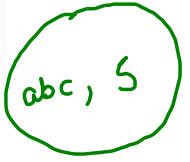
Priority queue

```java
public static class Student implements Comparable<Student>{
    String name;
    int rn;

    Student() {

    }

    Student(String name,int rn) {
        this.name = name;
        this.rn = rn;
    }

    //this < o -> -ve
    //this > o -> +ve
    //this == o -> 0

    public int compareTo(Student o) {
        return this.rn - o.rn;
    }
}
```

rule

```java
PriorityQueue<Student>pq = new PriorityQueue<>(); //smaller value has higher priority

pq.add(new Student("afk",5));
pq.add(new Student("bks",4));
pq.add(new Student("amn",1));
pq.add(new Student("sme",3));
pq.add(new Student("kfc",2));

while(pq.size() > 0) {
    Student st = pq.peek();
    pq.remove();
    System.out.println(st.rn + " " + st.name);
}
```



S1

S2

S1.compareTo(S2)

this = S1

0 = S2

add, remove → log n

peek, size → o(1)

n → no. of elements in pq.

K Largest Elements

arr : 20 12 30 6 5 18 7 19 14

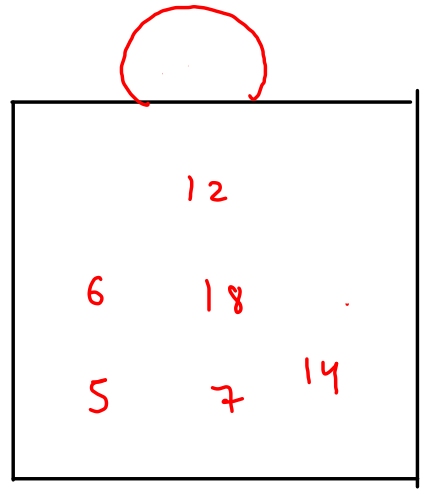
```
for(int ele : arr) {
    pq.add(ele);
}

int[]ans = new int[k];
int i = k-1;

while(k-- > 0) {
    ans[i] = pq.remove();
    i--;
}

for(int val : ans) {
    System.out.println(val);
}
```

nlogn

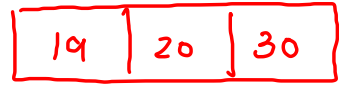
klogn

k

maxpq



ans



i

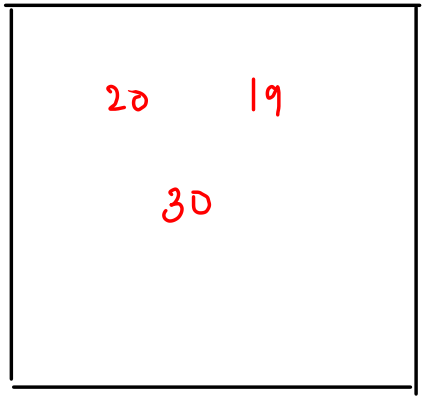
T : nlogn + klogn

S : n+k

arr : 20 6 30 12 5 18 7 19 14

T : nlogk

S : k



minPQ

Smaller value
has ↑ priority

At most k elements are allowed in pq.

arr : 20 6 30 12 5 18 7 19 14

↑

k = 4

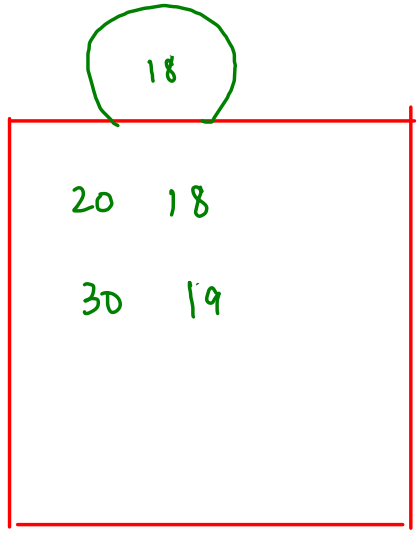
```
for(int val : arr) {
    if(pq.size() < k) {
        pq.add(val);
    }
    else if(pq.peek() < val) {
        pq.remove();
        pq.add(val);
    }
}
```

n log k

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

k log k

minPQ



18 19 20 30

T : n log k + k log k

S : k

Sort K-sorted Array

input

k-sorted array: 20 40 10 60 30 35 75 70 50

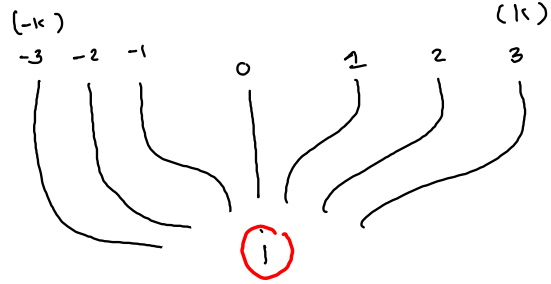
sorted arr: 10 20 30 35 40 50 60 70 75

k+1

k = 3

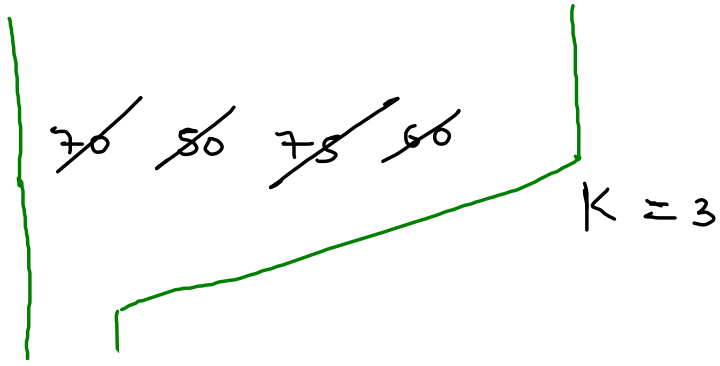
T: nlogk

S: k



k+1

k-sorted array: 20 40 10 60 30 35 75 70 50



Sorted arr: 10 20 30 35 40 50 60 70 75

K sorted array

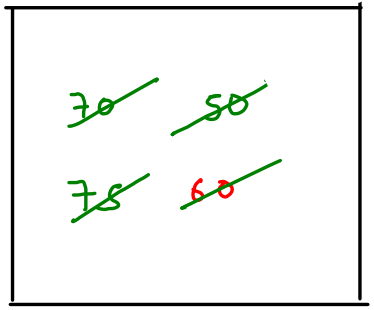
20	40	10	60	30	35	75	70	50
0	1	2	3	4	5	6	7	8

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

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


//print remaining k+1 elements
while(pq.size() > 0) {
    System.out.println(pq.remove());
}
```

$k \log k$

$(n-k) \log k$

$k \log k$



$k = 3$

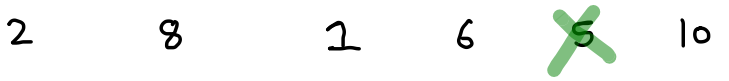
T : $n \log k$

S : k

10 20 30 35 40 50 60 70 75

Median Priority Queue

```java
public static class MedianPriorityQueue {
  PriorityQueue<Integer> left;
  PriorityQueue<Integer> right;

  public MedianPriorityQueue() {
    left = new PriorityQueue<>(Collections.reverseOrder());
    right = new PriorityQueue<>();
  }

  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
  }
}
```



add(2)

add(8)

add(1)

add(6)

add(5)

peek() → median = 5

add(10)

peek() → median = 5

remove()

peek() → median = 6

2 9 6 3 4 17 1

1 2 3 | 4 9 17

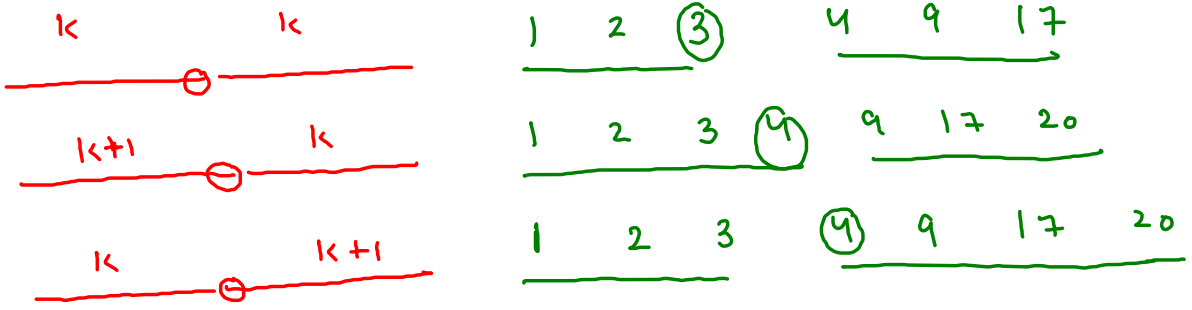
add (6)

maxPQ

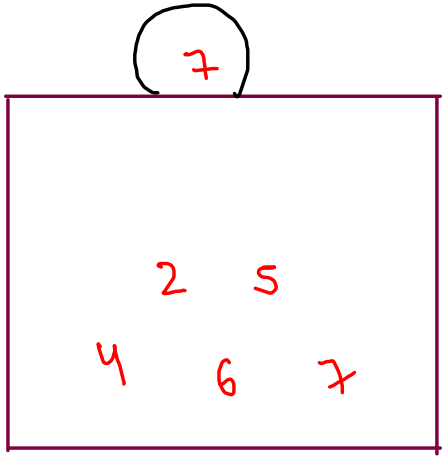
left

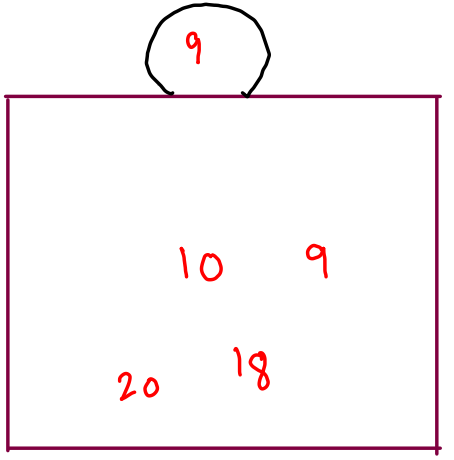
minPQ

right

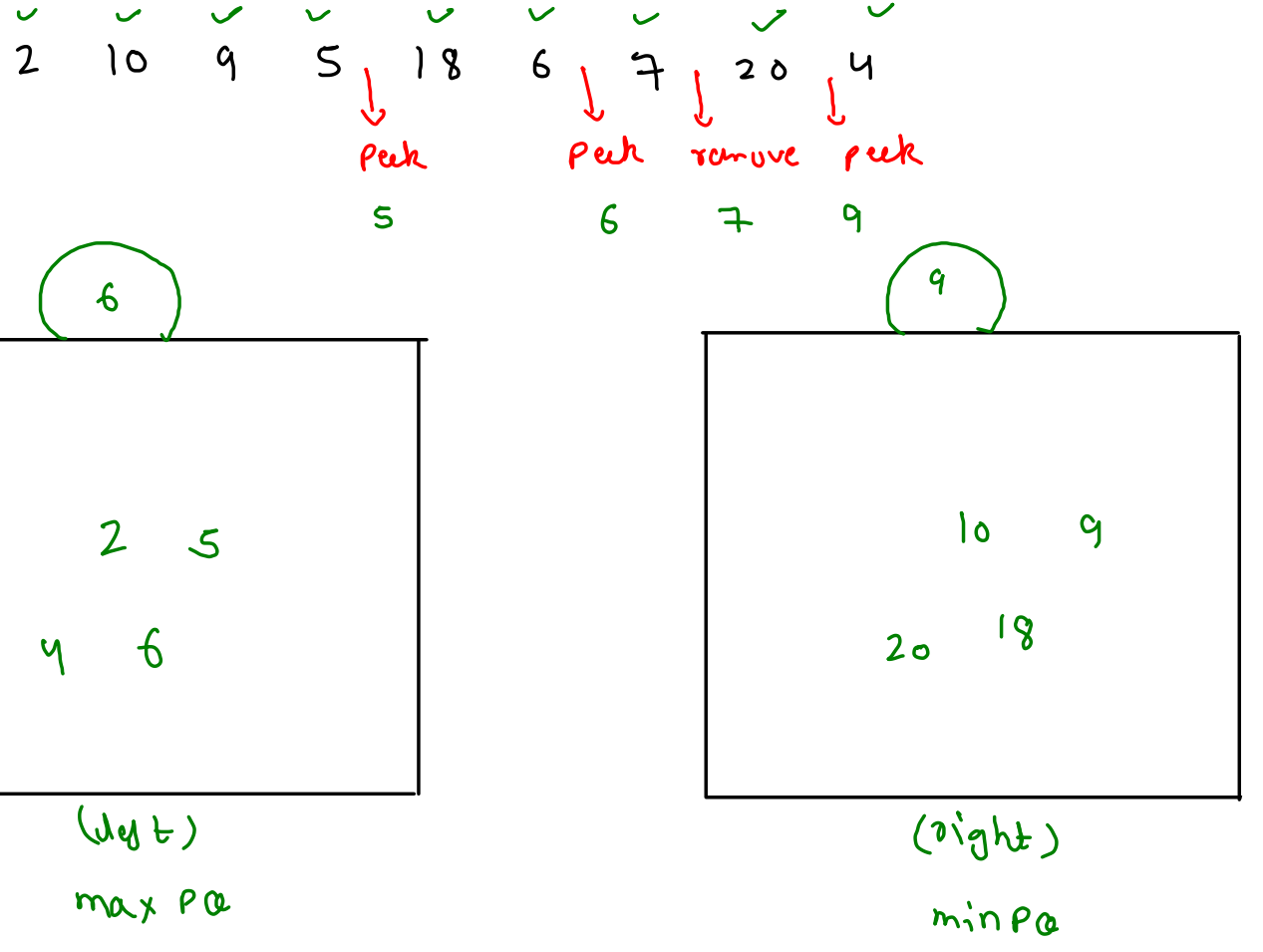


2 10 9 5 18 6 7 20 4





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


    //to manage the size gap
    if(left.size() - right.size() == 2) {
        right.add(left.remove());
    }
    else if(right.size() - left.size() == 2) {
        left.add(right.remove());
    }
}
public int remove() {
    if(size() == 0) {
        System.out.println("Underflow");
        return -1;
    }

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

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

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



Merge K Sorted Lists

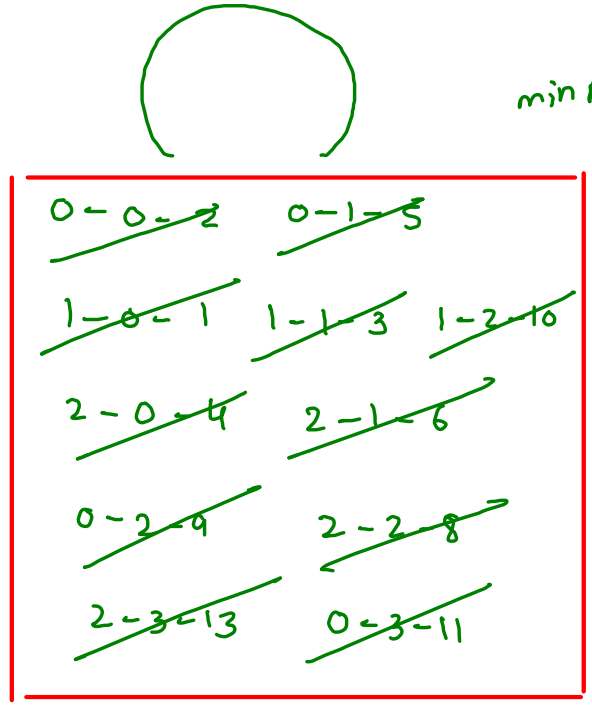
ln_i_val

l1 : 2$_0$ 5$_1$ 9$_2$ 11$_3$ →

l2 : 1$_0$ 3$_1$ 10$_2$ →

l3 : 4$_0$ 6$_1$ 8$_2$ 13$_3$ →

ml → 1 2 3 4 5 6 8 9 10 11 13



Pair {

int ln;

int i;

int val;

}

l_0 : 2_0 5_1 9_2 11_3 →

l_1 : 1_0 3_1 10_2 →

l_2 : 4_0 6_1 8_2 13_3 →

T : nk log k

S : k

mL → 1 2 3 4 5 6 8 9 10 11 13

(ln - i - val)

n → avg. no elements in a list

k → no. of lists

```
for(int i=0; i < lists.size();i++) {
    Pair p = new Pair(i,0,lists.get(i).get(0));
    pq.add(p);
}

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

    int ln = rem.ln;
    int i = rem.i + 1;

    if(i < lists.get(ln).size()) {
        Pair p = new Pair(ln,i,lists.get(ln).get(i));
        pq.add(p);
    }
}
```

k * log k

nk * log k

minPQ