Welcome ☺

Agenda : Heap Sort
In-place heap
2-3 ques^n's.

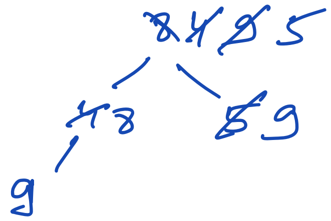
Q Sort an array using a heap.

⇒ 1. Build a min. heap → O(N) / O(N) space

2. Extract min. values and store the results.
↳ O(N log N)

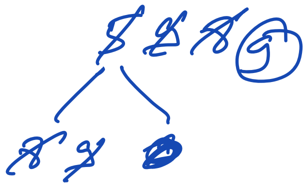
T.C ⇒ O(N log N)

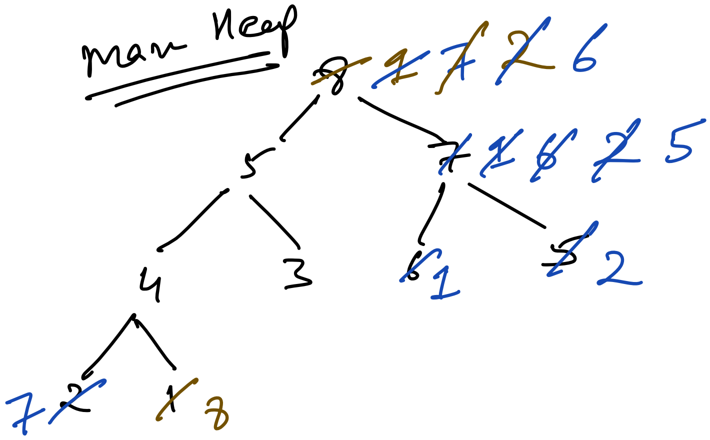
SC ⇒ O(N)



Can we reduce the extra space ?

2	4	5	8	9





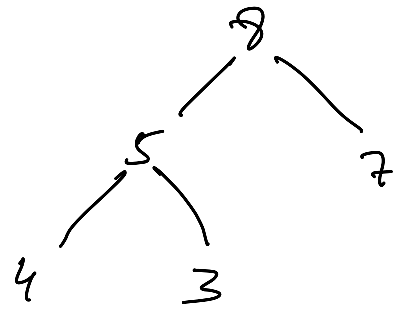
0	1	2	3	4	5	6	7	8
~~8~~ ~~1~~ 7	5	~~7~~ ~~1~~ 6	4	3	~~6~~ 1	5	2	~~1~~ 8

1. Build max Heap. ⇒ O(N)/O(1) extra space

2. Extract getMax

3. Heapify (with size reduced)

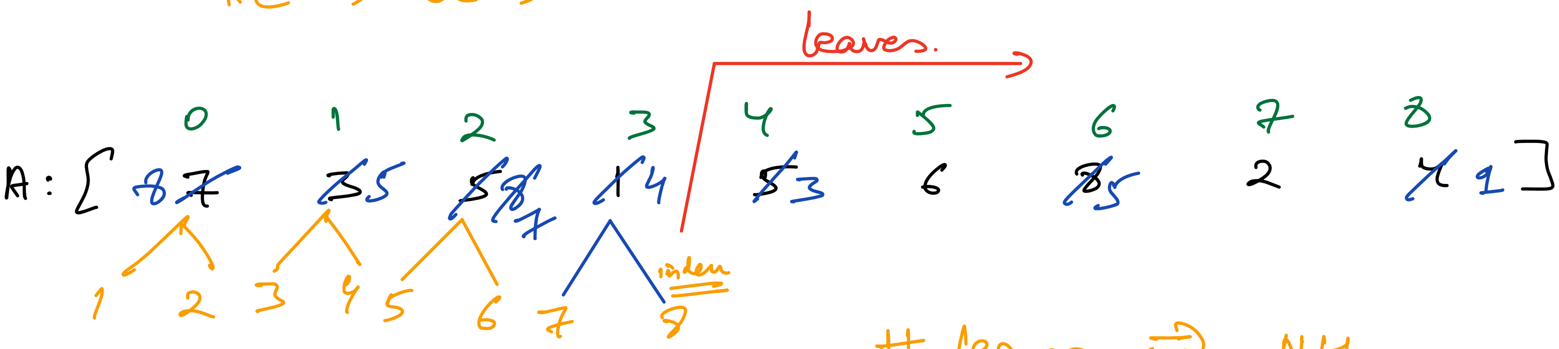
4. Repeat 2 & 3 (N-1) times.



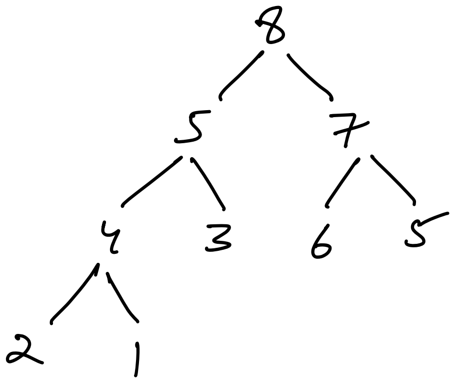
In-place Heap Build

S.C ⇒ O(1)

T.C ⇒ O(N)



leaves ⇒ $\frac{N+1}{2}$



T.C ⇒ O(N)

SC = O(1)

Pseudocode

```
1.  Build Max Heap.
2.  j = N-1
    while ( j > 0)
    {
        swap ( A[0] , A[j] )
        j--
        heapify ( 0, A , j )
    }
```

Q Given array; find K^{th} largest element

App 1 Sort array & return $N-K^{th}$ element.

App.2 Binary Search.

App.3

1. Build a max heap.
2. Call extractMax() K-1 times to remove K-1 elements.

   Call extractMax() to get K^{th} Largest.

T.C ⇒ $O(N + K\log N)$

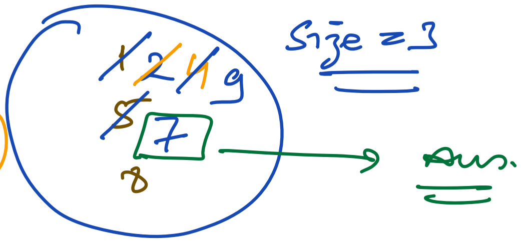
SC ⇒ $O(1)$ / $O(N)$

App4 Using min Heap.

8 5 1 2 4 9 7 $K = 3$

Extract & Insert if element is greater than smallest element in the heap.

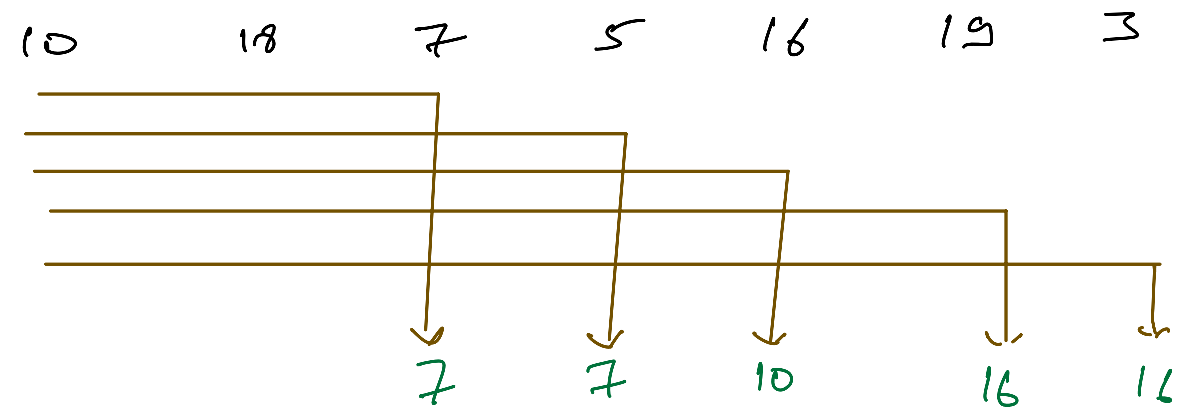
T.C $(K + (N-K)\log K)$



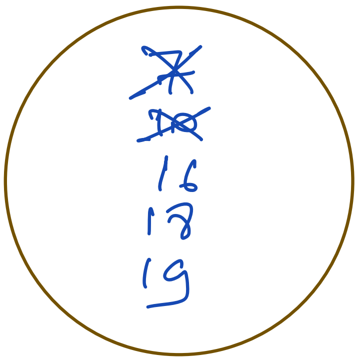
Q Find K^{th} largest element for all windows of an array starting from index 0. ∀ window of size ≥ K.

eg:

K=3



1. Min heap
2. Compare & update
3.



Ans- 7 7 10 16 16

Pseudocode

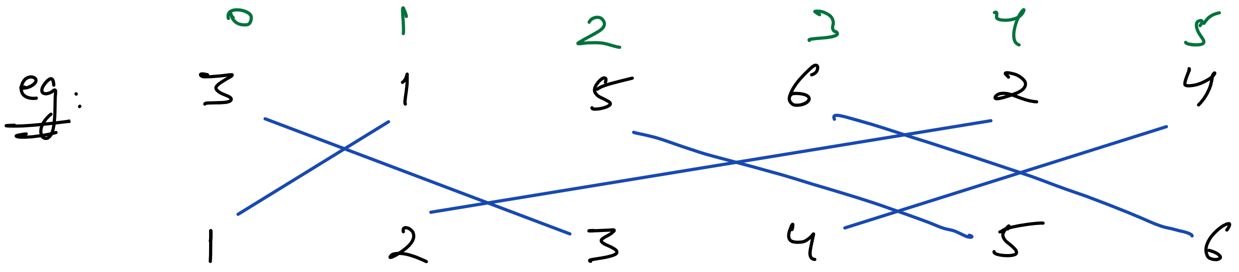
```
if ( A[i] ≤ root of min. Heap ) → no updates

else {
        remove min element from heap.
        Insert A[i]
}
```

Q Sort nearly sorted arrays.

↳ every element is at max K distance away from its posⁿ in sorted order.

eg:

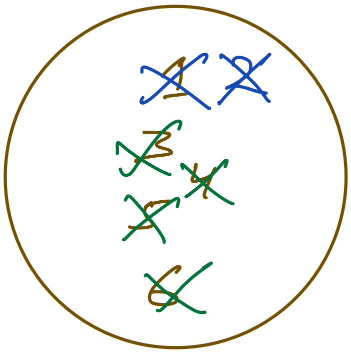


K=3

BF solⁿ Sort the array. O(NlogN)

Solⁿ

1	2	3	4	5	6	



Pseudo

```
Build min. Heap of size K+1
for ( i → 0 to K) {
      heap. insert ( A[i])
}
ind = 0
for ( i → K+1 to N-1)
{
      ans [ ind ] = heap. extractMin ( )
      ind ++
      heap. insert ( A[i])
}
while ( ! heap .isEmpty ( ) ) {
      ans [ ind ] = heap. extractMin ( )
      ind ++
}
```

T.C ⇒ Nlog(K+1)

Q Given an integer input stream, find median at every step.

↳ middle element of sorted data.

i/p	9	8	7	3	6
o/p	9	$8.5 \simeq 8$	8	$7.5 \simeq 7$	7

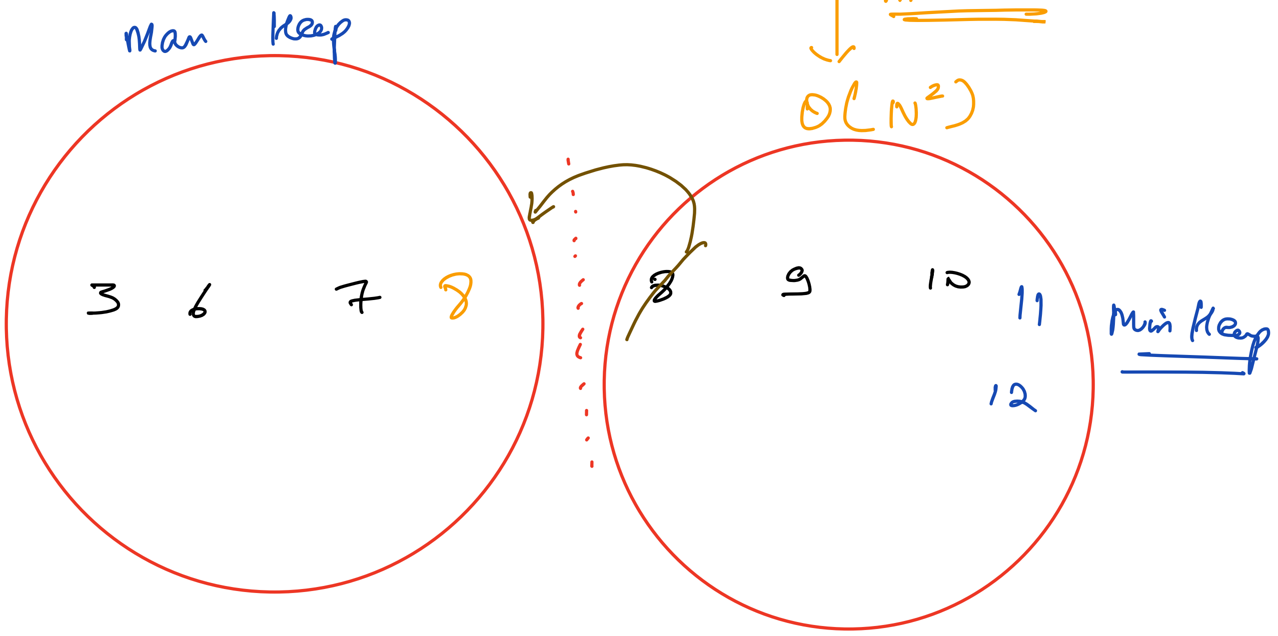
Sol1 Sort at every step / insertion.

$$N * N \log N = \underline{\underline{N^2 \log N}}$$

$\downarrow$ insertion

$O(N^2)$

Sol2



insert 11 $\Longrightarrow$ 8

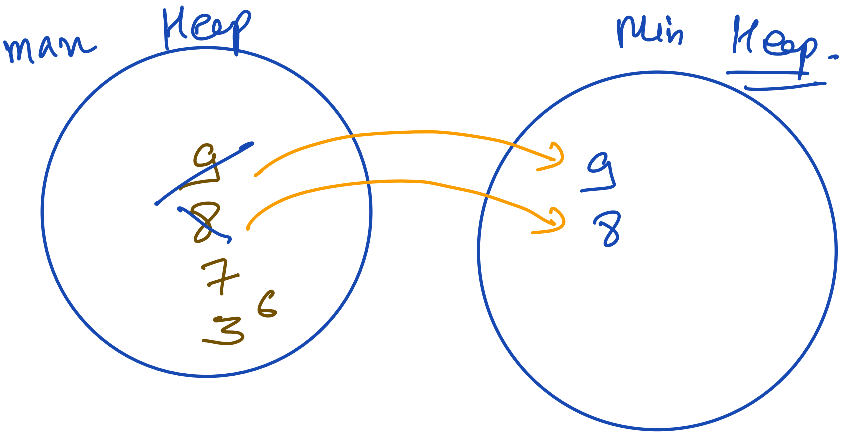
insert 12

$$TC \Rightarrow \sum_{i=1}^{N} i \log i$$

$$\Rightarrow N \log N$$

$$SC \Rightarrow O(N)$$

9 8 7 3 6



O/P ⇒ 9 8.5≃2 8 7.5≃7 7

Code

```
∀ input x
if ( x ≤ root of Max Heap )
{
        insert (x) in max Heap.
        if( maxHeapSize - minHeapSize > 1 )    //reshuffle
        {
            move root of maxHeap to minHeap.
        }
        return median based on size
}
else {
        insert (x) in min Heap.
        if( min Heap size - maxHeap size > 1 )    //reshuffle
        {
            move root of minHeap to maxHeap.
        }
        return median based on size
}
```

Syllabus ⇒ Tree Heaps Greedy.