Q: Sort the array [Heap Sort]

arr[] : [13, 14, 7, 6, 10, 2, 5, 8, 3, 1]

arr[] = [1, 2, 3, 5, 6, 7, 8, 10, 13, 14]

Min heap. O(n)

Extract min element till heap is empty.

extraction of 1 element takes O(log n)

⇒ T.C : n log(n)

T.C : n + n log(n)

n → creating

n log(n) → extracting all element

T.C = n log(n)

S.C = O(n)

Optimization [space complexity O(1)]

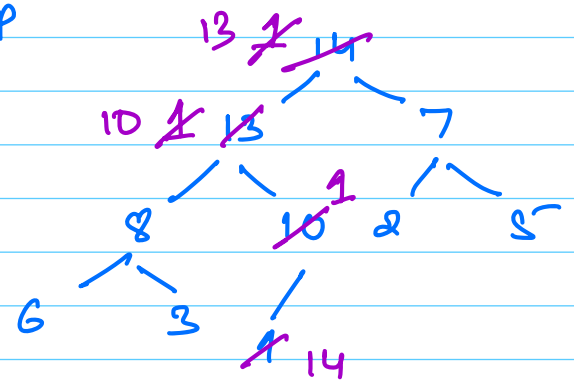
* Heap sort
O(1) spac

arr = [13, 14, 7, 6, 10, 2, 5, 8, 3, 1]

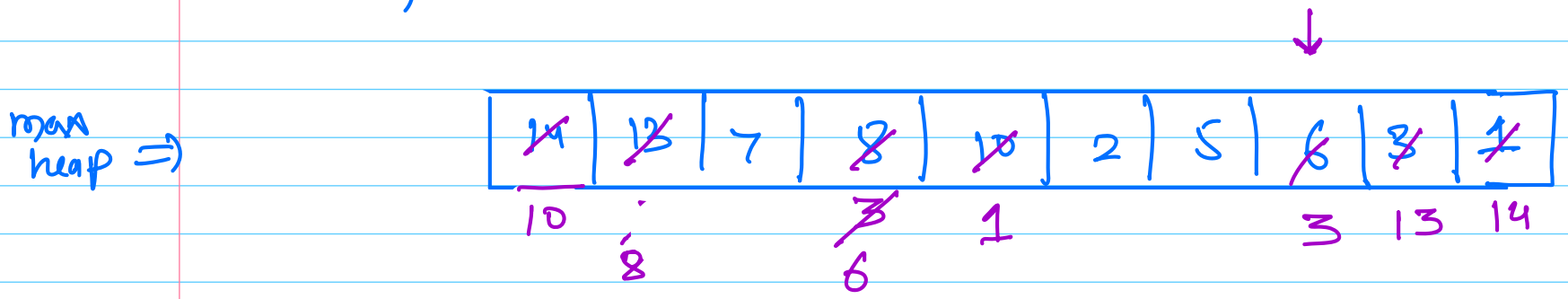
↓ Max heap : O(n)

arr = [14, 13, 7, 8, 10, 2, 5, 6, 3, 1] → max-heap. Put element in sorted orde inplace.

max-heap



ascending



Steps:

1) Replace root from last element (n-1)
2) heapify (heap, 0, n-2)
3) n = n-1

Pseudo:

```
Build max heap          → O(N)
j = N-1
while (j > 0) {
    swap ( A[0], A[j])
    j--;
    heapify ( heap, 0, j)
}
```

T.C: n + n log(n)

T.C = n log(n)
S.C = O(1)

Q. Given an array find the k^{th} largest element. ***

arr = [8, 5, 1, 2, 4, 9, 7]

$n \log n$

k = 3 = 7

k = 1 = 9

k = 2 = 8

Quiz [1, 2, 3, 4, 5] k = 5

ans = 1

Brute force : Sorting descending order
return k-1

T.C: $n \log(n)$.

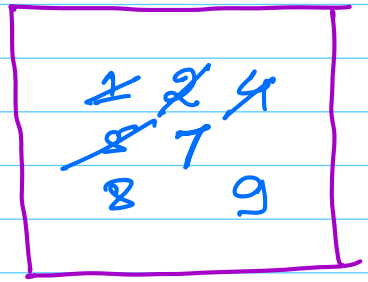
2): Heap sort : $k \log(n)$

arr = [8, 5, 1, 2, 4, 9, 7]

K = 3

min-heap.

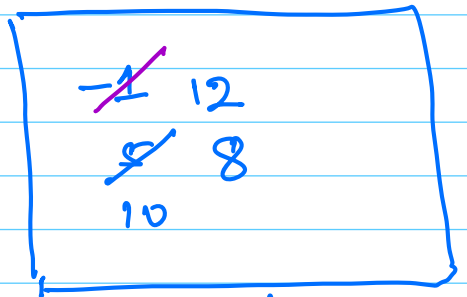
size = 3



⇒ min / root of heap is our answer.

⇒ arr = [5, 10, −1, 12, 8]

K = 3



minheap

ans = 8.

* to find k^{th} smallest element.

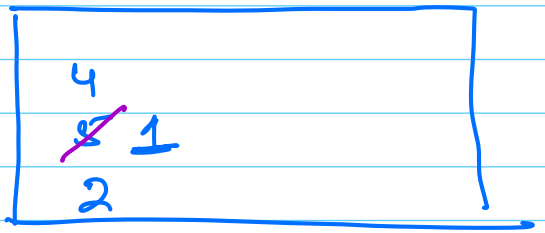
max-heap.

[4, 5, 2, 1, 10, 12, 14]

K = 3 smallest

T.C:

$n \log(k)$



max heap (3)

1) put first k element in heap.

ans = 4

Pseudo code

k^{th} largest element

Build min-heap for first k element

Iterate on the remaing elements :

for each element check.

```
if (cur_ele > min of heap)
{
    extract min() .
    insert (cur_element)
}
```

return . extract min()

T.C= n log(k) .

S.C: O(K) .

10:12 pm

Qu Find kth largest element for all the windows of an array starting from 0 index.

arr = [10, 18, 7, 5, 16, 19, 3]

K = 3

10, 18, 7	7
10, 18, 7, 5	7
10, 18, 7, 5, 16	10.
10, 18, 7, 5, 16, 19	16.
10, 18, 7, 5, 16, 19, 3	16.

ans = [7, 7, 10, 16, 16]

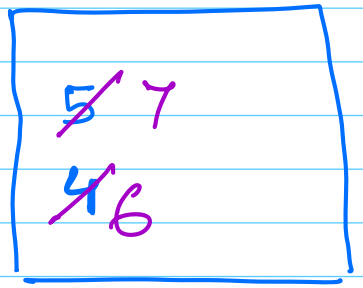
Quiz

[5, 4, 1, 6, 7]

k=2

5	4				4
5	4	1			4
5	4	1	6		5
5	4	1	6	7	6

ans: [4, 4, 5, 6]



[4, 4, 5, 6]

Pseudo

```
ans = [ ]
Build  min heap for first K elements
ans.add ( getroot() );
Iterate for the remaing elements:
        check if (cur_elem > getroot())
        {
            extract min(),
            insert (cur_element)
        }
        ans.add ( get root() )
}
```

Qus Given a nearly sorted array. You need to sort it.

nearly sorted array: Every element is shifted away from its correct position by almost k step.

Sorted. [11, 13, 20, 22, 31, 45, 48, 50, 60]

[13, 22, 31, 45, 11, 20, 48, 60, 50]

K = 4

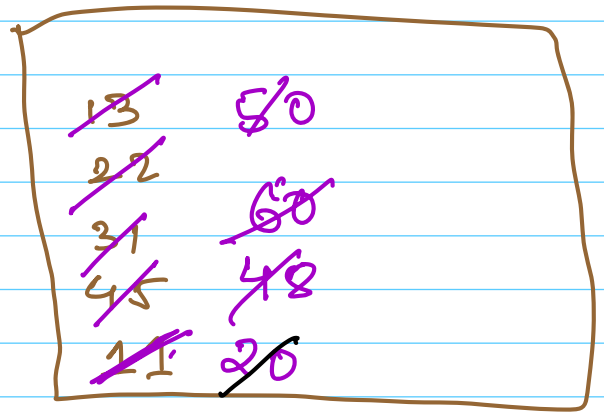
Brute force Sort the array → return.

T.C. n log n

[13, 22, 31, 45, 11, 20, 48, 60, 50]

K = 4

minheap.



Size = 5

[11, 13, 20, 22, 31, 45, 48, 50, 60]

Pseudo code

ans = [ ]

1. build min heap for first k+1 elements

2. for (i = k+1 ; i < N ; i++)
{
ans.append (extract min ( ))
insert (arr [i])
}

T.C: n log k
S.C: k

3: While (minheap not empty)
ans. appen (extract min ( ))

return ans.

Qu: median ⇒

[5, 4, −1, 2, 6]

1) sort.

[−1, 2, 4, 5, 6]

↓

median = 4

[−1, 2, 4, 5, 6, 7]

↳ median = $\frac{4+5}{2}$ = 4.5

1) odd → return middle
2) even → return middle avg.

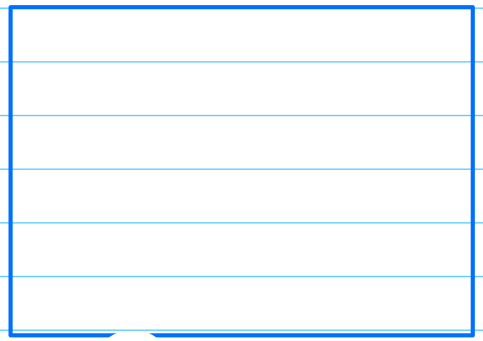
Quiz [1, 2, 3, 4] = $\frac{2+3}{2}$ = 2.5

Ques Given an infinite stream of number. Find the median of element of the current set.

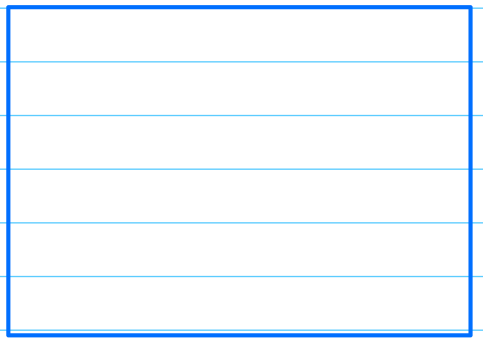
arr = [6, 3, 8, 11, 20, 2, 10, 8, 13, 50 ----]

leftside < rightside.

max-difference in size of both heap ≤ 1

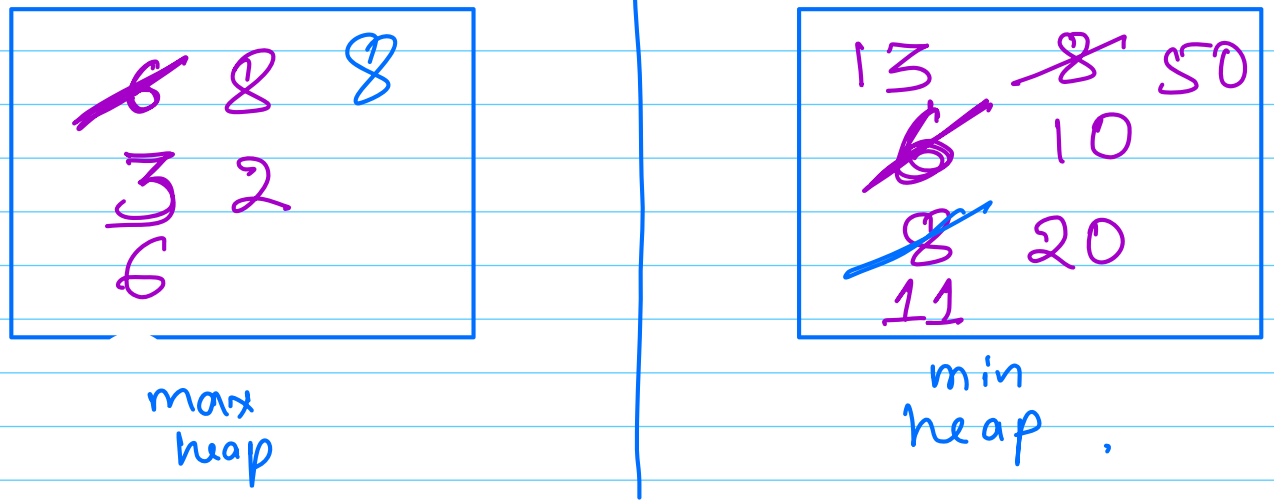


max heap



min heap.

arr = [6, 3, 8, 11, 20, 2, 10, 8, 13, 50 ----]



* max heap size == min heap size.

  ↳ even array ⇒ $\frac{getmax() + getmin()}{2}$.

* if size is not equal:

  return root of heap with size bigger.

pseudo code

```
arr = []

h1  ( max_heap)
h2  ( min_heap)

h1. insert ( arr[0])

for ( i  1 ⟶ n-1)
{
    if (arr[i] > h1.getmax())
        h2. insert (arr[i])
    else
        h1. insert (arr[i])

diff = abs(h1.size() - h2.size())
```

two heap.

inserting in right place

```
if (diff > 1)
{
    if (h1.size() > h2.size())
    {
        h2.insert(h1.getmax())
    }
    else
    {
        h1.insert(h2.getmin())
    }
}
if (h1.size() > h2.size())
{
    Print (h1.getmax())
}
```

Balancing the heap

odd case

```
elif ( h2.size() > h1.size() )
    print ( h2.getmin() )
else:
    print ( (h1.getmax() + h2.getmin()) / 2 )
```

even case.

T.C : N log(N)

S.C : O(N)