Heap Sort

	0	1	2	3	4	5	6	7
A =	[7	3	5	1	9	6	7	1]

Sol 1 → 1) Create min-heap/max-heap.

2) getMin()/getMax() element N times & store in ans.

$TC = O(N + N\log(N)) = \underline{O(N\log(N))}$

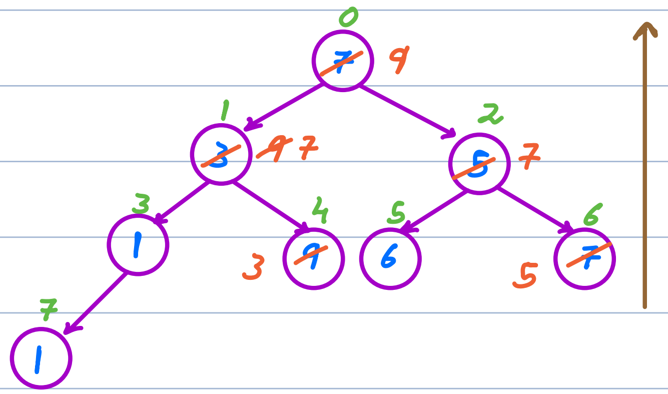
$SC = \underline{O(N)}$ → $\underline{O(1)}$ **Inplace Heap Build**

i/p A[] → Heap → Sorted A[]

Min / Max ✓

	0	1	2	3	4	5	6	7
A =	[~~7~~ 9	~~3~~ ~~9~~ 7	~~5~~ 7	1	~~9~~ 3	6	~~7~~	1]
							5	

lc = A[1], rc = A[2] | lc = A[3], rc = A[4] | lc = A[5], rc = A[6] | lc = A[7], rc = A[8] x



Structure → Complete Binary Tree

N nodes → # leaves = (N+1)/2

8 → # leaves = 4

i → lc = 2i+1

rc = 2i+2

parent = (i-1)/2

TC = $\underline{O(N)}$

↙ **Max Heap**

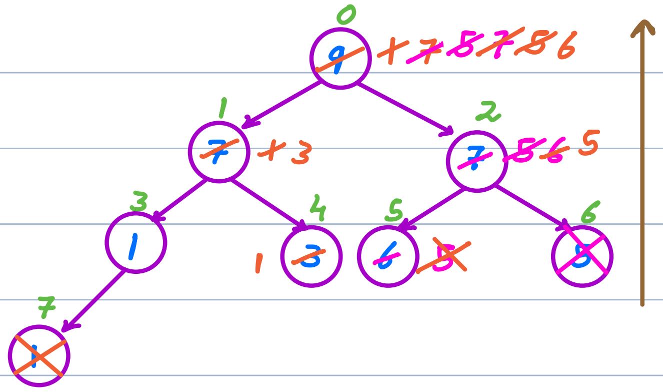
	0	1	2	3	4	5	6	7
A =	[~~9~~	~~7~~	~~7~~	1	~~3~~	~~6~~	~~5~~	~~1~~]
	~~1~~	~~1~~	~~5~~		1	~~5~~	7	9
	~~7~~	3	~~6~~			7		

~~5~~ 5

~~7~~

~~5~~

6



A = [~~6~~ | ~~3~~ ~~5~~ ~~1~~ ~~1~~ 7 7 9]

~~1~~ ~~1~~ 5 6

1 3

~~1~~

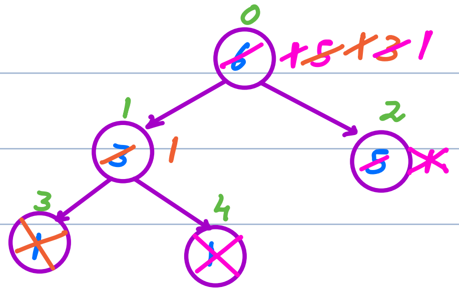
~~5~~

~~1~~

~~3~~

~~1~~

1

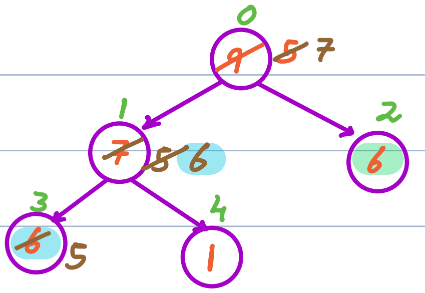


A = [1 1 3 5 6 7 7 9]

SC = O(1)

TC = O(N log (N))

Unstable sorting



Q → Find K^th largest element.

A = [8 5 2 4 1 7] K = 3 Ans = 5

A = [1 2 3 4 5] K = 5 Ans = 1

Sol 1 → sort A[]

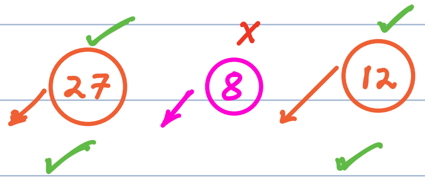
Ans = A[N-K]

TC = O(N log (N))

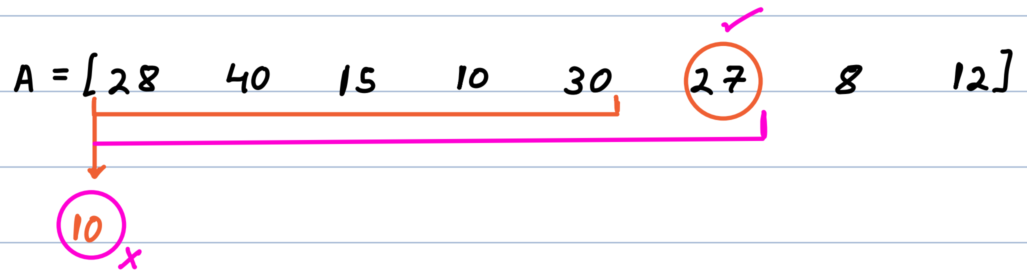
SC = O(1)

K	Ans
1	A[N-1]
2	A[N-2]
⋮	⋮

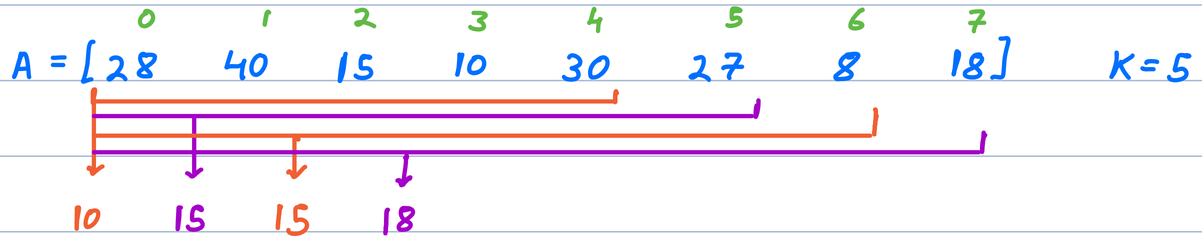
Team of 5 players

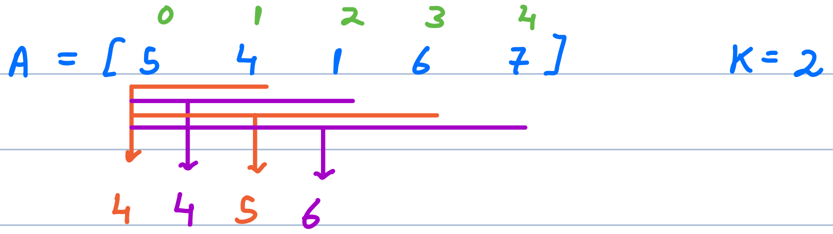


28 ✓ 40 ✓ ~~15~~ ~~10~~ 30 ✓

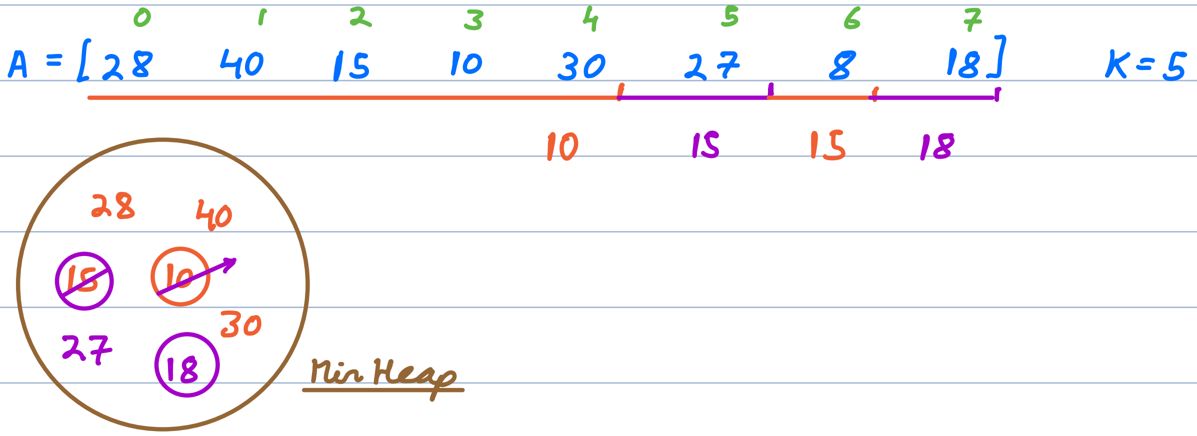


Q → Find K^th largest element ∀ subarray from 0 to i where i >= (K-1).





K^th largest → Min Heap of size K to store K largest elements.



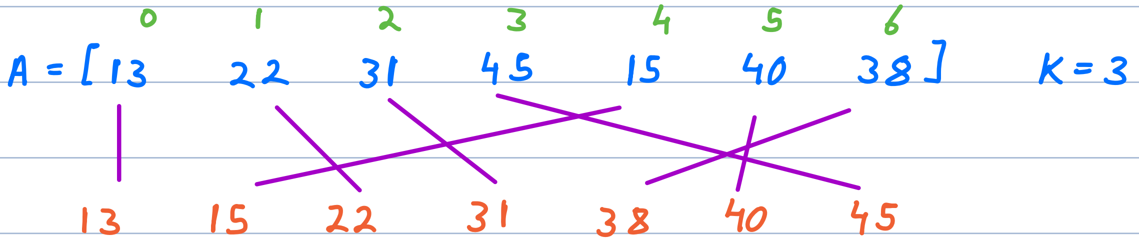
```
// h → minHeap
for i → 0 to (K-1) {
    h.insert(A[i])
}

print(h.root)          // h[0]
for i → K to (N-1) {
    if (h.root < A[i]) {
        h.getMin()     // delete root
        h.insert(A[i])
    }
    print(h.root)
}
```

SC = $O(K)$

TC = $O(N \log(K))$

Q → Sort the given array where every element is atmost K distance away from its position in sorted order. (N >> K)

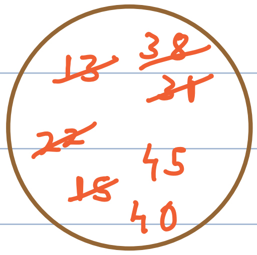


Smallest → [0 ——— K]

use minHeap of size (K+1)

A = [13 22 31 45 15 40 38]   K = 3

Ans → 13 15 22 31 38 40 45



SC = O(K)

TC = O(N log(K))

Q → Given a stream of integers.
Find median ∀ integer intake.

median → middle element in sorted order.

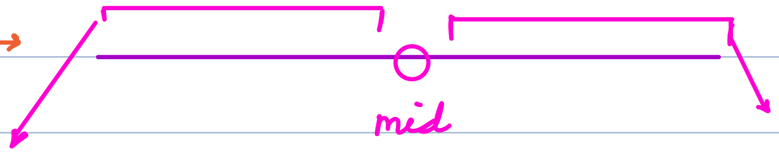
i/p → 9 10 11 . . .
o/p → 9 9 10 . . .

A = [1 2 4 3]
1 2 3 4
→ 2.5

odd → middle in sorted.
even → first middle in sorted. (2)

Bruteforce → ∀ intake, sort & find middle.
TC = O(N * N log(N)) SC = O(1)

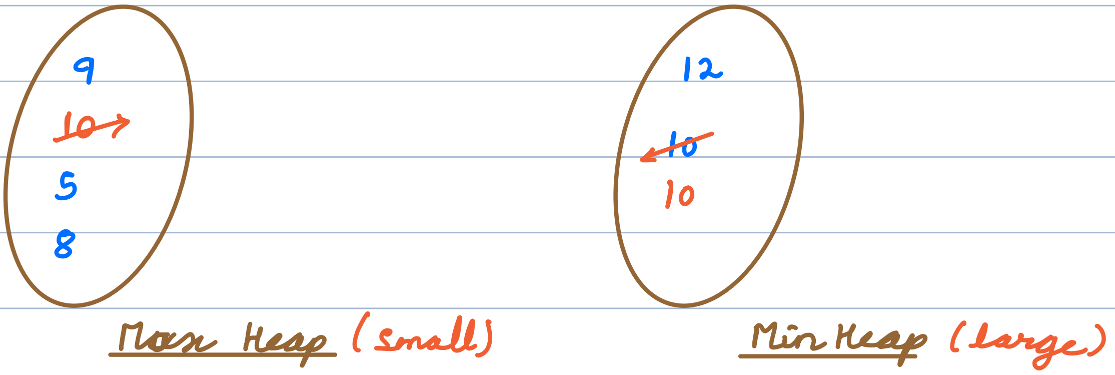
Sol →



store small values in maxHeap ✓

store large values in minHeap. ✓

$$\boxed{\text{Ans = maxHeap root node}}$$



i/p → 9 12 10 5 8

o/p → 9 9 10 9 9

```
// i/p → x
if (x <= root of maxHeap) {
    insert in maxHeap
    if (size of maxHeap - size of min Heap > 1)
            move root of maxHeap to minHeap
} else {
    insert in min Heap
    if (size of min Heap - size of maxHeap > 0)
            move root of min Heap to maxHeap
}
```

TC = $O(N \log(N))$ SC = $O(N)$