Coding challenges

Mohamed Boucetta

Exercise 0.1 *Build a function which takes a matrix of integers with n rows and m columns and returns a matrix with the same rows but sorted by their sum. For intance if the input is [[0, 1, 2], [1, -1, 0], [7, 8, -9], [0, 3, 1], [0, 2, -2]], the sum of the first row is 3, 0 for the second row, 6 for the third one, 4 the fourth one and 0 for the last one. So the output is [[1, -1, 0], [0, 2, -2], [0, 1, 2], [0, 3, 1], [7, 8, -9]].*

Solution : We illustrate our solution by taking as input

$$M = [[0,1,2],[1,-1,0],[7,8,-9],[0,3,1],[0,2,-2]].$$

1. First we build a function named sumArray which takes an array and return the sum of its items.
2. We create two variables. The first one named sumRowsSorted which is an array with has the same number of items as the number of rows in our input. The second one named sortedMatrix is a matrix containing the same number of rows as our input.

   $$\mathrm{sumRowsSorted} = [0,0,0,0,0] \quad \text{and} \quad \mathrm{sortedMatrix} = [[0],[0],[0],[0],[0]].$$

3. We insert the sums of the rows of our input in sumRowsSorted :

   $$\mathrm{sumRowsSorted} = [3,0,6,4,0].$$

4. We sort sumRowsSorted :

   $$\mathrm{sumRowsSorted} = [0,0,3,4,6].$$

5. We run a for loop with the range from 0 to $n-1$ (where n is the number of rows in our input). For any index i, we compute the sum s_i of the row i, we look in sumRowsSorted for the first index j where s_i appears, we put the row i in the j index of sortedMatrix and we replace sumRowsSorted[j] by infinity.

6. Return sortedMatrix.

   In our example, the for loop goes as follows :

   (a) $i = 0$, $s_0 = 3$. The value 3 appears at $j = 2$ so

   $$\begin{cases} \text{sumRowsSorted} = [0, 0, infinity, 4, 6], \\ \text{sortedMatrix} = [[0], [0], [0, 1, 2], [0], [0]]. \end{cases}$$

   (b) $i = 1$, $s_0 = 0$. The value 0 appears at $j = 0$ so

   $$\begin{cases} \text{sumRowsSorted} = [infinity, 0, infinity, 4, 6], \\ \text{sortedMatrix} = [[1, -1, 0], [0], [0, 1, 2], [0], [0]]. \end{cases}$$

   (c) $i = 2$, $s_0 = 6$. The value 6 appears at $j = 4$ so

   $$\begin{cases} \text{sumRowsSorted} = [infinity, 0, infinity, 4, Infinity], \\ \text{sortedMatrix} = [[1, -1, 0], [0], [0, 1, 2], [0], [7, 8, -9]]. \end{cases}$$

   (d) $i = 3$, $s_0 = 4$. The value 4 appears at $j = 3$ so

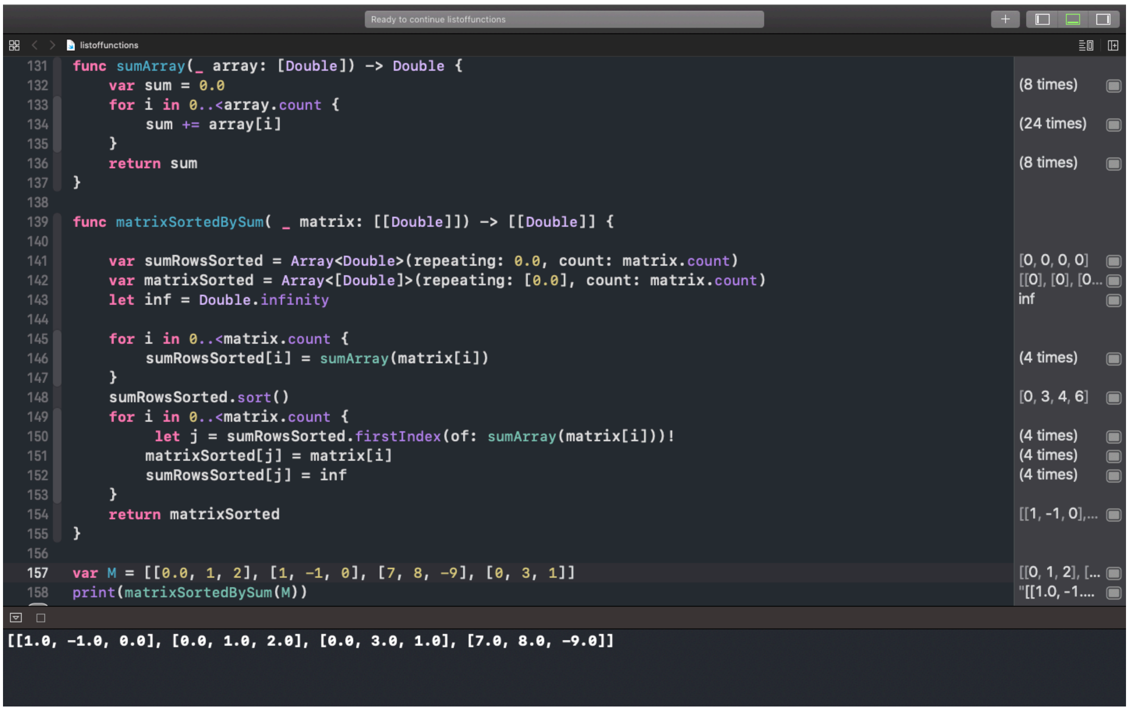
   $$\begin{cases} \text{sumRowsSorted} = [infinity, 0, infinity, infinity, infinity], \\ \text{sortedMatrix} = [[1, -1, 0], [0], [0, 1, 2], [0, 3, 1], [7, 8, -9]]. \end{cases}$$

   (e) $i = 4$, $s_0 = 0$. The value 0 appears at $j = 1$ so

   $$\begin{cases} \text{sumRowsSorted} = [infinity, infinity, infinity, infinity, infinity], \\ \text{sortedMatrix} = [[1, -1, 0], [0, 2, -2], [0, 1, 2], [0, 3, 1], [7, 8, -9]]. \end{cases}$$

The time complexity is $O(NlogN)$: the for loop takes N iterations and in each iteration we look in a sorted array of a value which take $logN$.

Since we create a matrix, the space complexity is $O(pN)$, where N is the number of rows and p the number of columns.



```swift
func sumArray(_ array: [Double]) -> Double {
    var sum = 0.0
    for i in 0..<array.count {
        sum += array[i]
    }
    return sum
}

func matrixSortedBySum( _ matrix: [[Double]]) -> [[Double]] {

    var sumRowsSorted = Array<Double>(repeating: 0.0, count: matrix.count)
    var matrixSorted = Array<[Double]>(repeating: [0.0], count: matrix.count)
    let inf = Double.infinity

    for i in 0..<matrix.count {
        sumRowsSorted[i] = sumArray(matrix[i])
    }
    sumRowsSorted.sort()
    for i in 0..<matrix.count {
         let j = sumRowsSorted.firstIndex(of: sumArray(matrix[i]))!
        matrixSorted[j] = matrix[i]
        sumRowsSorted[j] = inf
    }
    return matrixSorted
}

var M = [[0.0, 1, 2], [1, -1, 0], [7, 8, -9], [0, 3, 1]]
print(matrixSortedBySum(M))
```

```
[[1.0, -1.0, 0.0], [0.0, 1.0, 2.0], [0.0, 3.0, 1.0], [7.0, 8.0, -9.0]]
```