Problem Set 2 (Version 1)

Updated: 12noon, Tue Sep 8 2020
Due: 4:45pm, Fri Sep 11 2020

Change log:

• Due date: pushed back by 2 days due to delays in grading Problem Set 1.

• Problem 1 (Version 1): intended complexity changed from \(O(n^2)\) to \(O(n^3)\). Also added two hints.

• Problem 3 (Version 1): restricted all numbers to be positive.

This problem set has a total of 5 problems on 3 pages. Written solutions should be submitted to GradeScope before 4:45pm on Friday Sep 11.

Your solutions should be submitted according to the guidelines [https://www.cc.gatech.edu/~rpeng/CS4540_F20/ProblemSetGuidelines.pdf](https://www.cc.gatech.edu/~rpeng/CS4540_F20/ProblemSetGuidelines.pdf) In particular:

1. If you choose not to submit a typed write-up, please write neat and legibly.

2. No credit will be given to solutions obtained verbatim from the Internet or other sources, and uploaded codes will be ran through similarity checking software.

1. There are \(n\) points on a 1-dimensional track, the \(i^{th}\) of which is at position given by the integer \(x_i\). Note that \(x_i\) may be a positive or negative integer. You start at position 0. You can move along the track in either direction at a rate of 1 unit per second, and have \(T\) seconds to visit points on the track. If you reach a point \(x_i\) at time \(t \leq T\) you receive a score \(T - t\) for that point. You will only receive a score for a point the first time you reach that point.

Give an algorithm to find the maximum sum of scores you can earn in \(T\) seconds. The algorithm should run in \(O(n^3)\).

Auto judge: [https://dmoj.ca/problem/ncco2d1p1](https://dmoj.ca/problem/ncco2d1p1)

Hints:

(a) Note that the \(t \leq T\) condition is unnecessary: negative valued points can be omitted.

(b) If there are \(k\) points that still need to be reached, then the loss in total points incurred by taking a step that costs time \(\hat{t}\) is \(k \cdot \hat{t}\).

(c) If we guessed the total number of points reached at the very start, \(k_{final}\), the number of points remaining at some state \([l, r]\) can be deduced from the length of the state by \(k_{final} - (r - l + 1)\).
2. There are \(n\) numbers in a sequence. Consider the game with two players alternatively taking numbers from the ends of the sequence. Each number may be taken by only 1 player, and the game is over when no number remains. The objective is to maximize the sum of numbers taken. Player 1 plays optimally, but player 2 plays by \textit{GREEDY}. That is, player 2 always take the number from the larger end, preferring the left end in case of a tie.

Give an \(O(n^2)\) time/space algorithm for calculating player 1’s maximum winnings.

Auto judge: \(https://dmoj.ca/problem/ecna05h\)

3. There are \(n\) items \textit{with positive values} in a row, the \(i^{th}\) of which has size \(x_i > 0\). You may perform the following operation as many times as you’d like:

- Select two items with equal size such that there is at most 1 item between them.
- Remove the two items along with any items that were between them from the row.
- In their place, add a single item with size equal to the sum of sizes of all items removed during this operation.

Give an algorithm to calculate the maximum size of a single item in the row after performing the above operation any desired number of times. The algorithm should run in \(O(n^3)\) time and take \(O(n^2)\) space.

Auto judge: \(https://dmoj.ca/problem/ccc16s4\)

4. You want to print a given string \(str[1 \ldots n]\) using a stack. Specifically, you can perform one of three operations to the stack:

(a) push a character on to it,
(b) remove the character from the top of it,
(c) print the character at the top of the stack.

Give an \(O(n^3)\) time, \(O(n^2)\) space algorithm to determine the minimum number of operations needed to print the given string \((str[1 \ldots n])\) and leave the stack empty at the end.

There is a example at the auto-judge link showing the sequence of operations required for the string “abba”.

Auto judge: \(https://dmoj.ca/problem/rmrc16j\)

5. You are given a string \(str\) of length \(n\). There are \(c\) operations you can perform any number of times on this string. Operation \(i\) is specified by a string \(s_i\) (\(|s_i| \leq n\)) and a value \(p_i\). The \(i^{th}\) operation is performed as follows:

- Select a continuous substring of \(S\) equal to \(s_i\) (or \(s_i\) reversed).
- Remove the substring from \textit{str}. This divides \textit{str} into two parts.
- Concatenate the two parts to form the new string \textit{str}.
- Add $p_i$ to your score.

Note that each operation may be performed multiple times, and $s_i$ may be reversed prior to comparison with a substring of \textit{str}. Give an algorithm for computing the maximum score that can be earned from a given string \textit{s} and \textit{c} specified operations. The algorithm should take $O(cn^4)$ time and use $O(cn^3)$ space, but better solutions exist depending on how you handle string comparison.

Auto judge: [https://dmoj.ca/problem/swerc17d](https://dmoj.ca/problem/swerc17d)