Lec-42

25/12/24 4:05 PM

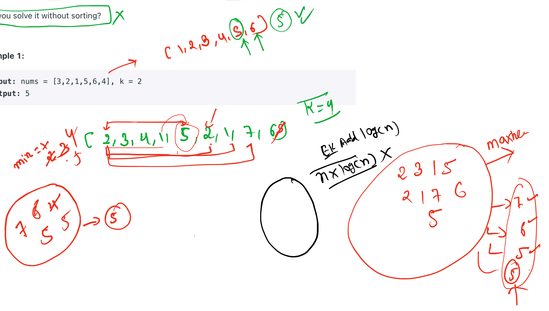
Given an integer array `nums` and an integer `k`, return *the* k^{th} *largest element in the array.*

Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

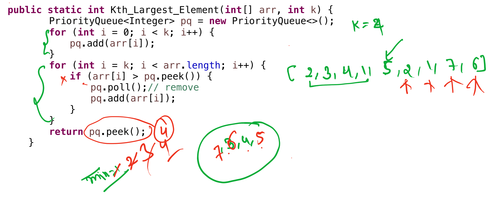
Can you solve it without sorting?

Example 1:

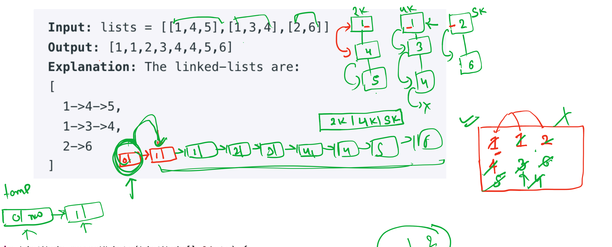
```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```



```
public static int Kth_Largest_Element(int[] arr, int k) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for (int i = 0; i < k; i++) {
            pq.add(arr[i]);
        }
        for (int i = k; i < arr.length; i++) {
            if (arr[i] > pq.peek()) {
                pq.poll();// remove
                pq.add(arr[i]);
            }
        }
        return pq.peek();
    }
```

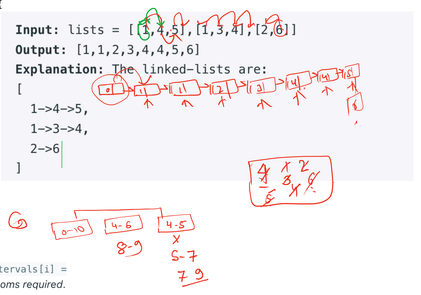


```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
```

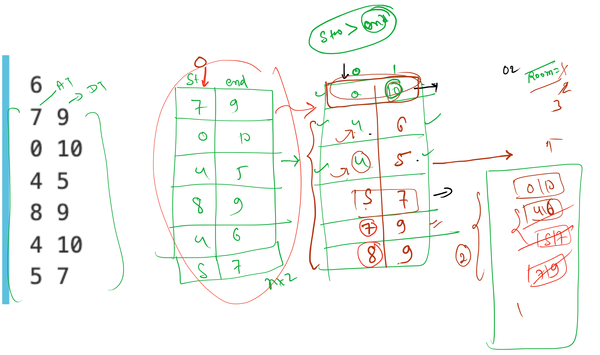


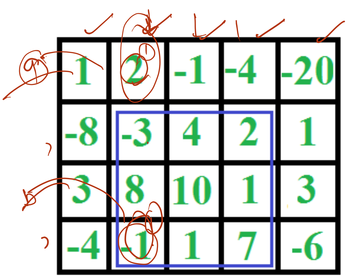
```
public ListNode mergeKLists(ListNode[] lists) {
        PriorityQueue<ListNode> pq = new PriorityQueue<>();
        ListNode Dummy = new ListNode();
        ListNode temp = Dummy;
        for (int i = 0; i < lists.length; i++) {
            if (lists[i] != null) {
                pq.add(lists[i]);
            }
        }

        while (!pq.isEmpty()) {
            ListNode rv = pq.poll();
            Dummy.next = rv;
            Dummy = Dummy.next;
            if (rv.next != null) {
                pq.add(rv.next);
            }
        }
```



Given an array of meeting time intervals `intervals` where `intervals[i] = [start`$_i$`, end`$_i$`]`, return *the minimum number of conference rooms required.*





```
int[][] intervals = { { 7, 9 }, { 0, 10 }, { 4, 5 }, { 8, 9 }, { 4, 6 }, { 5, 7 } };

public static int minrooms(int[][] intervals) {
        Arrays.sort(intervals, (p, q) -> p[0] - q[0]);
        PriorityQueue<int[]> pq = new PriorityQueue<>((p, q) -> p[1] - q[1]);
        int room = 1;
        pq.add(intervals[0]);
        for (int i = 1; i < intervals.length; i++) {
            if (intervals[i][0] >=pq.peek()[1]) {
                pq.poll();
                pq.add(intervals[i]);

            } else {
                pq.add(intervals[i]);
                room++;
            }
        }
        return room;
    }
```