j-1)$ and $(i-1, j+2)$ (See Figure 1 Right). Explain how to modify the transition function and the algorithm. Prove the correctness of your new algorithm.

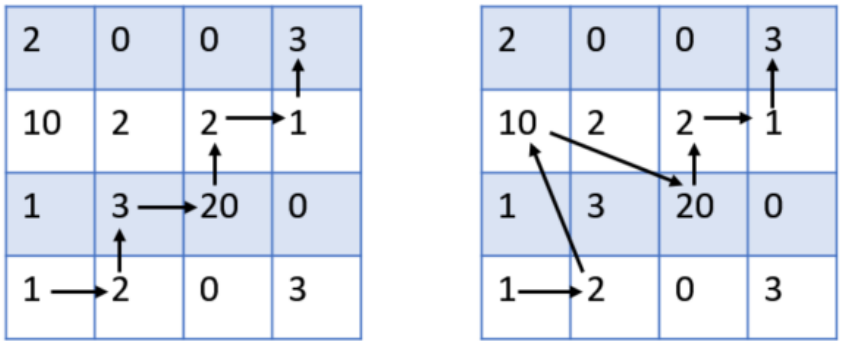

Figure 1: Left: Example for (a) (b), number in the table are $v[i, j]$, arrows show the optimal path. Output = 31. Right: Example for (c). Output = 38.

Problem 3 (Sum of Products)**.** (20 points) You are given a sequence of positive real numbers $a[1..n]$. You can now add '+' and '$\times$' signs between these numbers, and your goal is to generate an expression that has the largest value. As an example, if $a = \{2, 3, 0.5, 2\}$, then you should output the expression $2 \times 3 + 0.5 + 2 = 8.5$. This is larger than any other expression (e.g. $2 \times 3 \times 0.5 \times 2 = 6, 2 + 3 + 0.5 + 2 = 7.5, 2 + 3 \times 0.5 + 2 = 5.5$ ...). You must add either a '+' or a '$\times$' between two consecutive numbers, and you are not allowed to change the ordering of the numbers or add brackets. As usual the expression is evaluated to first compute the products and then the sum.

Design an algorithm that runs in time $O(n^2)$ and outputs the largest possible value. For this problem you can assume all additions, multiplications and comparisons of real numbers can be done in $O(1)$ time.

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

(b) (10 points) Design an algorithm for the Sum of Product problem. (No need to prove correctness for this problem. If your algorithm is slower than $O(n^2)$ you will not receive full credit.)