**Sort a k-sorted array**

Given a k–sorted array that is almost sorted such that each of the n elements may be misplaced by no more than k positions from the correct sorted order. Find a space-and-time efficient algorithm to sort the array.

For example,

**Input:**  
   
arr = [1, 4, 5, 2, 3, 7, 8, 6, 10, 9]  
k = 2  
   
**Output:**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[Practice This Problem](https://www.techiedelight.com/?problem=SortKSortedArray)

A simple solution would be to use an [**efficient sorting algorithm**](https://www.techiedelight.com/quicksort/) to sort the whole array again. The worst-case time complexity of this approach will be O(n.log(n)), where n is the size of the input. This method also does not use the fact that the array is k–sorted. We can also use the [**insertion sort algorithm**](https://www.techiedelight.com/insertion-sort-iterative-recursive/) to correct the order in just O(n.k) time. Insertion sort performs really well for small values of k, but it’s not recommended for a large value of k (we can use it for k < 12).

We can solve this problem in O(n.log(k)) using a [**min-heap**](https://www.techiedelight.com/introduction-priority-queues-using-binary-heaps/#Heap). The idea is to construct a min-heap of size k+1 and insert the first k+1 elements into the heap. Then remove minimum from the heap and insert the next element from the array into the heap and continue the process till both array and heap are exhausted. Each pop operation from the heap should insert the corresponding top element in its correct position into the array.

The time complexity of the above solution is O(n.log(k)) since each insertion operation takes O(log(k)) time, and there are n elements in the input. The additional space used by the program is O(k).