COMPSCI330 Design and Analysis of Algorithms
Assignment 2

Due Date: Thursday, January 31, 2019

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. 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 1a). **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 (Balancing Knapsacks)**.** (20 points) You are going on a hike with a friend. There are n items that you need to bring. The weight for i-th item is an integer w_i. The total weights for all the items is $m = \sum_{i=1}^{n} w_i$. You would like to split the items between you and your friend as balanced as possible. That is, suppose S is the set of items that you will carry, and T is the set of items that your friend will carry, you would like to minimize the maximum load $\max\{\sum_{i\in S} w_i, \sum_{i\in T} w_i\}$ (every item needs to be either in S or in T).

Your goal for this problem is to design an algorithm that computes the optimal load.

(a) (8 points) Let $a[i,j] = \mathit{true}$ if using a subset of first i items (item $\{1,2,3,...,i\}$), it is possible to form a knapsack of weight exactly j. Write the transition function for $a[i,j]$, and specify the base cases.

(b) (8 points) Design an algorithm to compute $a[i,j]$ for any $i = 0,1,...,n$ and $j = 0,1,2,...,m$. Your algorithm should run in time $O(mn)$. Prove the correctness of the algorithm.

(c) (4 points) Show how to use the dynamic programming table ($a[i,j]$'s) to compute the optimal load. That is, find the smallest number w such that it is possible to split the items into two knapsacks and both knapsacks have weight at most w.

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.

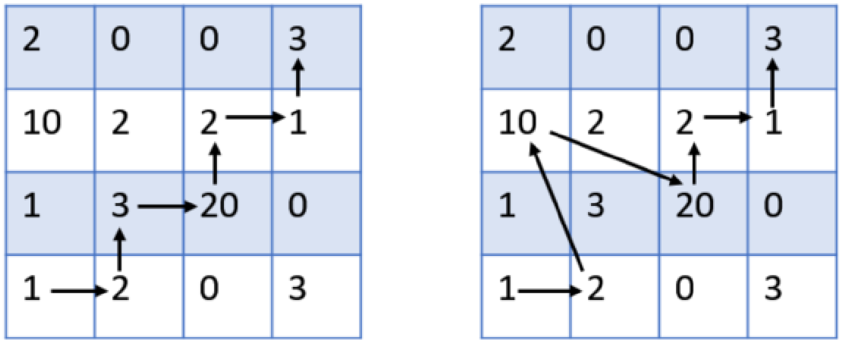

Figure 1: Left: Example for (a) (b), number in the table are $v[i, j]$, arrows show the optimal path. Output = 31. Right: Example for (c). Output = 38.

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.

Hint: there are different ways of thinking about this problem as making a sequence of decisions. The most basic one is to decide for each location whether you use + or $\times$. This does not work well (why?). Try to think about this problem as making a different sequence of decisions.

(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.)