Today's agenda

- Introduction to Heap / PQ.
- K smallest element. +1
- Median of an array. -> {leetcode hard}

// Introduction

	insert (n)	getmin()	deletemin()
ArrayList	O(1)	O(N)	O(N)
LinkedList	~~O(N)~~ O(1)	O(N)	O(N)
queue	O(1)	O(N)	O(N)
HashMap	O(1)	O(N)	O(N)
PQ	O(logN) (no. of elements in PQ)	O(1)	O(logN)

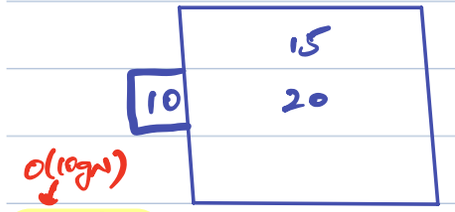
PriorityQueue<Integer> PQ = new PriorityQueue<>();
→ Collections.reverseOrder()

min PQ

max PQ

↳ PriorityQueue<Integer> PQ = new PriorityQueue<>(); (PQ: name)

min PQ



O(logN)

add(10)
add(20)
add(5)

add(15)
remove() → 5
min() → 10

→ PQ.add(n); → O(logN)
→ PQ.remove(); → O(logN)
→ PQ.peek(); → O(1)
→ PQ.size(); → O(1)

Q) Kth Smallest Element

↳ Given N distinct elements, Print K Smallest elements.

Ex: arr[10] = { 8 3 10 4 11 2 7 6 14 1 }
(0 1 2 3 4 5 6 7 8 9)

K=4: 1 2 3 4

arr[8] = { -3 6 2 0 8 7 10 4 }
(0 1 2 3 4 5 6 7)

K=3: -3 0 2

//idea 1

↳ Sort the array and Return the first K elements.

T.C: O(NlogN)

//idea 2

↳ Add all elements to min PQ and get the first K elements.

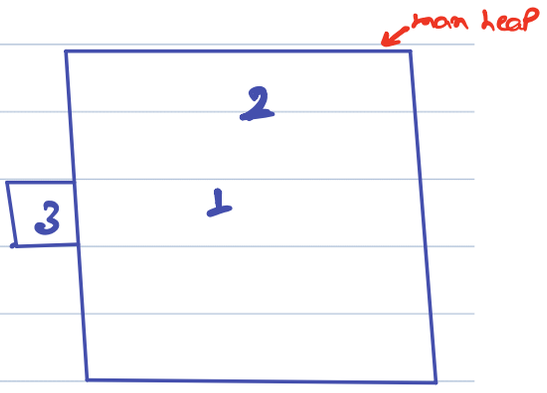
T.C: O(NlogN)

//idea3 → K smallest elements

arr[10]: { 8 3 10 4 11 2 7 6 14 1 }
0 1 2 3 4 5 6 7 8 9

K:3

4 1 2 3



//Pseudo code

Smallest → dhundh
↓
weakest Performer

```
void KthSmallest(int arr[N], int K){
    maxHeap<int> mh;

    for(int i=0; i<K; i++){
        mh.add(arr[i]);
    }
    for(int i=K; i<N; i++){
        if(arr[i] < mh.peek()){
            mh.remove();
            mh.add(arr[i]);
        } else { continue; }
    }
    while(mh.size() > 0){
        S.O.P(mh.remove());   // or return mh.peek();
    }
}
```

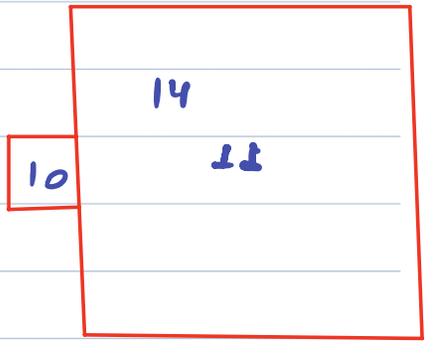
T.C: O(NlogK)
S.C: O(K)

Q) K largest elements

	0	1	2	3	4	5	6	7	8	9
arr[10]: {	8	3	10	4	11	2	7	6	14	1 }

K:3

✓min heap



//median

↳ middle element of sorted array.

arr[3] : { 2 5 3 }

↳ {2, 3, 5} → 3

arr[5] : { 4 3 6 8 5 }

↳ { 3 4 5 6 8 } → 5

arr[6] : { 4 3 9 5 12 2 }

↳ { 2 3 4 5 9 12 } → $\frac{4+5}{2}$ = 4.5

arr[4] : { 4 6 10 14 }

↳ $\frac{6+10}{2} = \frac{16}{2} = 8$

Break till 9:28 PM

leetCode 295

Q) Print median after each insertion.

arr[5]: 9 6 3 10 4

9 → 9 | 6 → 7.5 | 3 → 6 | 10 → 7.5 | 4 → 6

//idea1

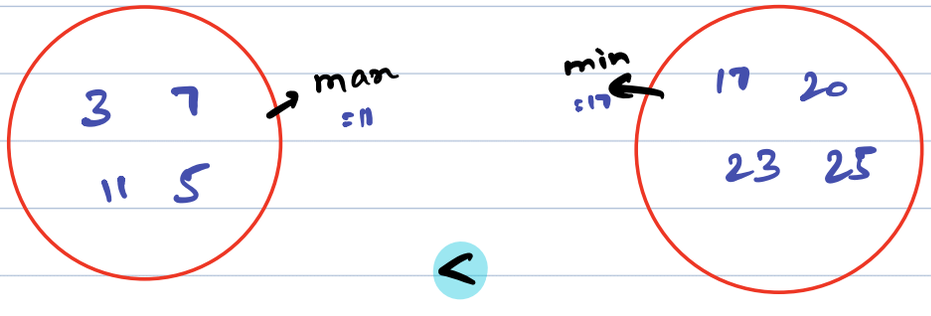
↳ After each insertion, sort the array & return the middle one.

T.C: $N * N\log N \approx O(N^2 \log N)$

$< N\log N$

//idea2

3 5 11 23 20 25 17 7

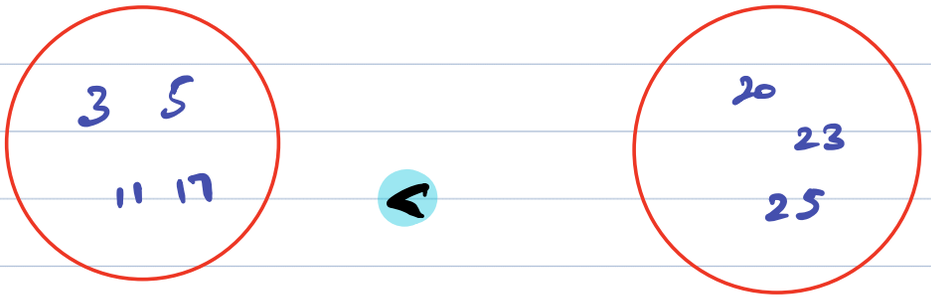


* if total elements are even:

↳ median = $\frac{\text{max of left bucket} + \text{min of right bucket}}{2}$

odd

3 5 11 23 20 25 17



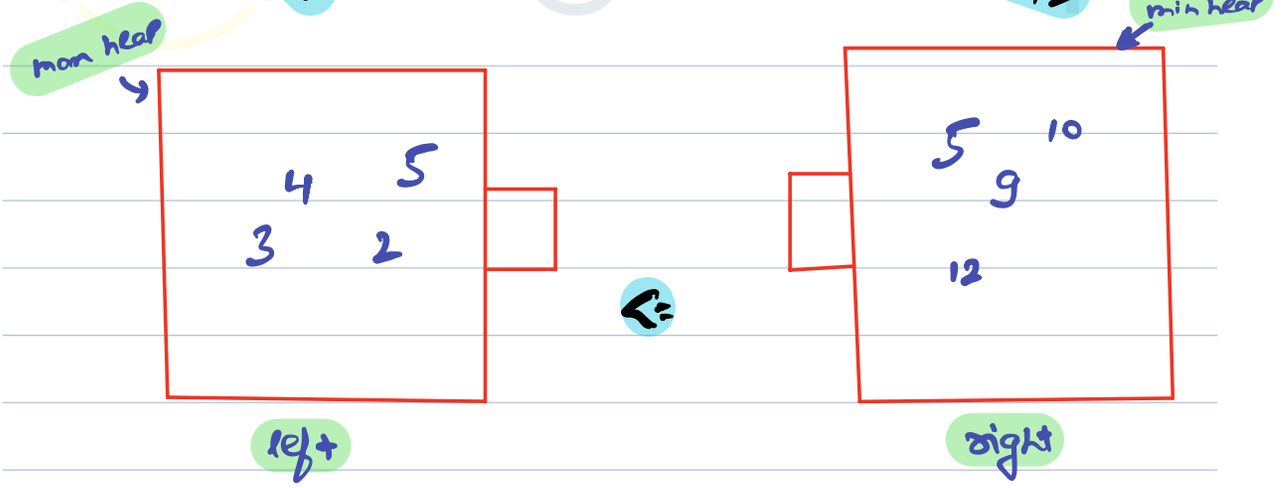
if total elements are odd:

↳ median = max of left bucket

4 9 5 3 12 2 10 5

4 6.5 5 4.5 5 4.5 5 5

N/2 N/2



if (left.size() == right.size())

↳ ultimately new element should be added to left PQ but to maintain inequality pass it via right PQ.

if (left.size() != right.size())

↳ ultimately new element should be added to right PQ but to maintain inequality pass it via left PQ.

//Pseudo Code

```
Class MedianFinder {
    maxHeap<Integer> left;
    minHeap<Integer> right;

    Public MedianFinder() {
    }
```

T.C: O(N*logN)

S.C: O(N)

```
    Public void addNum(int num){
        if(left.size() == right.size()){
            right.add(num);
            left.add(right.remove());
        }
        else {
            left.add(num);
            right.add(left.remove())
```

```
    }
}

Public double findMedian() {
    if (left.size() == right.size()) {
        double ans = (left.peek() + right.peek()) / 2.0;
        return ans;
    }
    else {
        return left.peek() * 1.0;
    }
}
```