


Priority Queue

⇒ min priority queue

PriorityQueue < Integer > pg = ^{name} new PriorityQueue < > ();

↳ minimum element will be removed first.

⇒ max priority queue

PriorityQueue < Integer > pg = new PriorityQueue < > (Collections.reverseOrder());

↳ maximum element will be removed first.

functions

1) add → pg.add(5); ^{log n}

2) peek → pg.peak(); ^{o(1)}

3) remove → pg.poll(); ^{log n}

$O(n)$ \gg $O(\log n)$

4) size → pg.size();
↓
 $O(1)$

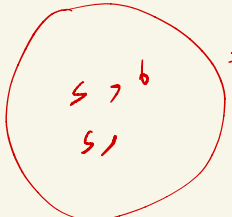
$k=3$

```
public int findKthLargest(int[] nums, int k) {  
    int n=nums.length;  
  
    PriorityQueue<Integer> pq=new PriorityQueue<>(Collections.reverseOrder());  
  
    for(int i=0; i<n; i++){  
        pq.add(nums[i]);  
    }  
}
```

[3,2,3,1,2,4,5,5,6], k = 4



```
public int findKthLargest(int[] nums, int k) {  
    int n=nums.length;  
  
    PriorityQueue<Integer> pq=new PriorityQueue<>();  
  
    for(int i=0; i<n; i++){  
        pq.add(nums[i]);  
  
        if(pq.size()>k){  
            pq.remove();  
        }  
    }  
  
    return pq.remove();  
}
```

min \Rightarrow  \Rightarrow k largest elements