- Heap Sort
- Kth largest element
- Sort nearly sorted array
- Median of stream of integer

Ques Heap Sort :-

Sort an array

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

↓ sort

ans[] → _____

idea :- array → Build a min heap. → O(n)

↓

getMin & deleteMin → O(nlogn)

↓

ans[] array.

T.C → O(n) + O(nlogn) => O(nlogn)

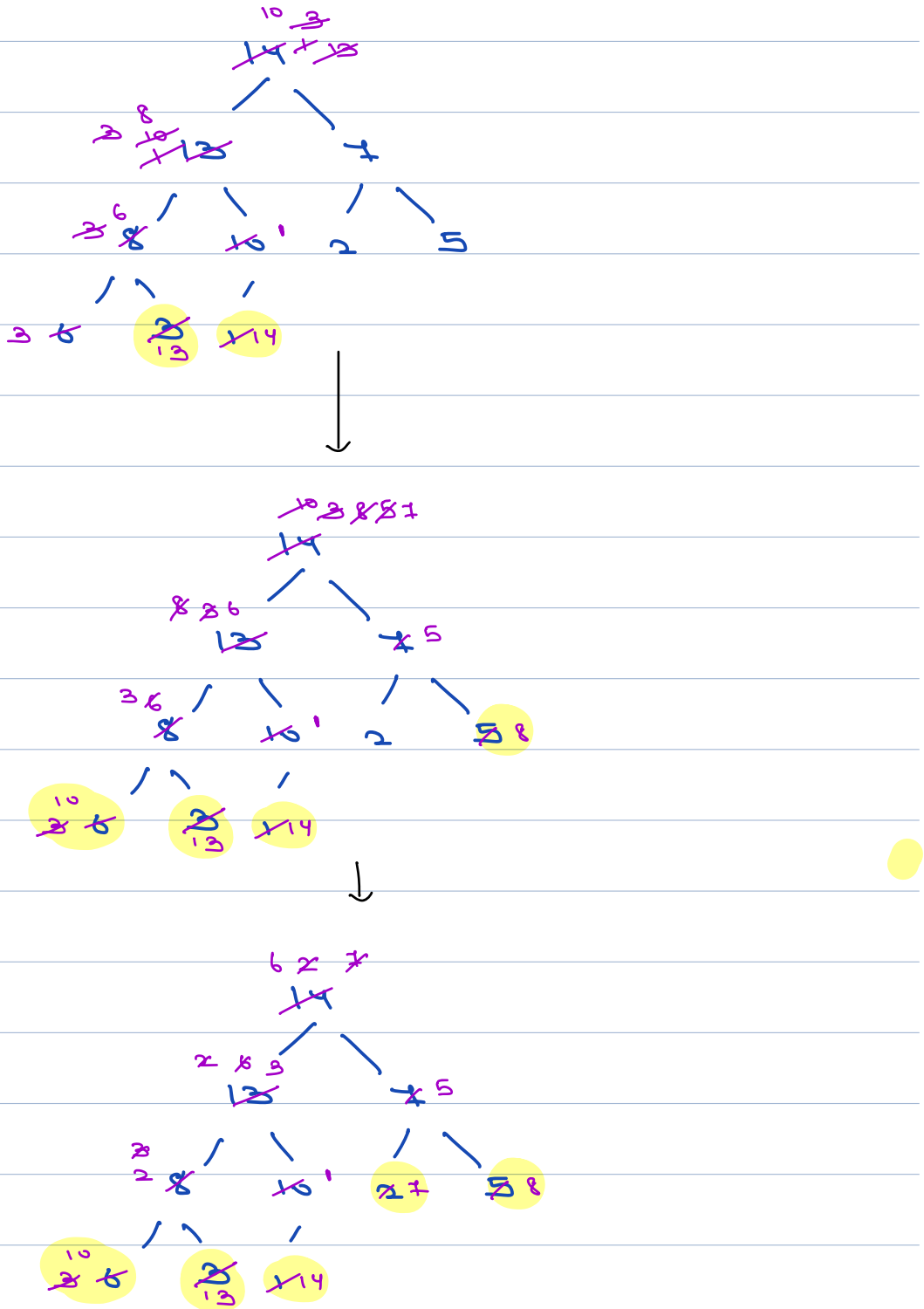
S.C → O(n) (Extra Space)

idea 2 :- Do it in place.

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

↓ max heap.

	0	1	2	3	4	5	6	7	8	9
ans[] →	14,	13,	7,	8,	10,	2,	5,	6,	3,	1



given array → arr[]

1) Build Min heap → O(n)

```
j = n-1;
while (j > 0) {                    → O(nlogn)
    swap(0, j);
    j--;
    heapify(0, arr, j);            → last valid idx
}                                  3
```

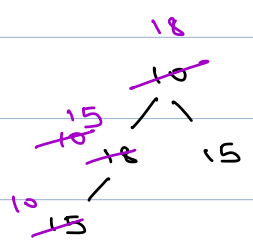
T.C → O(n) + O(nlogn) → O(nlogn)

S.C → O(1)

Heap Sort → In Place → S.C → O(1)

Stable → No.

10, 15, 15, 15



Ques :- arr[N], find k^{th} largest element.

e.g 1).

arr[] = [8, 5, 1, 2, 4, 9, 7]
k = 3

Ans :- 7.

- First largest element = 9
- Second largest element = 8
- Third largest element = 7

e.g 2)

arr[] → 1, 2, 3, 4, 5.

k = 5, → 1.

idea 1 :- Sorting & return arr[N-k]

e.g, k=3, arr[] → [8, 5, 1, 2, 4, 9, 7]

↓ sort

0 1 2 3 4 5 6
[1, 2, 4, 5, 7, 8, 9]

idea 2 :- Heap Sort :-

Build a max heap.

↓

do the heap sort step k-1 times.

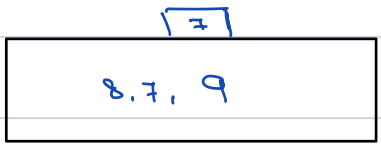
T.C → O(n) + O(klogn)

S.C → O(1).

idea 3 Min heap :-

k = 3.

8, 5, 1, 2, 4, 9, 7, 4



① Store first k Elements in a minheap (pq). → k log k

Iterate on remaining elements, → (n-k) log k

for every element, check

if (curr element > min element in heap)

extractMin()

insert (current elem).

3

3

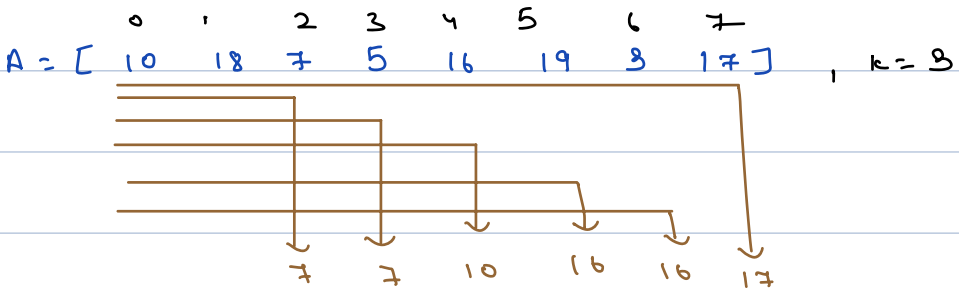
3

ans = getMin().

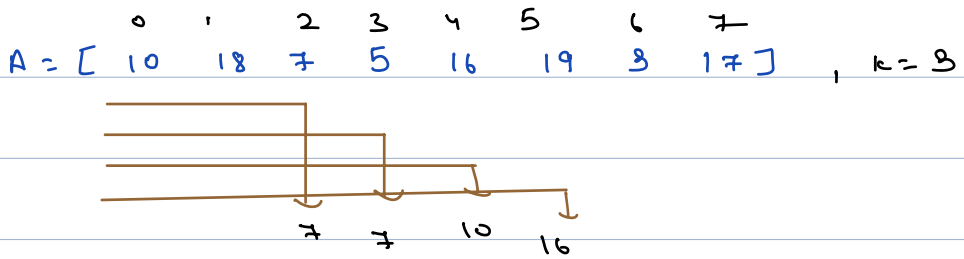
T.C → O(n log k)

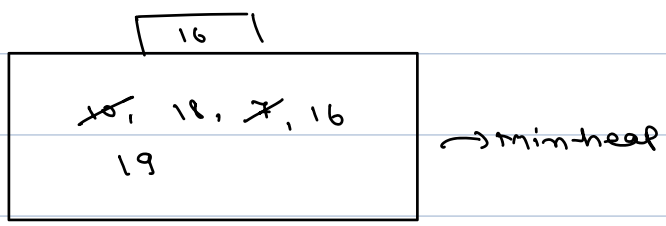
S.C → O(k)

Ques Given an integer array, $\forall\ i = (k-1)$, find kth largest element from $0 - i$.

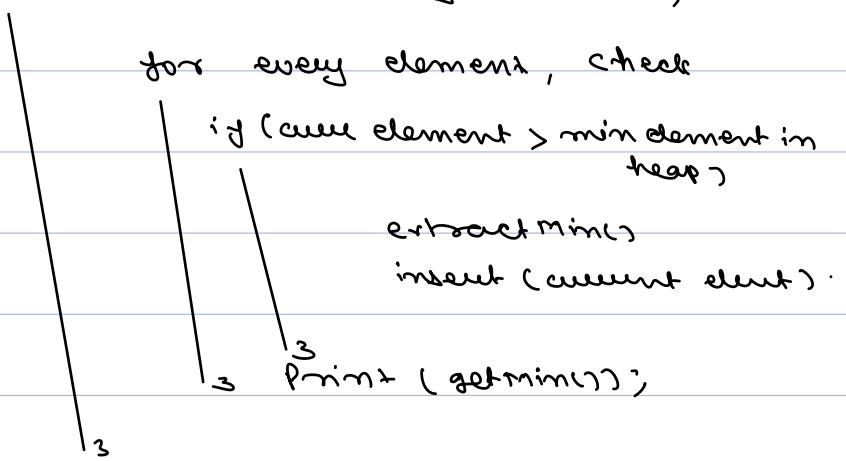


Idea :-





① Store first k Elements in a minheap (pq). → k log k
Print (getmin());
Iterate on remaining elements, → (n-k) log k



T.C → O(n log k)
S.C → O(k)

Ques Given an int[], & k, every element is at max k dist. away from its sorted pos^n, we have to sort the array.

↳ k << n

```
      0  1  2  3  4  5  6
A :  [ 3, 1, 5, 6, 2, 7, 4 ]      k = 3
```

```
A =  [ 1, 2, 3, 4, 5, 6, 7 ]
```

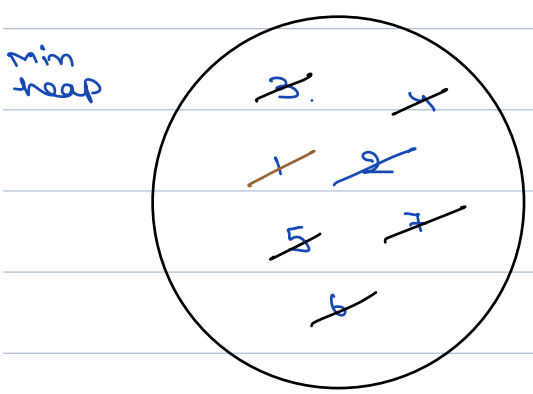
T.C → O(nlogk)

idea :- Sort the array.

T.C → O(nlogn).

idea 2 :-

```
      0  1  2  3  4  5  6
A :  [ 3, 1, 5, 6, 2, 7, 4 ]      k = 3.
      1  2  3  4  5  6  7
```



0 → 0 to 3
1 → 0 to 4
2 → 0 to 5
3 → 0 to 6
4 → 1 to 6
5 → 2 to 6
6 → 3 to 6

1) pq → minheap.

2) for i → 0 to n-1

pq.add(arr[i]);

if (pq.size() > k) {

next → pq.getmin(); // smaller elem

pq.remove();

}

}

when loop ends remove all rest elements one by one .

T.C → O(nlogk)

S.C → O(k)

Ques Given a running stream of integers, find median for all input.

{ 5, 10, 2, 1, 43 } → { 1, 2, 4, 5, 103 } → 4

{ 5, 10, 2, 3, 1, 43 } → { 1, 2, 3, 4, 5, 103 } → $\frac{3+4}{2}$ => 3.5

I/P → 9 8 17 20 25 10 5 3

9 8.5 9 13 17 13.5 10 9.5

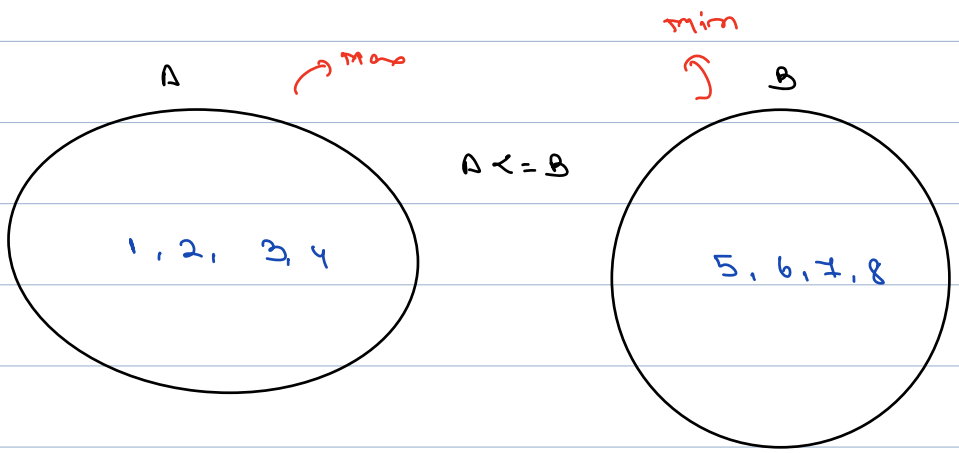
Idea 1:- use insertion sort.

T.C → O(n²)

S.C → O(1)

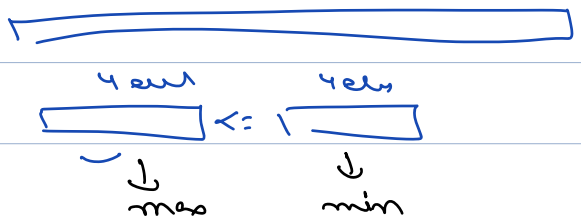
idea 2 :- 1, 5, 7, 4, 3, 6, 2, 8

sizeof(A) - sizeof(B) = 0



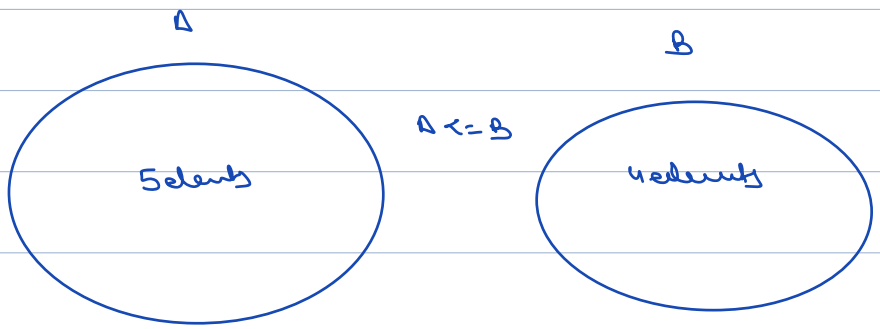
median = $\frac{\text{max of A + min of B}}{2}$

8 elements



Case 2 when odd elements :-

9 elements



median = max of A.

5th element.

size of (A) - size of (B) = 1

A <= B

1) Even

size(A) - size(B) = 0

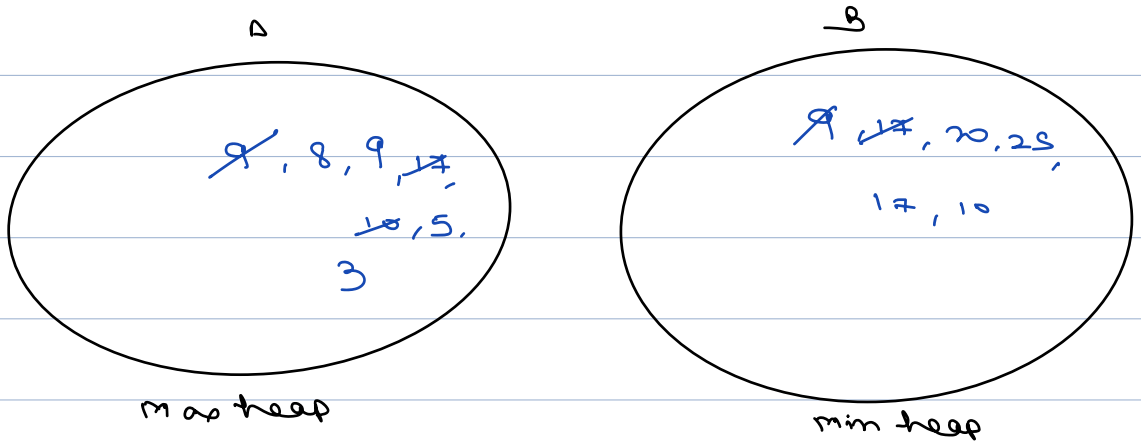
median = $\frac{max(A) + min(B)}{2}$

2) Odd

size(A) - size(B) = 1

median = max(A)

I/P →	9	8	17	20	25	10	5	3
o/p	9	8.5	9	13	17	13.5	10	9.5



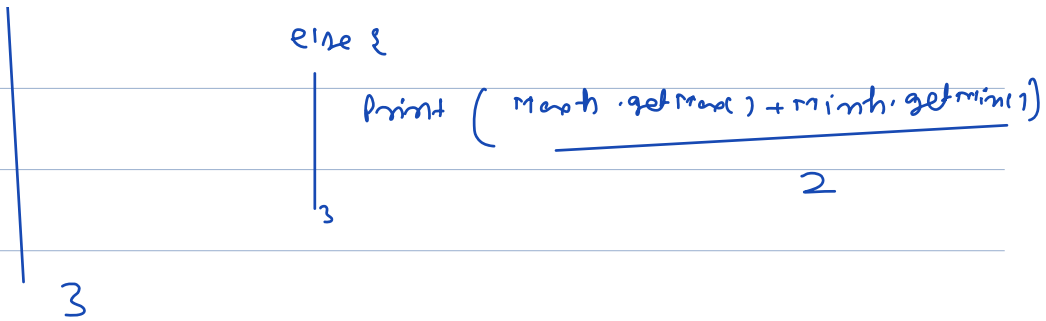
```
void running_median (int arr[]) {
    Max heap <int>  maxh;
    Minheap <int>  minh;
    maxh.insert (arr[0]);
    print (arr[0]);
    for (i = 1; i<n; i++) {
        ele = arr[i];
        if (ele < maxh.getMax()) {
            maxh.insert (ele);
        }
        else {
            minh.insert (ele);
        }
        if (maxh.size() - minh.size() > 1) {
            Transfer 1 element from maxh to minh;
        }
        if (maxh.size() - minh.size() < 0) {
            Transfer 1 element from minh to maxh;
        }
        int s = maxh.size() + minh.size();
        if (s%2 == 1) {
            print (maxh.getMax());
        }
```

log n

log n

log n

log n



T.C → O(nlogn)

S.C → O(n).

4, 3, 6, 1, 10

1, 2, 4, 3, 5

There are **N** poles, and the height of the poles is given by an integer array **A**.
You have to connect two poles such that all the poles between them have a height **smaller** than the minimum height of the two poles.
Find the total number of pairs of such poles.

NOTE:

The heights of the given poles are distinct.

i J

arr[k] < min(arr[i], arr[j]) (i < k < j)

Orig → 1, 2, 3, 4

curState → 1, 2, 3, 4

↓

2, 2, 3, 4

↓

3, 2, 3, 4

↓

4, 2, 3, 4