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

using namespace std;

struct Item {
  int arrayIdx;
  int elementIdx;
  int num;
};

class MinHeap {
  public:
  vector<Item> heap;

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

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

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

  void siftDown(int currentIdx, int endIdx, vector<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[childTwoIdx].num < heap[childOneIdx].num) {
        idxToSwap = childTwoIdx;
      } else {
        idxToSwap = childOneIdx;
      }
      if (heap[idxToSwap].num < heap[currentIdx].num) {
        swap(currentIdx, idxToSwap, heap);
        currentIdx = idxToSwap;
        childOneIdx = currentIdx * 2 + 1;
      } else {
        return;
      }
    }
  }

  void siftUp(int currentIdx, vector<Item>& heap) {
    int parentIdx = (currentIdx - 1) / 2;
    while (currentIdx > 0 && heap[currentIdx].num < heap[parentIdx].num) {
      swap(currentIdx, parentIdx, heap);
      currentIdx = parentIdx;
      parentIdx = (currentIdx - 1) / 2;
    }
  }

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

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

  void swap(int i, int j, vector<Item>& heap) {
    Item temp = heap[j];
    heap[j] = heap[i];
    heap[i] = temp;
  }
};

// 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
vector<int> mergeSortedArrays(vector<vector<int>> arrays) {
  vector<int> sortedList;
```

```
    vector<Item> smallestItems;

    for (int arrayIdx = 0; arrayIdx < arrays.size(); arrayIdx++) {
      smallestItems.push_back(Item {
        arrayIdx,
        0,
        arrays[arrayIdx][0],
      });
    }

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

    return sortedList;
  }

```