Problem 1. What is the worst-case run-time complexity (Big O) of the following pieces of code?

(a)
```
for i = 1 to m+5 do
    for j = 1 to n do
     // some code (assume it runs in constant time)
```

Solution: O(mn)

(b)
```
for i = 1 to n do
    for j = 1 to 5 do
     // some code ( assume it runs in constant time)
```

Solution: O(n)

(c)
```
foo(n){
    if(n==1) return;
    foo(n-1);
}
```
for $n \geq 1$.

Solution: O(n)

Problem 2. For the maximum contiguous subarray problem, Kadane's algorithm gave us a linear runtime algorithm. Is this the best we can do for this problem or can we do better? Why or why not? Write your answer in one or two sentences.

Solution: Since there are n elements we will need to at least read them once. Kadane's algorithm is a linear time algorithm (proportional to the number of elements) and therefore the best we can do.

Problem 3. In computer vision we are always interested in finding the change of intensity values, usually for edge detection. These changes in intensities are obtained by convolving (or cross correlating) an image with a filter. The convolution results in a correlated image representing the changes in intensity values in the horizontal and vertical directions in the image. These changes in gradients or derivatives may contain positive, negative or zero values.

Suppose you are provided these images of gradients for edge detection. We are interested in finding the region of the brightest edge. Write pseudocode to find the region (rectangular area) of the maximum intensity(gradient) change. For each such region the gradient(or intensity) value is obtained by adding all the values of that subregion. For example, for the following image

the maximum intensity region is given by the subregion

6	-5	-7	4	-4
-9	3	-6	5	2
-10	4	7	-6	3
-8	9	-3	3	-7

4	7
9	-3

Solution: Find the accumulated sums of the contiguous rows forming 1D subarrays. Run Kadane's alagorithm on each one of these 1D subarrays. For example,

```
[1]:         6 -5 -7 4 -4
[1,2]:      -3 -2 -13 9 -2
[1,2,3]:   -13 2 -6 3 1
[1,2,3,4]: -21 11 -9 6 -6
[2]:        -9 3 -6 5 2
[2,3]:     -19 7 1 -1 5
[2,3,4]:   -27 16 -2 2 -2
[3]:       -10 4 7 -6 3
[3,4]:    -18 13 4 -3 -4
```

The last subarray for rows [3,4] and columns [2,3] returns the maximum subregion.

```
Input array:  A
function MSSum(A):
    maxSum = 0
    for i = 1 to n
        S = A[i,]
        for j = i to n
            S = S + A[j,]
            $max\_start\_index, max\_end\_index$,M = Kadane's algorithm(S)
            if(M $>$ maxSum):
                maxSum = M
                $max\_subarray\_start\_row, max\_subarray\_end\_row,$
                    $max\_subarray\_start\_col, max\_subarray, end\_col$ =
                    $i, j, max\_start\_index, max\_end\_index$
    return $max\_subarray\_start\_row, max\_subarray\_end\_row,$
        $max\_subarray\_start\_col, max\_subarray, end\_col$
```