Today's agenda

- ↳ Introduction to Heap/PQ. → PriorityQueue
- ↳ K smallest element. +1
- ↳ median of an array. → {leetcode Hard}

AlgoPrep

// Introduction

	insert (x)	getmin ()	delete min ()
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)	O(1)	O(logN)

Smart Array

Min PQ

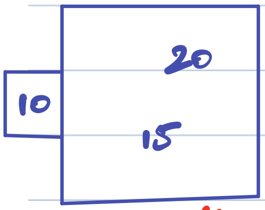
name

PriorityQueue < Integer > PQ = new PriorityQueue <>();

Max PQ

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

PQ



→ PQ.add (10);
→ PQ.add (20);
→ PQ.add (15);
→ PQ.add (5);
→ PQ.remove(); → 5
→ PQ.peek(); → 10
→ PQ.size(); → 3
↳ O(1)

no. of elements in PQ

↳ PQ.add(x) → add value x → O(logN)
↳ PQ.remove() → remove min ele → O(logN)
↳ PQ.peek() → get the min ele.

Q) Kth Smallest element

↳ Given N distinct elements, Print K Smallest elements

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

               0   1  2  3  4  5  6   7
    arr[9] = {-3  6  2  0  8  7  10  4}
      K=3:   -3  0  2
```

//idea 1

↳ Sort the array and return the first K elements.

T.C: $O(N\log N + K)$

//idea 2

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

T.C: $O(N\log N + K\log N)$

AlgoPrep

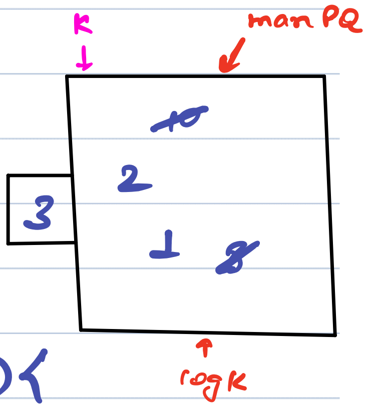
//idea 3

k->3

0 1 2 3 4 5 6 7 8 9

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

↳ 1 2 3



//Psuedo code

```
void Kthsmallest (int arr[N], int k){
    maxHeap < Integer> 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]);
        }
    }

    while (mh.size() >0){
        s.o.p (mh. remove());
    }
}
```

T.C: O(NlogK)

S.C: O(K)

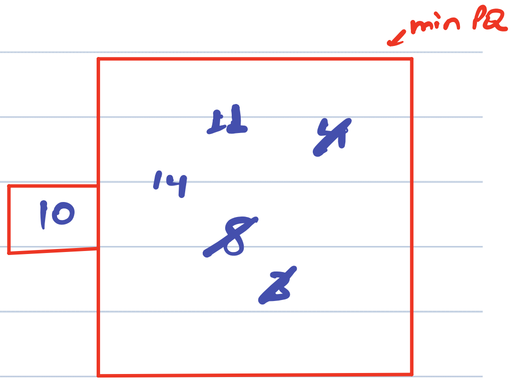
Q) K largest elements

$$\begin{array}{lcccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ arr[10]: \{ & 8 & 3 & 10 & 4 & 11 & 2 & 7 & 6 & 14 & 1 \} \end{array}$$

$\downarrow i$

K=3

↳ 10 11 14



AlgoPrep

//median

↳ middle element of sorted number.

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

AlgoPrep

arr[4] : {4 6 10 14}

↳ $\frac{6+10}{2}$ = 8

Break till 9:18 pm

leetCode 295

Q) Print median after each insertion.

arr[5]: 9 6 3 10 4

$\downarrow$ 9 $\quad\downarrow$ 7.5 $\quad\downarrow$ 6 $\quad\downarrow$ 7.5 $\quad\downarrow$ 6

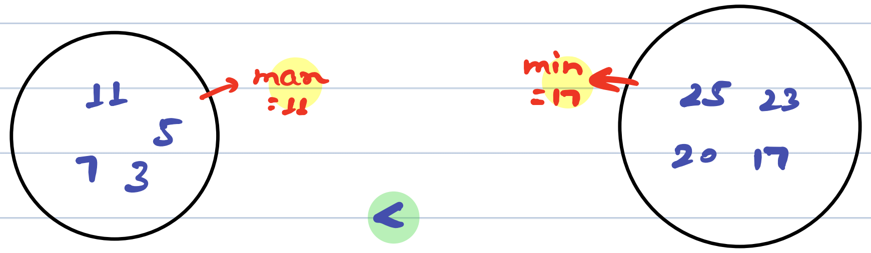
//idea1

↳ After every insertion, sort the array and return the middle one.

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

//idea2

3 5 11 23 20 25 17 7



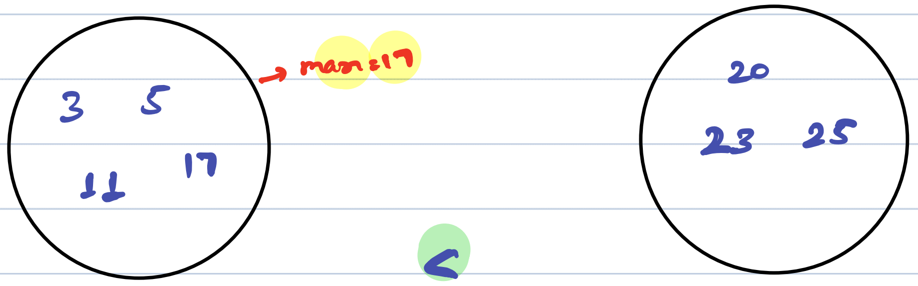
* if element count is even:

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

AlgoPrep

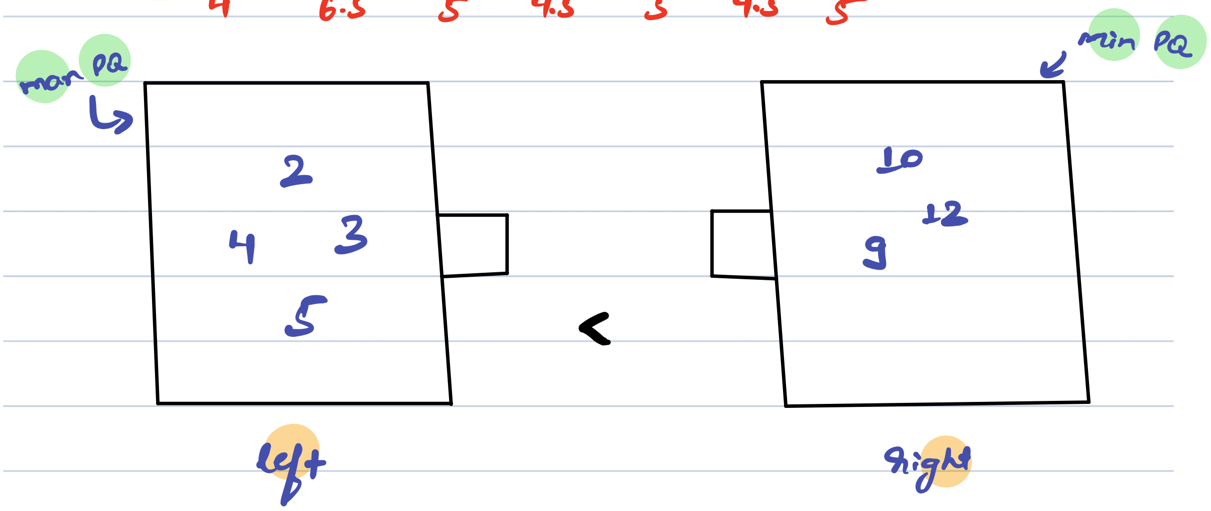
* if element count is odd :

3 5 11 23 20 25 17



median= max of left

→	4	9	5	3	12	2	10
	↓	↓	↓	↓	↓	↓	↓
	4	6.5	5	4.5	5	4.5	5



AlgoPrep

```
if ( left.size() == right.size()) {
        ↳ ultimately new number should go to left
    PQ but to maintain inequality you have to Pass it via
        right PQ.
}
```

```
if (left.size() != right.size()){
        ↳ ultimately new number should go to right
    PQ but to maintain inequality you have to Pass it via
        left PQ.
}
```

AlgoPrep

//Psuedo Code

T.C: O(NlogN)
S.C: O(N)

```
class medianfinder {
    maxHeap<Integer> left;
    minHeap<Integer> right;

    Public medianfinder() {

    }
    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()) {
            return $\frac{left.Peek() + right.peek()}{2.0}$;
        }
        else {
            return left.peek()*1.0;
        }
    }
}
```

3logN ←

O(1) ←

AlgoPrep