Challenge 2: Maximum-sum subarray

The *maximum-sum subarray problem* was first surveyed by Bentley in his “Programming Pearls” column of CACM in 1984. The solution returns the sum of a contiguous subarray within a one-dimensional array of numbers which has the largest sum.

In the two-dimensional case, the task is to find a submatrix such that the sum of its elements is maximized. This problem is widely used in applications such as pattern recognition, image processing, biological sequence analysis and data mining.

One-dimensional Case:

In the array [-2, -3, **4, -1, -2, 1, 5**, -3] the maximum-sum subarray is [4,-1,-2, 1, 5] with a sum of 7 (4 + (-1) + (-2) + 1 + 5 = 7). A brute force solution checks all subarrays (which are quadratically many), but the problem is solvable in linear-time using Kadane’s algorithm.

Kadane’s Algorithm for a one-dimensional array is given below as an implementation in C:

```
int maxSubArraySum(int a[], int size)
{
    int max_so_far = 0, max_ending_here = 0;
    for (int i = 0; i < size; i++)
    {
        max_ending_here = max_ending_here + a[i];
        if (max_ending_here < 0)
            max_ending_here = 0;
        else if (max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}
```

The One-dimensional Verification Task:

Verify that

1. Kadane’s Algorithm returns a value which is the sum of a contiguous subarray within array a, and
2. the sum of every contiguous subarray within a is not greater than the returned value.

Two-dimensional Case:

For the given two-dimensional array

$$\begin{array}{rrrr}
0 & -2 & -7 & 0 \\
\mathbf{9} & \mathbf{2} & -6 & 2 \\
\mathbf{-4} & \mathbf{1} & -4 & 1 \\
\mathbf{-1} & \mathbf{8} & 0 & -2
\end{array}$$

the maximum-sum submatrix is

$$\begin{array}{rr}
9 & 2 \\
-4 & 1 \\
-1 & 8\ .
\end{array}$$

To get an $O(N^3)$ algorithm, we create $O(N^2)$ one-dimensional subproblems by iterating over all possible contiguous sequences of row indices. We then apply Kadane's algorithm to each 1D subproblem, with the maximum-sum subarray amongst these returning the solution to the 2D problem.

The subproblems are arrays which contain accumulated sums of contiguous rows. Dynamic programming can be used to obtain these efficiently.

In the example above the contiguous sequences of row indices from which sums are formed are:
`[0], [0,1], [0,1,2], [0,1,2, 3], [1], [1,2], [1,2,3], [2], [2,3], [3].`

The arrays holding the accumulated sums are:

[0]	0	-2	-7	0
[0,1]	9	0	-13	2
[0,1,2]	5	1	-17	3
[0,1,2, 3]	4	9	-17	1
[1]	9	2	-6	2
[1,2]	5	3	-10	3
[1,2,3]	**4**	**11**	-10	1
[2]	-4	1	-4	1
[2,3]	-5	9	-4	-1
[3]	-1	8	0	-2

We apply Kadane's algorithm to each of these arrays, keeping the maximum sum found across **all** arrays. For example, above, the maximum sum of 15 is found in row sequence [1,2,3].

The maximum-sum submatrix column bounds (0 and 1 here) are obtained from the bounds of the maximum-sum subarray. The row bounds are obtained from the row sequence yielding the maximum sum (1-3 in the example).

Two-dimensional Verification Task (That is an advanced challenge!)

Implement and verify a function that solves the maximum-sum submatrix problem.
You should verify that

1. the function returns a value which is the sum of a submatrix within the input matrix, and that
2. the sum of every submatrix within the input matrix is not greater than the returned value.