SEMINAR 2

LIST AND 2D LISTS

WEEK 6 – Seminar 2

Exercise 1:

Brute force approach

```
Function maxSumSubarray(numbers:int[]):int
    // We assume the array is not empty
    globalMax = numbers[0]
    for i := 0 to numbers.size()-1 do
        localMax = numbers[i]
        currentSum = numbers[i]
        for j := i+1 to numbers.size()-1 do
            currentSum += numbers[j]
            localMax = max(localMax, currentSum)
            if localMax > globalMax then
                globalMax = localMax
            endif
        endfor
    endfor

    return globalMax
```

This algorithm makes in the order of n^2 comparisons

Kadane's Algorithm

This algorithm was shown during the seminar session. You can find a video of the Kadane's algorithm on YouTube.

```
Function maxSumSubarray(numbers:int[]):int
    // We assume the array is not empty
    localMax = globalMax = numbers[0]
    for i := 1 to numbers.size()-1 do
        localMax = max(numbers[i], numbers[i]+localMax)
        if localMax > globalMax then
            globalMax = localMax
        endif
    endfor

    return globalMax
```

Exercise 2:

Brute force approach

```
Function maxSumSubarray(matrix:int[][]):int
    // We assume the array is not empty
    rows, cols := matrix.size()
    globalMax = matrix[0][0]
    for topRow := 0 to rows-1 do
        for topCol := 0 to cols-1 do
            for bottowRow := topRow to rows-1 do
                for bottomCol := topCol to cols-1 do
                    globalMax = max(globalMax,
                                    sumMatrix(matrix,
                                              topRow,
                                              topCol,
                                              bottomRow,
                                              bottomCol)
                endfor
            endfor
        endfor
    endfor

    return globalMax
```

We also need to write the algorithm for the function `sumMatrix` which computes the sum of a rectangle in a matrix (2D array).

```
Function sumMatrix(matrix:int[][],
                   trow:int,
                   tcol:int,
                   brow:int
                   bcol:int):int
    // For simplicity We assume the inputs are valid
    total = 0
    for row := trow to brow do
        for col := tcol to bcol do
            total += matrix[row][col]
        endfor
    endfor

    return total
```

Currently, the approach is executing the + operation in the order of $n^3 \times m^3$ times, where n is the number of rows and m the number of columns of the 2D array. This approach can be improved by modifying the for loops and adding/initialising some variable(s) where needed to compute the sum of the rectangle within the loops rather that using the function `sumMatrix`. If done properly, the number of + operation should be in the order of $n^2 \times m^2$. Modify the algorithm to do so.