```python
# Copyright © 2020 AlgoExpert, LLC. All rights reserved.

# O(nlog(k) + k) time | O(n + k) space - where where n is the total
# number of array elements and k is the number of arrays
def mergeSortedArrays(arrays):
    sortedList = []
    smallestItems = []
    for arrayIdx in range(len(arrays)):
        smallestItems.append({"arrayIdx": arrayIdx, "elementIdx": 0, "num": arrays[arrayIdx][0]})
    minHeap = MinHeap(smallestItems)
    while not minHeap.isEmpty():
        smallestItem = minHeap.remove()
        arrayIdx, elementIdx, num = smallestItem["arrayIdx"], smallestItem["elementIdx"], smallestItem["num"]
        sortedList.append(num)
        if elementIdx == len(arrays[arrayIdx]) - 1:
            continue
        minHeap.insert(
            {"arrayIdx": arrayIdx, "elementIdx": elementIdx + 1, "num": arrays[arrayIdx][elementIdx + 1]}
        )
    return sortedList


class MinHeap:
    def __init__(self, array):
        self.heap = self.buildHeap(array)

    def isEmpty(self):
        return len(self.heap) == 0

    def buildHeap(self, array):
        firstParentIdx = (len(array) - 2) // 2
        for currentIdx in reversed(range(firstParentIdx + 1)):
            self.siftDown(currentIdx, len(array) - 1, array)
        return array

    def siftDown(self, currentIdx, endIdx, heap):
        childOneIdx = currentIdx * 2 + 1
        while childOneIdx <= endIdx:
            childTwoIdx = currentIdx * 2 + 2 if currentIdx * 2 + 2 <= endIdx else -1
            if childTwoIdx != -1 and heap[childTwoIdx]["num"] < heap[childOneIdx]["num"]:
                idxToSwap = childTwoIdx
            else:
                idxToSwap = childOneIdx
            if heap[idxToSwap]["num"] < heap[currentIdx]["num"]:
                self.swap(currentIdx, idxToSwap, heap)
                currentIdx = idxToSwap
                childOneIdx = currentIdx * 2 + 1
            else:
                return

    def siftUp(self, currentIdx, heap):
        parentIdx = (currentIdx - 1) // 2
        while currentIdx > 0 and heap[currentIdx]["num"] < heap[parentIdx]["num"]:
            self.swap(currentIdx, parentIdx, heap)
            currentIdx = parentIdx
            parentIdx = (currentIdx - 1) // 2

    def remove(self):
        self.swap(0, len(self.heap) - 1, self.heap)
        valueToRemove = self.heap.pop()
        self.siftDown(0, len(self.heap) - 1, self.heap)
        return valueToRemove

    def insert(self, value):
        self.heap.append(value)
        self.siftUp(len(self.heap) - 1, self.heap)

    def swap(self, i, j, heap):
        heap[i], heap[j] = heap[j], heap[i]
```