```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {
  // 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
  public static List<Integer> mergeSortedArrays(List<List<Integer>> arrays) {
    List<Integer> sortedList = new ArrayList<Integer>();
    List<Item> smallestItems = new ArrayList<Item>();

    for (int arrayIdx = 0; arrayIdx < arrays.size(); arrayIdx++) {
      smallestItems.add(new Item(arrayIdx, 0, arrays.get(arrayIdx).get(0)));
    }

    MinHeap minHeap  = new MinHeap(smallestItems);
    while (!minHeap.isEmpty()) {
      Item smallestItem = minHeap.remove();
      sortedList.add(smallestItem.num);
      if (smallestItem.elementIdx == arrays.get(smallestItem.arrayIdx).size() - 1) continue;
      minHeap.insert(new Item(
        smallestItem.arrayIdx,
        smallestItem.elementIdx + 1,
        arrays.get(smallestItem.arrayIdx).get(smallestItem.elementIdx + 1)
      ));
    }

    return sortedList;
  }

  static class Item {
    public int arrayIdx;
    public int elementIdx;
    public int num;

    public Item(int arrayIdx, int elementIdx, int num) {
      this.arrayIdx = arrayIdx;
      this.elementIdx = elementIdx;
      this.num = num;
    }
  }

  static class MinHeap {
    List<Item> heap = new ArrayList<Item>();

    public MinHeap(List<Item> array) {
      heap = buildHeap(array);
    }

    public boolean isEmpty() {
      return heap.size() == 0;
    }

    public List<Item> buildHeap(List<Item> array) {
      int firstParentIdx = (array.size() - 2) / 2;
      for (int currentIdx = firstParentIdx; currentIdx >= 0; currentIdx--) {
        siftDown(currentIdx, array.size() - 1, array);
      }
      return array;
    }

    public void siftDown(int currentIdx, int endIdx, List<Item> heap) {
      int childOneIdx = currentIdx * 2 + 1;
      while (childOneIdx <= endIdx) {
        int childTwoIdx = currentIdx * 2 + 2 <= endIdx ? currentIdx * 2 + 2 : -1;
        int idxToSwap;
        if (childTwoIdx != -1 && heap.get(childTwoIdx).num < heap.get(childOneIdx).num) {
          idxToSwap = childTwoIdx;
        } else {
          idxToSwap = childOneIdx;
        }
        if (heap.get(idxToSwap).num < heap.get(currentIdx).num) {
          swap(currentIdx, idxToSwap, heap);
          currentIdx = idxToSwap;
          childOneIdx = currentIdx * 2 + 1;
        } else {
          return;
        }
      }
    }

    public void siftUp(int currentIdx, List<Item> heap) {
      int parentIdx = (currentIdx - 1) / 2;
```

```
      while (currentIdx > 0 && heap.get(currentIdx).num < heap.get(parentIdx).num) {
        swap(currentIdx, parentIdx, heap);
        currentIdx = parentIdx;
        parentIdx = (currentIdx - 1) / 2;
      }
    }

    public Item remove() {
      swap(0, heap.size() - 1, heap);
      Item valueToRemove = heap.get(heap.size() - 1);
      heap.remove(heap.size() - 1);
      siftDown(0, heap.size() - 1, heap);
      return valueToRemove;
    }

    public void insert(Item value) {
      heap.add(value);
      siftUp(heap.size() - 1, heap);
    }

    public void swap(int i, int j, List<Item> heap) {
      Item temp = heap.get(j);
      heap.set(j, heap.get(i));
      heap.set(i, temp);
    }
  }
}
```