

Prompt	Scratchpad	Our Solution(s)	Video Explanation	Run Code
--------	------------	-----------------	-------------------	----------

Solution 1Solution 2

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 package main
4
5 ▾ type Item struct {
6     ArrayIdx int
7     ElementIdx int
8     Num int
9 }
10
11 // O(nlog(k) + k) time | O(n + k) space - where where n is the total
12 // number of array elements and k is the number of arrays
13 ▾ func MergeSortedArrays(arrays [][]int) []int {
14     sortedList := []int{}
15     smallestItems := []Item{}
16
17     for arrayIdx := 0; arrayIdx < len(arrays); arrayIdx++ {
18         ▾ smallestItems = append(smallestItems, Item{
19             ArrayIdx: arrayIdx,
20             ElementIdx: 0,
21             Num: arrays[arrayIdx][0],
22         })
23     }
24
25     mh := NewMinHeap(smallestItems)
26     ▾ for mh.length() != 0 {
27         smallestItem := mh.Remove()
28         sortedList = append(sortedList, smallestItem.Num)
29         ▾ if smallestItem.ElementIdx == len(arrays[smallestItem.ArrayIdx])-1 {
30             continue
31         }
32         ▾ mh.Insert(Item{
33             ArrayIdx: smallestItem.ArrayIdx,
34             ElementIdx: smallestItem.ElementIdx + 1,
35             Num: arrays[smallestItem.ArrayIdx][smallestItem.ElementIdx+1],
36         })
37     }
38     return sortedList
39 }
40
41 type MinHeap []Item
42
43 ▾ func NewMinHeap(array []Item) *MinHeap {
44     heap := MinHeap(array)
45     ptr := &heap
46     ptr.BuildHeap(array)
47     return ptr
48 }
49
50 // O(n) time | O(1) space
51 ▾ func (h *MinHeap) BuildHeap(array []Item) {
52     first := (len(array) - 2) / 2
53     ▾ for currentIndex := first + 1; currentIndex >= 0; currentIndex-- {
54         h.siftDown(currentIndex, len(array)-1)
55     }
56 }
57
58 // O(log(n)) time | O(1) space
59 ▾ func (h *MinHeap) siftDown(currentIndex, endIndex int) {
60     childOneIdx := currentIndex*2 + 1
61     ▾ for childOneIdx <= endIndex {
62         childTwoIdx := -1
63         ▾ if currentIndex*2+2 <= endIndex {
64             childTwoIdx = currentIndex*2 + 2
65         }
66         indexToSwap := childOneIdx
67         ▾ if childTwoIdx > -1 && (*h)[childTwoIdx].Num < (*h)[childOneIdx].Num {
68             indexToSwap = childTwoIdx
69         }
70         ▾ if (*h)[indexToSwap].Num < (*h)[currentIndex].Num {
71             h.swap(currentIndex, indexToSwap)
72             currentIndex = indexToSwap
73             childOneIdx = currentIndex*2 + 1
74         } else {
75             return
76         }
77     }
78 }
79
80 // O(log(n)) time | O(1) space
81 ▾ func (h *MinHeap) siftUp() {
82     currentIndex := h.length() - 1
83     parentIndex := (currentIndex - 1) / 2
```

```
84 ▾   for currentIndex > 0 {
85       current, parent := (*h)[currentIndex].Num, (*h)[parentIndex].Num
86 ▾   if current < parent {
87       h.swap(currentIndex, parentIndex)
88       currentIndex = parentIndex
89       parentIndex = (currentIndex - 1) / 2
90 ▾   } else {
91       return
92   }
93 }
94 }
95
96 // O(log(n)) time | O(1) space
97 ▾ func (h *MinHeap) Remove() Item {
98     l := h.length()
99     h.swap(0, l-1)
100     peeked := (*h)[l-1]
101     *h = (*h)[0 : l-1]
102     h.siftDown(0, l-2)
103     return peeked
104 }
105
106 // O(log(n)) time | O(1) space
107 ▾ func (h *MinHeap) Insert(value Item) {
108     *h = append(*h, value)
109     h.siftUp()
110 }
111
112 ▾ func (h MinHeap) swap(i, j int) {
113     h[i], h[j] = h[j], h[i]
114 }
115
116 ▾ func (h MinHeap) length() int {
117     return len(h)
118 }
119
```