EECS 3101 Fall 2021 (B & E) – Assignment 3

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of the course materials, and the course staff, that you consulted**. You must submit a PDF file with the required filename. The PDF file could be generated using LaTeX(recommended), Microsoft Word (saved as PDF, **not** the `.docx` file), or other editor/tools. The submission must be **typed**. Handwritten submissions are **not** accepted.

Due November 19, 2021, 10:00 PM (on eClass); required file(s): a3sol.pdf

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets]. Your submitted file does **not** need to include/repeat the questions — just write your solutions.

The assignment must be completed individually.

1. **[10] Maximum Segment Sum, Leveled Up**

   Recall that we studied a "maximum segment sum" problem when learning divide-and-conquer. In this exercise, we will solve an upgraded version of this problem using dynamic programming.

   Given a non-empty array `A` (size n) of integers (each of which could be positive, negative, or zero), we would like to find the maximum sum of a **non-empty, contiguous** segment (sub-array) of the array, i.e., we only consider segments with size ≥ 1. In addition, when calculating the segment sum, we have the option to **negate** the sign (e.g., -5 becomes 5, 42 become -42) of **at most one** (i.e., 0 or 1) element in the segment.

   For example, if the input array is `[3, -4, 4, 1, -5, 9, -2]`, then the maximum segment sum is 19 of the segment `[4, 1, -5, 9]` with -5 negated. If the input array is `[3, 1, 4, 2]`, then the answer would be 10 of the entire array without any negation performed.

   Requirement:

   - Your solution must use **bottom-up** dynamic programming.
   - Your algorithm must have a time complexity of $\mathcal{O}(n)$ and a space complexity of $\mathcal{O}(n)$.

   Describe the design of your solution by answering the following questions.

   (a) What are the subproblems and what data structures are used to store the answers to the subproblems?

   (b) What is the recursive relation between the answers to the subproblems? Explain your answer concisely.

   (c) Does the recursive relation have the optimal substructure? Justify your answer by providing a simple proof.

   (d) What are the base cases of the subproblems, and what are the answers for the base cases?

   (e) In which order do you populate the data structure storing the subproblem solutions?

   (f) What is the time complexity of your solution? Justify your answer concisely.

   (g) What is the space complexity of your solution? Justify your answer concisely.

   (h) Provide the pseudocode of your solution.

   Hint: It might be helpful to learn about the Kadane's algorithm for the regular maximum segment sum problem: `https://en.wikipedia.org/wiki/Maximum_subarray_problem`.

2. **[10] Coin Collection, Leveled Up**

   In this exercise, we will be collecting coins again, with the level-up that the coin collection becomes a two-player game.

   Given a row of n coins, each of which with a positive integer value. Player 1 and Player 2 take turns to take coins from the row, and the objective of each player is to take the maximum amount of **coin value** (**not** the number of coins!) before the row is emptied. Player 1 starts first.

   A variable S is used to keep track of the game in the following manner:

- Initially, $S = 1$.
- In each round of a player's turn, the player has the option to take **all of the first** k coins of the remaining coins, where k can be any value that satisfies $1 \leq k \leq 2S$. This means that if there are $\leq 2S$ coins left in the row, the player has the option to take all the remaining coins.
- At the end of each player's turn, the value of S is updated to $S = \max(S, k)$.
- The game continues until there is no coin left in the row.

Assume that **both players play the optimal strategy**, i.e., the strategy based on the correct dynamic programming solution that maximizes the coin value taken.

Your job is to find the maximum amount of coin **value** that **Player 1** can take in this game. For example, if the input is `[3, 6, 8, 5, 4]`, then the output should be 12 because of the following: if Player 1 takes one coin first (S stays 1), Player 2 takes two coins (S becomes 2), then Player 1 takes two coins again (S stays 2) and gets $3 + 5 + 4 = 12$ value in total. If Player 1 takes two coins first (S becomes 2), then Player 2 will take all three coins left (S becomes 3), and Player 1 gets $3 + 6 = 9$ value in total. The final answer is 12 since it is larger.

Requirements:

- You may use either memoization or bottom-up to implement your solution.
- The time complexity and space complexity of your algorithm must be both in polynomial time, i.e., $\mathcal{O}(n^j)$ for some $j > 0$, and you should try to make it as efficient as possible. More efficient solutions will be given higher marks.

Describe the design of your solution by answering the following questions.

(a) What are the subproblems and what data structures are used to store the answers to the subproblems?

(b) What is the recursive relation between the answers to the subproblems? Explain your answer concisely.

(c) Does the recursive relation have the optimal substructure? Justify your answer by providing a simple proof.

(d) What are the base cases of the subproblems, and what are the answers for the base cases?

(e) In which order do you populate the data structure storing the subproblem solutions?

(f) What is the time complexity of your solution? Justify your answer concisely.

(g) What is the space complexity of your solution? Justify your answer concisely.

(h) Provide the pseudocode of your solution.