```
// Create a node class
// this will contain the nodes vale and which array it belongs to
class HeapNode{
    constructor(value, origin){
        this.value = value;
        this.origin = origin;
    }
}

// Create a Min Heap Class
class MinHeap{
    constructor(A){
        this.arrayLength = A.length;
        this.heap = A;
        this.heapSize = A.length;
        this.BuildMinHeap(this.heap);
    }
    // Recursively Heapith a Vlaue "bubble down"
    MinHeapify(i){
        let l = 2*i+1;
        let r = 2*i+2;
        let smallest = i;// Index of smallest element
        if(l < this.heapSize && this.heap[l].value < this.heap[smallest].value){
            smallest = l;
        }
        if(r < this.heapSize && this.heap[r].value < this.heap[smallest].value){
            smallest = r;
        }
        if(smallest != i){// swap A[i] and A[smallest]
            let tmp = this.heap[i];
            this.heap[i] = this.heap[smallest];
            this.heap[smallest] = tmp;
            this.MinHeapify(i)
        }
    }

    // Build Min Heap from a sorted array
    BuildMinHeap(){
        let startIdx = Math.floor(this.arrayLength/2) - 1;
        for (let i = startIdx; i >= 0; i--){
            this.MinHeapify(i);
        }
    }

    // Add a Heapsort Method
    HeapSort(){
        let last = this.arrayLength;
        for(let i=last-1; i>=0; i --){
            let tmp = this.heap[0];
            this.heap[0] = this.heap[i];
            this.heap[i] = tmp;
            this.heapSize = i;
            this.MinHeapify(0);
        }
        this.heapSize = this.arrayLength;// Restore this property
        this.heap.reverse();
    }

    // Add ability to replace the root with an outside node
    ReplaceRoot(node){// returns the previous root node
        // Get the in element (the root)
        let root = this.heap[0];

        // Replace the root with the last element
        this.heap[0] = node;

        // Re-Heapify the root node
        this.MinHeapify(0);

        return root;
    }

    // Add ability to return roots value and delete it
    PopRoot(){
        let root = this.heap[0];
        if(this.heap.length >= 1){// There needs to be atleast one elemnt in the heap
            // Get last element
            let lastElement = this.heap[this.heap.length - 1];

            // Replace the root with the last element, return the old root
            this.ReplaceRoot(lastElement);
```

```
            // Decrese size of the heap
            this.heap.length--;
            this.heapSize = this.heap.length;

            // Re-Heapify the root node
            this.MinHeapify(0);
        }
    }
}

// Lets generate some input values:
// k = number of lists
// l = length of each list
function generateSortedLists(k, l){
    let lists = [];
    for(let i = 0; i<k; i++){// Populate list of lists
        let list = [];
        let value = 0;
        for(let j=0; j<l; j++){// Populate individual list with values
            value += Math.ceil(Math.random()*10) + 1;// Random sorted number between 0 and 11*n
            let tmpNode = new HeapNode(value, i);
            list.push(tmpNode);// Add the node to the this list
        }
        lists.push(list);
    }
    return lists;
}

function RemoveEmptyLists(lists){
    let origin = 0;
    lists.forEach((list) => {
        if(list.length == 0){
            let dummyNode = new HeapNode(Infinity, origin);
            list.push(dummyNode);
        }
        origin ++;
    });
}

function FindMin(lists){
    let target = lists[0][0];
    lists.forEach((list) =>{
        target = list[0].value < target.value ? list[0] : target;
    });
    return target;
}

// Problem: Merge k sorted lists into one sorted array of size n (total number of elements in all input arrays)
// Note: We know the set cotaining the 0th value of each sorted array must contain the minimum
// we also know that the next value in any array must be less than the previous
// Approach: A priority queue could exploit these properties
function MergeKSortedArrays(lists){
    // For simplicity assume lists are all of even length
    let totalElements = lists.length*lists[0].length;
    let h_arr = [];
    let results = [];
    lists.forEach(list => h_arr.push(list.shift()));

    // Heapify this result
    minHeap = new MinHeap(h_arr);
    console.log(minHeap);

    while(results.length < totalElements){
        let root = minHeap.heap[0];
        let value = root.value;
        let origin = root.origin;

        // Add smallest value from our min heap (priority queue) to our results array
        results.push(value);
        console.log(minHeap);
        console.log("results: ", results);

        // Replace the heaps root with the next value from its originating array
        // which is the next largest val wrt origin array
        if(lists[origin].length > 0){
            minHeap.ReplaceRoot(lists[origin].shift());
        }
        else{// Our next min must be contained in the current heap
            minHeap.PopRoot();
            console.log("PopRoot");
        }
    }
```

```
    return results;
}


// Create the sorted list
// Note: A sorted list is already a minHeap!!!!!!
var sortedLists = generateSortedLists(4, 5);
console.log(sortedLists);

let results = MergeKSortedArrays(sortedLists);
console.log(results);
console.log("results.length = ", results.length);
```