Sort K sorted array

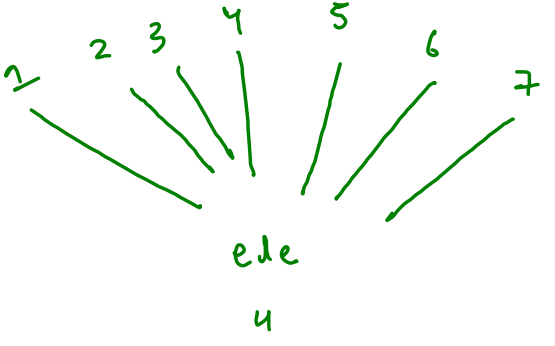
input : 2 3 1 4 7 5 8 6

sorted : 1 2 3 4 5 6 7 8

-1 2 -1 0 1 -2 2 -1

T : n log k

S : k



k = 3

(k = 3)

input :

3	2	4	1	6	5	7	9	8
0	1	2	3	4	5	6	7	8

$k = 3$

$\boxed{\cancel{7} \quad \cancel{8} \quad \cancel{9} \quad \cancel{6}}$

```java
public static void solve(int[]arr,int k) {
    PriorityQueue<Integer>pq = new PriorityQueue<>();

    //push starting k+1 elements in pq
    for(int i=0; i < k+1;i++) {
        pq.add(arr[i]);
    }

    //travel the rest of elements
    for(int i=k+1;i < arr.length;i++) {
        System.out.println(pq.remove());
        pq.add(arr[i]);
    }

    while(pq.size() > 0) {
        System.out.println(pq.remove());
    }

}
```

$k \log k$

$(n-k) \times \log k$

$k \log k$

1 2 3 4 5 6 7 8 9

$n - k \approx n$

T : $n \log k$ S : k

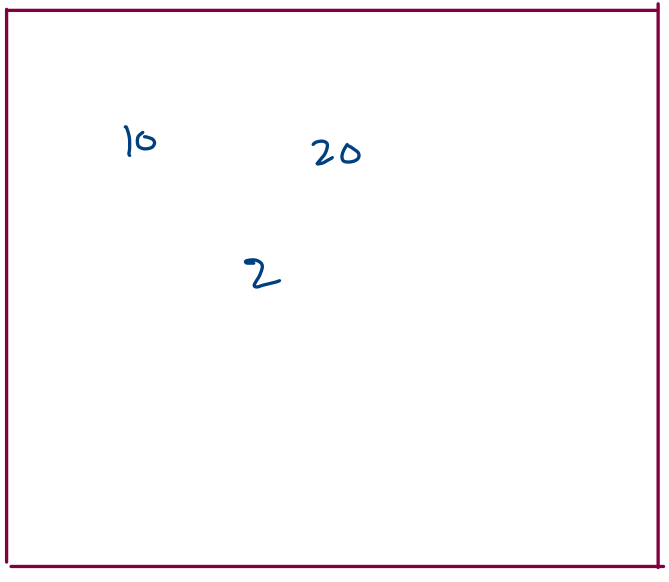
median

pq

add, remove → log n

peek() → O(1)

MPQ




```
public void add(int val) {
  // write your code here
}

public int remove() {
  // write your code here
}

public int peek() {
  // write your code here
}

public int size() {
  // write your code here
}
```

→ median

mpq.add(10)

mpq.add(20)

mpq.add(5)

mpq.peek() → 10

↳ O(1)

mpq.add(2);

mpq.peek() → 5

mpq.remove() → 5

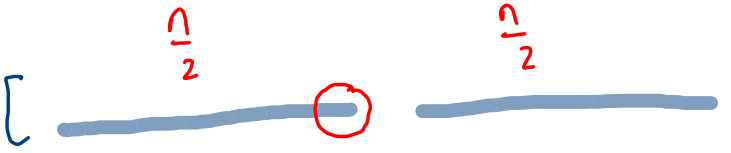
5, 9, 6, 2, 15, 10, ..., 12 ...., 4 ...., 13 20

2 4 5 6 9 | 10 12 13 15 20

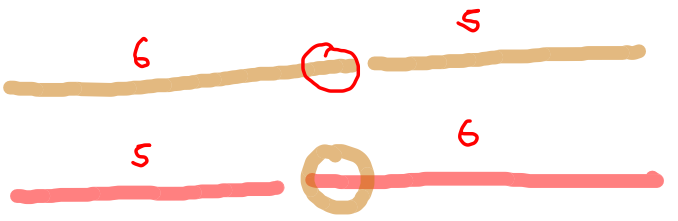
left (max) | right (min)

k	k+1
k+1	k
k	k

n is even



n is odd



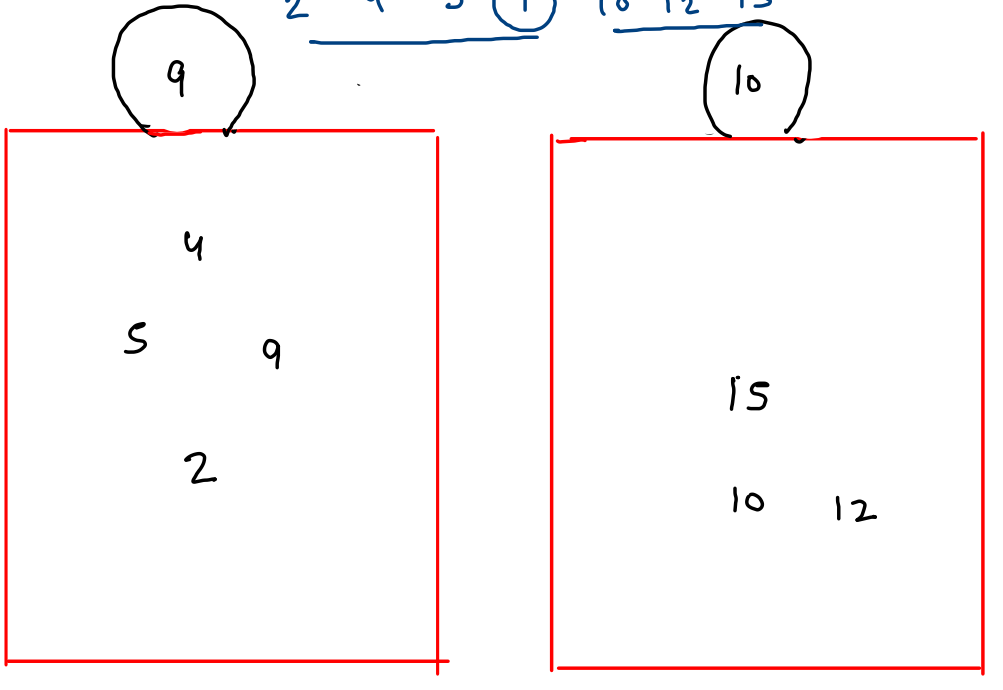
```java
public void add(int val) {
    if(left.size() == 0) {
        right.add(val);
    }
    else if(left.peek() < val) {
        right.add(val);
    }
    else {
        left.add(val);
    }


    if(left.size() - right.size() == 2) {
        right.add(left.remove());
    }
    else if(right.size() - left.size() == 2) {
        left.add(right.remove());
    }
}
```

```java
public int remove() {
    if(size() == 0) {
        System.out.println("Underflow");
        return -1;
    }

    if(left.size() >= right.size()) {
        return left.remove();
    }
    else {
        return right.remove();
    }
}
```

```java
public int peek() {
    if(size() == 0) {
        System.out.println("Underflow");
        return -1;
    }

    if(left.size() >= right.size()) {
        return left.peek();
    }
    else {
        return right.peek();
    }
}
```

15 , 9 , ~~6~~ , 2 , 5 , 10 , ... , 12 .... , 4 .... , 13 20

2 4 5 9 10 12 15

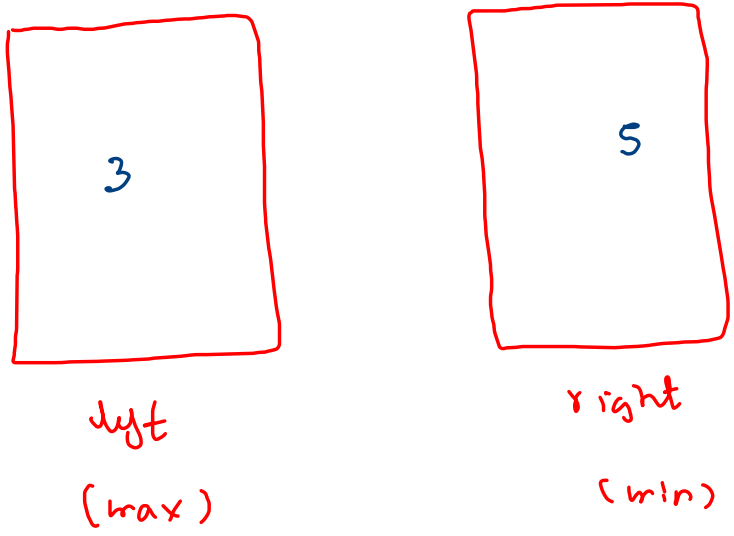


left (max)

right (min)

```java
public void add(int val) {
    if(left.size() == 0) {
        right.add(val);
    }
    else if(left.peek() < val) {
        right.add(val);
    }
    else {
        left.add(val);
    }


    if(left.size() - right.size() == 2) {
        right.add(left.remove());
    }
    else if(right.size() - left.size() == 2) {
        left.add(right.remove());
    }
}
```

2 1 3 5



mpq.add(2)

mpq.add(1)

mpq.add(3)

mpq.remove()

mpq.remove()

mpq.add()

mpq.peek()

$k = 3$

list : [[2, 9, 11], [1, 3, 5, 7], [6, 8, 13, 15]]

$n \rightarrow$ avg. length

total ele $= nk$

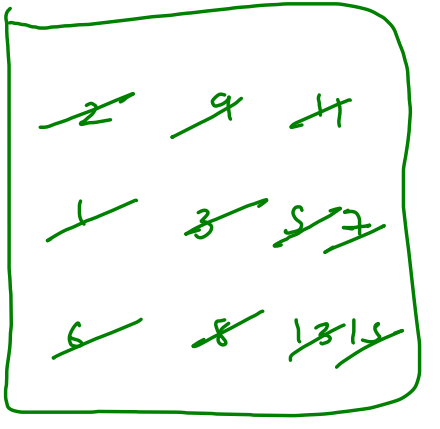
$\boxed{nk \log(nk)}$

S: nk

l_0 : 2 9 11

l_1 : 1 3 5 7

l_2 : 6 8 13 15



ans: 1 2 3 5 6 7 8 9 11 13 15

l_0 : 2_0 9_1 11_2

l_1 : 1_0 3_1 5_2 7_3

l_2 : 6_0 8_1 13_2 15_3

Pair {
int val;
int li; → list idx
int di; → data idx
}

PQ (min)

~~(2 - 0 - 0)~~ ~~(1 - 1 - 0)~~

~~(6 - 2 - 0)~~ ~~(3 - 1 - 1)~~

~~(9 - 0 - 1)~~ ~~(5 - 1 - 2)~~

~~(7 - 1 - 3)~~ ~~(8 - 2 - 1)~~

~~(15 - 2 - 3)~~

~~(13 - 2 - 2)~~ ~~(11 - 0 - 2)~~

ans: 1 2 3 5 6 7 8 9 11 13 15

```java
public static ArrayList<Integer> mergeKSortedLists(ArrayList<ArrayList<Integer>> lists) {
  ArrayList<Integer> ans = new ArrayList<>();

  PriorityQueue<Pair>pq = new PriorityQueue<>(); //smaller value has higher priority

  for(int i=0; i < lists.size();i++) {
      int li = i;
      int di = 0;
      int val = lists.get(i).get(di);

      pq.add(new Pair(val,li,di));
  }


  while(pq.size() > 0) {
      Pair rem = pq.remove();

      ans.add(rem.val);

      int nli = rem.li;
      int ndi = rem.di + 1;

      if(ndi < lists.get(nli).size()) {
          int val = lists.get(nli).get(ndi);
          pq.add(new Pair(val,nli,ndi));
      }
  }

  return ans;
}
```

nk × log k

Pair {

(val, li, di);

}

l_0 : 2_0 9_2 11_2 .

l_1 : 1_0 3_1 5_2 7_3 .

l_2 : 6_0 8_1 13_2 15_3 .

PQ (min)

~~(2 - 0 - 0)~~ ~~(6 - 2 - 0)~~

~~(1 - 1 - 0)~~ ~~(9 - 0 - 1)~~

~~(3 - 1 - 1)~~ ~~(5 - 1 - 2)~~

~~(7 - 1 - 3)~~ ~~(8 - 2 - 1)~~

~~(15 - 2 - 3)~~

~~(13 - 2 - 2)~~ ~~(11 - 0 - 2)~~

ans: 1 2 3 5 6 7 8 9 11 13 15

```java
public ListNode mergeKLists(ListNode[] lists) {

}
```

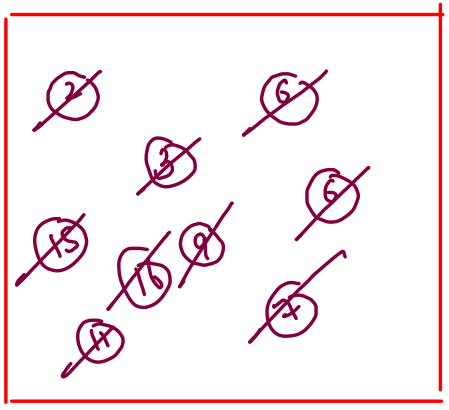
```java
* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode() {}
*     ListNode(int val) { this.val = val; }
*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
```

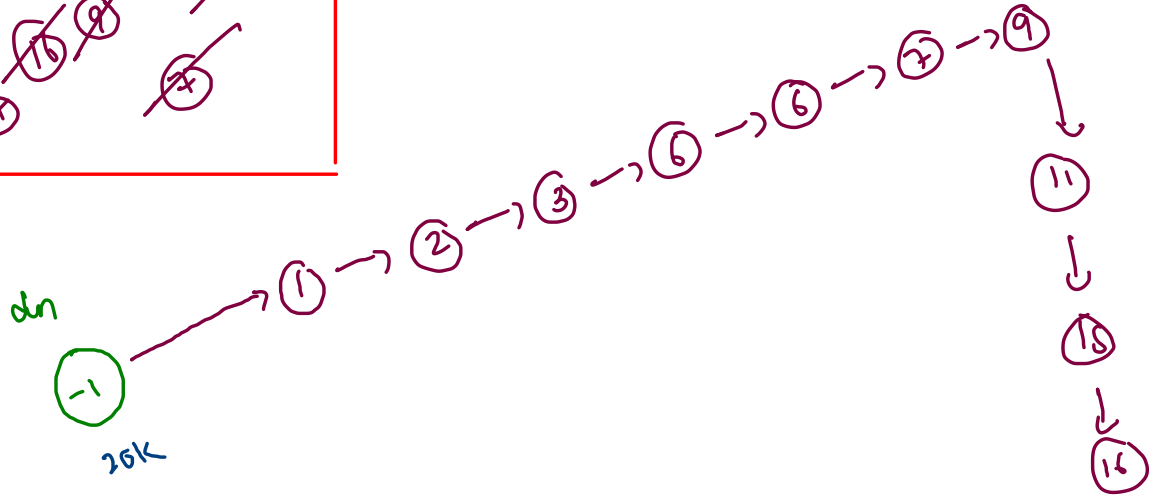
PQ < ListNode > pq ;

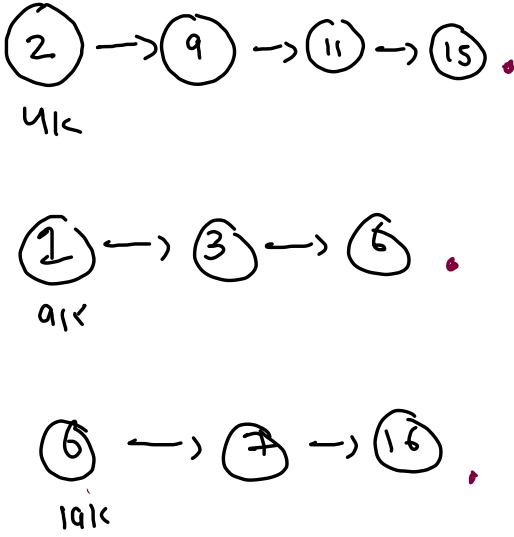
k = 3

[4k , 9k , 19k]

0 1 2





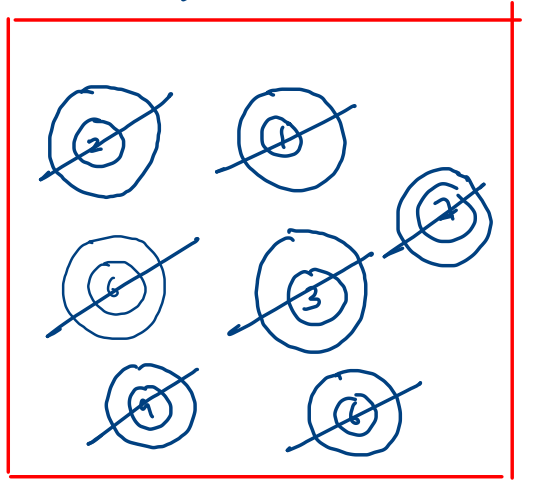


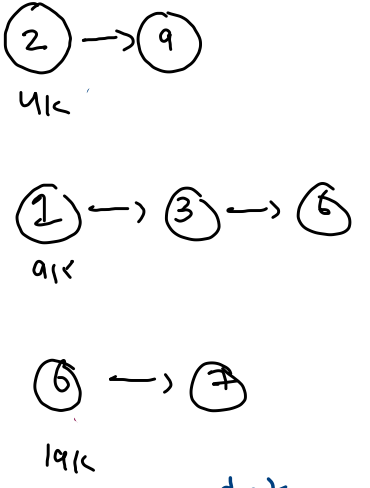
```java
public static class Helper implements Comparable<Helper>{
    ListNode node;

    Helper() {

    }

    Helper(ListNode node) {
        this.node = node;
    }

    public int compareTo(Helper o) {
        return this.node.val - o.node.val;
    }
}
```

```java
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
```

[4k, 9k, 19k]

pq <helper>






```java
public ListNode mergeKLists(ListNode[] lists) {
    PriorityQueue<Helper>pq = new PriorityQueue<>();
    ListNode dnh = new ListNode(-1);
    ListNode dnt = dnh;

    for(int i=0; i < lists.length;i++) {
        ListNode node = lists[i];
        if(node != null) {
            pq.add(new Helper(node));
        }
    }

    while(pq.size() > 0) {
        Helper rem = pq.remove();

        //add last
        ListNode nn = new ListNode(rem.node.val);
        dnt.next = nn;
        dnt = nn;

        if(rem.node.next != null ){
            pq.add(new Helper(rem.node.next));
        }
    }

    return dnh.next;

}
```