```
1  // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3  using System.Collections.Generic;
4
5  public class Program {
6    // O(nlog(k) + k) time | O(n + k) space - where where n is the total
7    // number of array elements and k is the number of arrays
8    public static List<int> MergeSortedArrays(List<List<int> > arrays) {
9      List<int> sortedList = new List<int>();
10     List<Item> smallestItems = new List<Item>();
11
12     for (int arrayIdx = 0; arrayIdx < arrays.Count; arrayIdx++) {
13       smallestItems.Add(new Item(arrayIdx, 0, arrays[arrayIdx][0]));
14     }
15
16     MinHeap minHeap = new MinHeap(smallestItems);
17     while (!minHeap.isEmpty()) {
18       Item smallestItem = minHeap.Remove();
19       sortedList.Add(smallestItem.num);
20       if (smallestItem.elementIdx ==
21         arrays[smallestItem.arrayIdx].Count - 1) continue;
22       minHeap.Insert(new Item(
23           smallestItem.arrayIdx,
24           smallestItem.elementIdx + 1,
25           arrays[smallestItem.arrayIdx][smallestItem.elementIdx + 1]
26           ));
27     }
28
29     return sortedList;
30   }
31
32   public class Item {
33     public int arrayIdx;
34     public int elementIdx;
35     public int num;
36
37     public Item(int arrayIdx, int elementIdx, int num) {
38       this.arrayIdx = arrayIdx;
39       this.elementIdx = elementIdx;
40       this.num = num;
41     }
42   }
43
44   public class MinHeap {
45     List<Item> heap = new List<Item>();
46
47     public MinHeap(List<Item> array) {
48       heap = buildHeap(array);
49     }
50
51     public bool isEmpty() {
52       return heap.Count == 0;
53     }
54
55     public List<Item> buildHeap(List<Item> array) {
56       int firstParentIdx = (array.Count - 2) / 2;
57       for (int currentIdx = firstParentIdx; currentIdx >= 0; currentIdx--) {
58         siftDown(currentIdx, array.Count - 1, array);
59       }
60       return array;
61     }
62
63     public void siftDown(int currentIdx, int endIdx, List<Item> heap) {
64       int childOneIdx = currentIdx * 2 + 1;
65       while (childOneIdx <= endIdx) {
66         int childTwoIdx = currentIdx * 2 + 2 <=
67           endIdx ? currentIdx * 2 + 2 : -1;
68         int idxToSwap;
69         if (childTwoIdx != -1 &&
70           heap[childTwoIdx].num < heap[childOneIdx].num) {
71           idxToSwap = childTwoIdx;
72         } else {
73           idxToSwap = childOneIdx;
74         }
75         if (heap[idxToSwap].num < heap[currentIdx].num) {
76           swap(currentIdx, idxToSwap, heap);
77           currentIdx = idxToSwap;
78           childOneIdx = currentIdx * 2 + 1;
79         } else {
80           return;
81         }
82       }
83     }
```

```
84
85      public void siftUp(int currentIdx, List<Item> heap) {
86        int parentIdx = (currentIdx - 1) / 2;
87        while (currentIdx > 0 && heap[currentIdx].num < heap[parentIdx].num) {
88          swap(currentIdx, parentIdx, heap);
89          currentIdx = parentIdx;
90          parentIdx = (currentIdx - 1) / 2;
91        }
92      }
93
94      public Item Remove() {
95        swap(0, heap.Count - 1, heap);
96        Item valueToRemove = heap[heap.Count - 1];
97        heap.RemoveAt(heap.Count - 1);
98        siftDown(0, heap.Count - 1, heap);
99        return valueToRemove;
100     }
101
102     public void Insert(Item value) {
103       heap.Add(value);
104       siftUp(heap.Count - 1, heap);
105     }
106
107     public void swap(int i, int j, List<Item> heap) {
108       Item temp = heap[j];
109       heap[j] =  heap[i];
110       heap[i] =  temp;
111     }
112   }
113 }
114
```