sort the array (heaps)

array $\xrightarrow{O(n)}$ build min heap ⟶ extract min again & again

↓

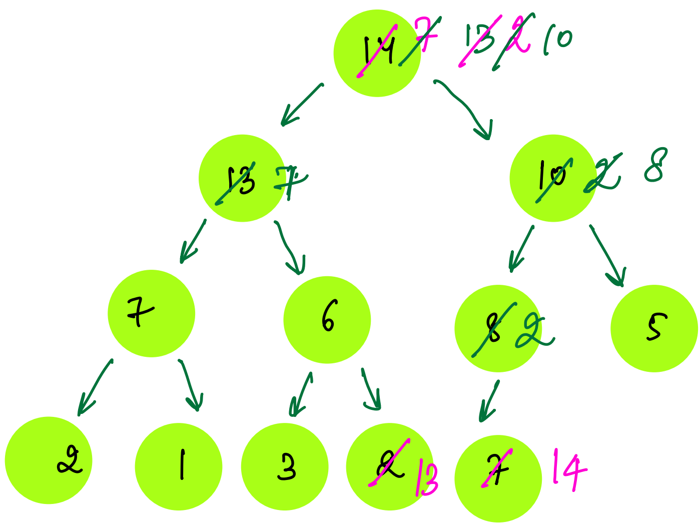
put in a sorted array

T.C: O(n) + nlogn

S.C: O(n) – O(1) (use reverse)

max value should be present at the end ⇒ maxheap

0	1	2	3	4	5	6	7	8	9	10	11
~~14~~ 7	~~13~~ 2	10	7	6	8	5	2	1	3	~~2~~ 13	~~7~~ 14



extractMax()

Heapsort

N + build

T.C: O(N logN)

S.C: O(1)

unstable

array ⟶ build maxheap ⟶ sorted arr

↓

extractMax()

C++ introsort

quick + heapsort

Find k^{th} largest element in the array. (heap)

8 5 1 2 9 6 7 K=3

max-heap → extract max() k times

$O(N) + K \log N$

$O(K) + (N-K) * \log K$

Find k^{th} largest element for every window (0-i) ∀ $i >= k-1$

0 1 2 3 4 5 6

10 8 7 5 16 19 3 K=3

⇓

7 7 8 10 10

B.F: 19 16 10 8 7 5 3

insertion sort

≈ $O(N^2)$

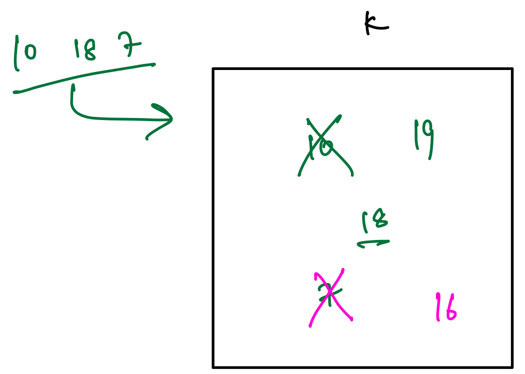
19 16 10 8 7 5 3

5 elements 5th largest

⇓

smallest no.

10 18 7 5 16 19 3



~~5~~ 16 19 3

extract min()
insert(new)

min heap

min of min heap 7

7

10

16

16

1) Build min heap for first k elements
O(k)

2) (n-k) * log k

kth smallest ? max heap

k-sorted array (almost sorted array)

given

⇓

every element in the array is atmost k positions away from its sorted pos

k = 3

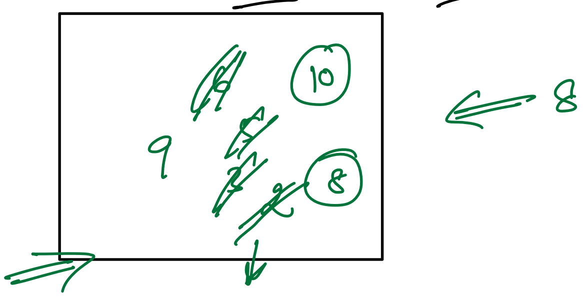
0	1	2	3	4	5	6
6	5	3	2	8	10	9

sorted: 2 3 5 6 8 9 10

B.F:- just perform sorting → n log n

min element will define lie from 0 to k first (k+1)
next min x<=k+1 index

min heap (k+1) element



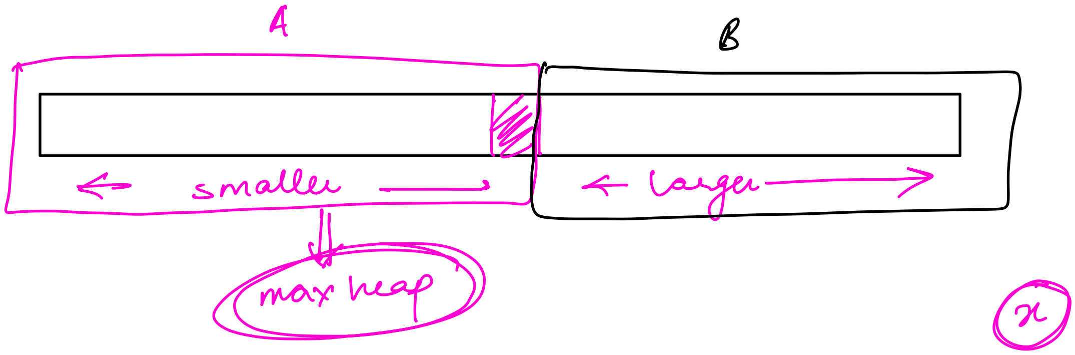
1) Build min heap k+1 elements
≈ O(k)
keep insert an elem
& remove
(n-k) * log k

arr[0]

2 3 5 6 8 9 10

k + (n-k) log k

1 2 3 4 5 6



size (A) - size(B) = {0,1}

x < root of max heap (A)
insert in A

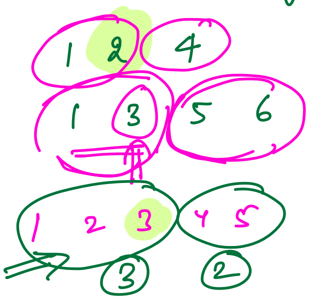
x > max of A
in B

infinite stream of integers, find median of current set of elements

median ↓ middle element in sorted array

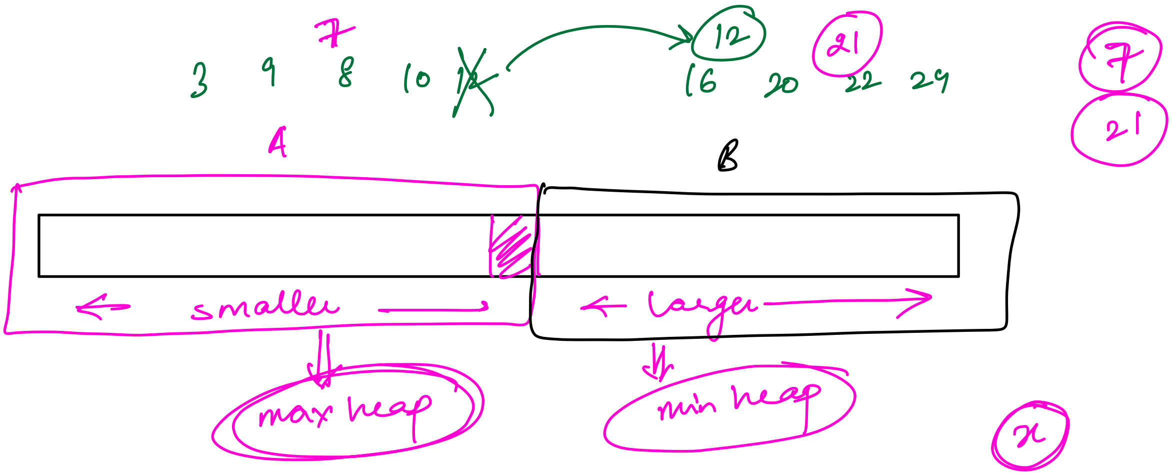
9 8 7 3 6 4 1 . . .

median 9 8 8 7 7 6 6 . . .



B.F:-

- sort every time & find the middle
  ⇓
- insertion sort — $O(n^2)$

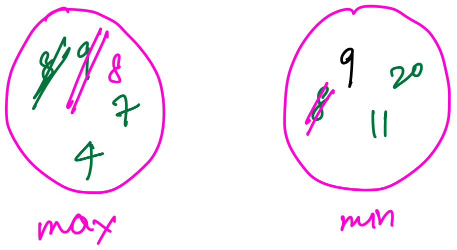


$$\text{size}(A) - \text{size}(B) = \{0, 1\}$$

x < root of max heap (A)
|
insert in A
↓
size is an issue
extract from A & put it in B.

x > max of A
↓
in B
if size cause
extract min from B & put it in A

9 8 7 4 11 20



ans = 9 8 8 7 8 8

A = max
B = min

```
if ( x < max of max heap)
{
    insert x in max heap
    if ( size of (A) - size (B) >1)
        extract max()
        insert into min heap
}
else
{
    insert x into minheap
    if ( size( minheap) > size( maxheap)
    {
        extractmin() &
        put it in maxheap
    }
}
result . insert ( max of max heap)

if ( size (A) == size (B))
{
    result . insert ( $\frac{\text{max of } A + \text{min of } B}{2}$ );
}
else
    result . insert ( max of A);
```