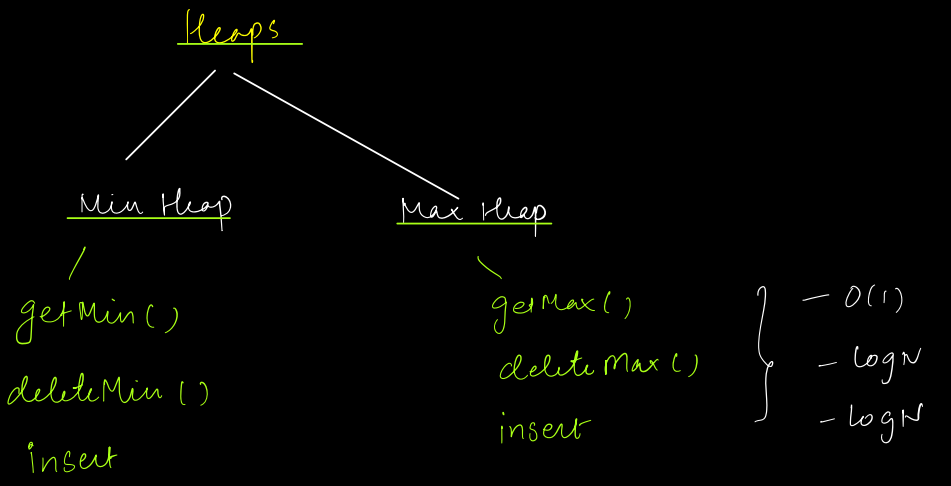
# Today's Agenda

1) Introduction
2) K Largest | Smallest elements
3) Sort K-Sorted array
4) Running Median

| | getMin() | deleteMin() | insert() |
|---|---|---|---|
| Arraylist / list | O(N) | O(N) | O(1) |
| Linked list | O(N) | O(N) | O(1) |
| Queue | O(N) | O(N) | O(1) |
| Hashmap | O(N) | O(N) | O(1) |
| Heaps | O(1) | O(log N) | O(log N) |




Inbuilt — Priority Queue — Check in your own language of choice.

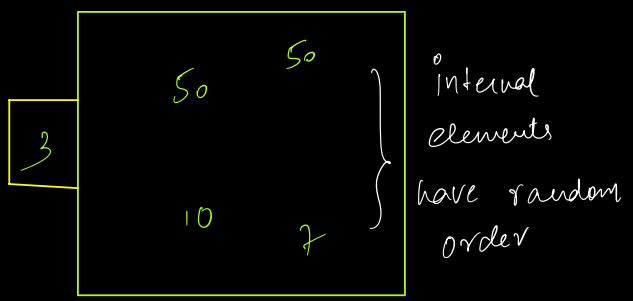
minheap < > mh

maxheap < > mh;

Priority Queue = Heap

```
minheap< int > mh;
```

```
add(50)
add(3)
add(10)
add(1)
add(7)
add(-5)
add(50)
add(-5)
```




Super Smart Array

```
deleteMin()
deleteMin()
```

} log N

10) Given N distinct elements, print K smallest elements in array.

Given K

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

K=4 : [1 2 3 4]

arr[]: {-3 6 2 0 8 7 10 4}

K=3: [-3 0 2]

Idea 1)

Sort the array & return the first K elements.

TC: $N \log N + K \simeq O(N \log N)$

Idea 2) Add all elements in min heap & get first K elements by removing one by one.

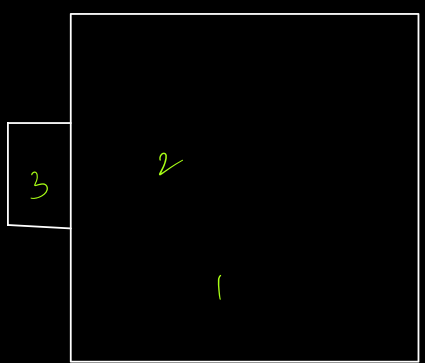
TC: $N \log N + K \log N$

Idea 3 : Max Heap

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

0 1 2 3 → * * * *

K = 3 :




K elements

insert / delete - log K

O/P : 3 2 1

TC : N * log K

SC : O(K)

O(N log K)

```
maxHeap < int > mh;

for(int i = 0; i < K; i++){
    mh.add(ar[i])
}

for(int i = K; i < N; i++){
    if(ar[i] < mh.getMax()){
        mh.deleteMax()
        mh.add(ar[i])
    }
}
```

Ans = heap elements

2Q) Given N distinct elements, every element is atmost K index far from its actual sorted position.

Sort given array.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[10]: | 4 | 9 | 27 | 3 | 39 | 44 | 7 | 14 | 30 | 21 |

K = 4

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| O/P | 3 | 4 | 7 | 9 | 14 | 21 | 27 | 30 | 39 | 44 |

Approach 1 : Sort

TC : O(N Log N)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[10]: | 4 | 9 | 27 | 3 | 39 | 44 | 7 | 14 | 30 | 21 |

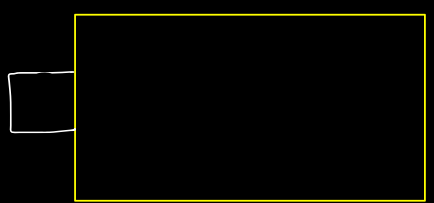
K = 4

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 3 | 4 | 7 | 9 | 14 | 21 | 27 | 30 | 39 | 44 |

→ for min heap

SC: O(K)

TC: N * log(K)



→ at max Min heap size: (K+1)

Code TODO

Break: 10 Min

Median ::

↳ Middle element of sorted array

Ex ar[3] : { 1 4 2 } : 2

Ex ar[5] : { 3 2 5 7 4 } : 4

Ex ar[6] : { 3 2 8 4 10 1 } : 3

1 2 3 4 8 10

Ex ar[6] : [1, 2, 5, 4, 3, 6] : 3

[1 2 3 4 5 6]

## 3Q) Print Median after each insertion

arr[] :⇒ 9 6 3 10 4

9 7 6 7 6

3 4 6 9 10

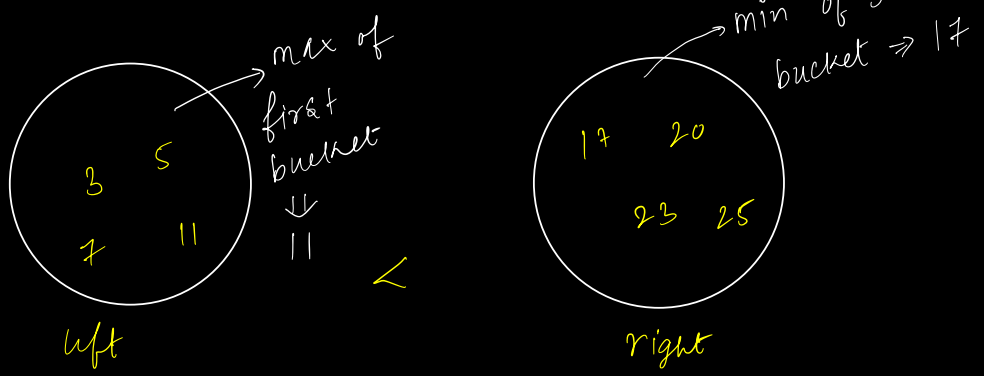
Left <= right

Idea1) After each insertion, sort the array & return middle one

Tc: $N(N \log N) \simeq N^2 \log N$

Idea2)

3 5 11 23 20 25 17 7

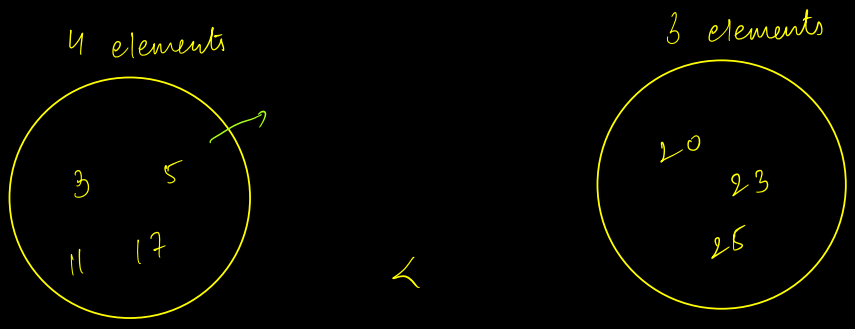



If total elements are even :-

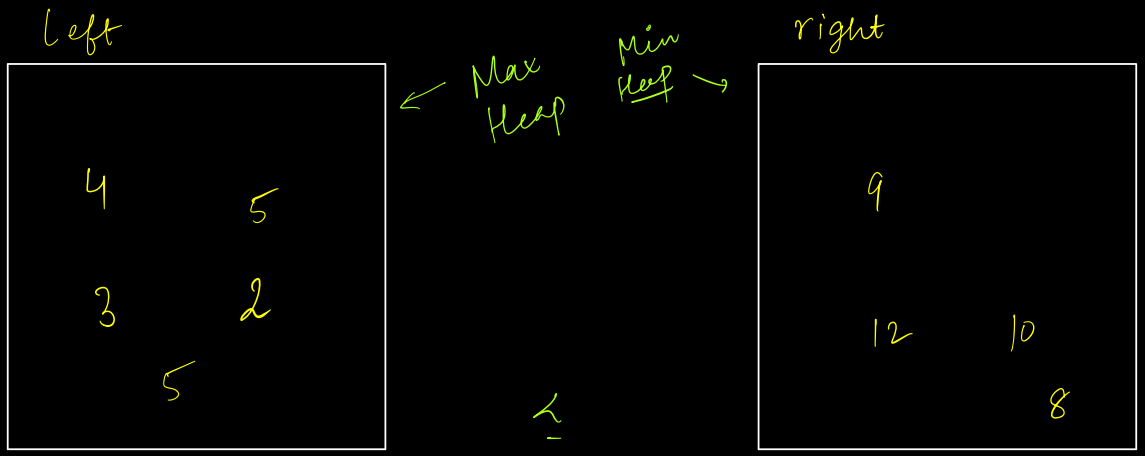
$$\text{Median} = \frac{\text{Max of first part} + \text{Min of II part}}{2}$$

Odd:

3 5 11 23 20 25 17




Median = Max of left bucket




| 4 | 9 | 5 | 3 | 12 | 2 | 10 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 6 | 5 | 4 | 5 | 4 | 5 | 5 | 5 |

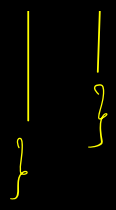
```
if (left.size() == right.size()) {
    ↳ ultimately element should go to left bucket but
    to maintain equality pass it via right bucket
    Median = left.getMax()
}
else {
    ↳ ultimately element should goto right but to
    maintain equality pass it via left bucket
```

Median = $\frac{\text{left.getMax} + \text{right.getMin}()}{2}$

}

Pseudo Code

```
minheap < int > right

maxheap < int > left

for(int i = 0; i < N; i++) {

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

        right.insert(ar[i])
        left.insert(right.getMin()), right.delMin())

        Median = left.getMax

    }
    else {

        left.insert(ar[i])
        right.insert(left.getMax()) left.delMax()
```



$$\text{Median} = \frac{\text{left.getMax} + \text{right.getMin}}{2}$$

TC : TODO

Thank You 😊

Please try to implement all 3 problems that we solved today!

If reach out