Heaps 2 : Problems

TODAY'S CONTENT

- HEAP SORT
- Kth Largest Element
- Sort Nearly Sorted Array
- Median of Stream of integer

Question:- Sort An Array [HEAP SORT]

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

↓ Sort

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

IDEA:- Array → Build Min Heap → getMin() & deleteMin()

Build Min Heap: $O(n)$

getMin() & deleteMin(): $O(n \log n)$

Quiz!:- what is the time complexity to convert an array to a heap?

↳ $O(n)$

TC → $O(n \log n)$

SC → $O(n)$

Quiz? :- Root element in a heap is ?

↳ Either min or max depending upon whether it is min or max heap.

⇒ OPTIMIZATION I Space Complexity → O(1)

IDEA? :- Do it inplace

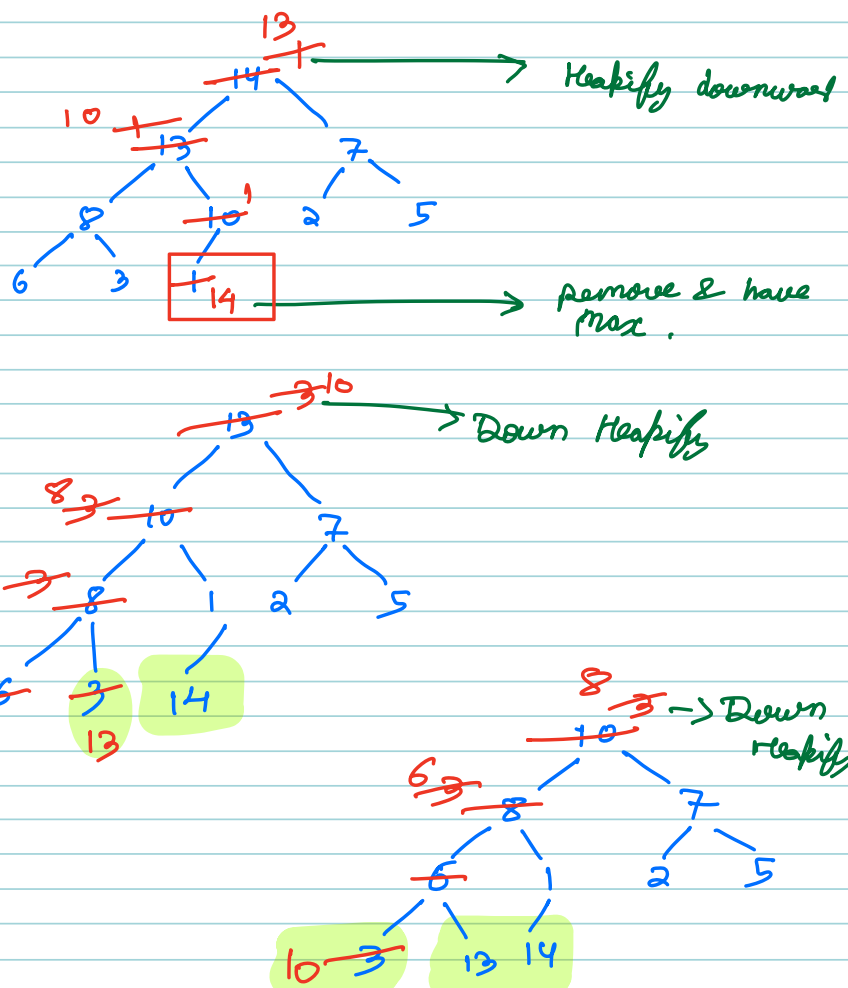
Sort it in ascending order

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

Ex :-



In this way the largest element will collect toward end & we will get sorted array.

PSEUDO CODE

```
Given arr → arr[]
① Build Max Heap → O(n)
② J = N-1
   while (J > 0){
       Swap (0, J);
       J--;
       Heapify (arr, 0, J);
   }
```

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

↓ Max heap: O(n)

arr[] → ~~14~~ 1, 13, 7, 8, 10, 2, 5, 6, 3, ~~1~~ 14 (J)

TC — O(n) + O(n log n)

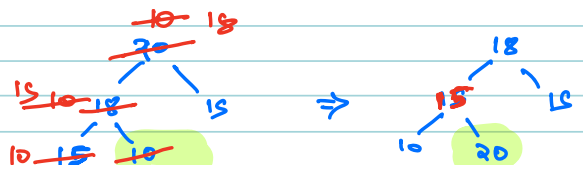
SC — O(1)

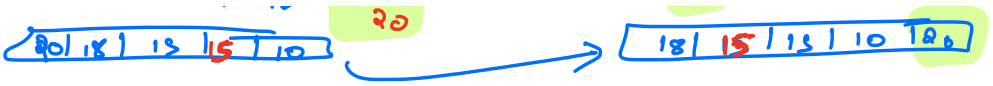
Q Is HEAP SORT Inplace Sort?

↳ yes

Q Is HEAP SORT Stable?

↳ Not a stable





Question :- Given arr[N], find the K^{th} largest element.

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

K = 3

→ First largest = 9
→ Second largest = 8
→ Third largest = 7 → Ans

Qus?:- What is the 5^{th} largest element of an array [1, 2, 3, 4, 5]

↳ 1

Brute force :- Sort & return $(N-K)^{th}$ element

TC → O(N log N)

IDEA 2 :- Heap Sort

↳ Build a max Heap & Apply K steps of Heap Sort.

TC → ~~O(n)~~ + O((K-1) * log N)

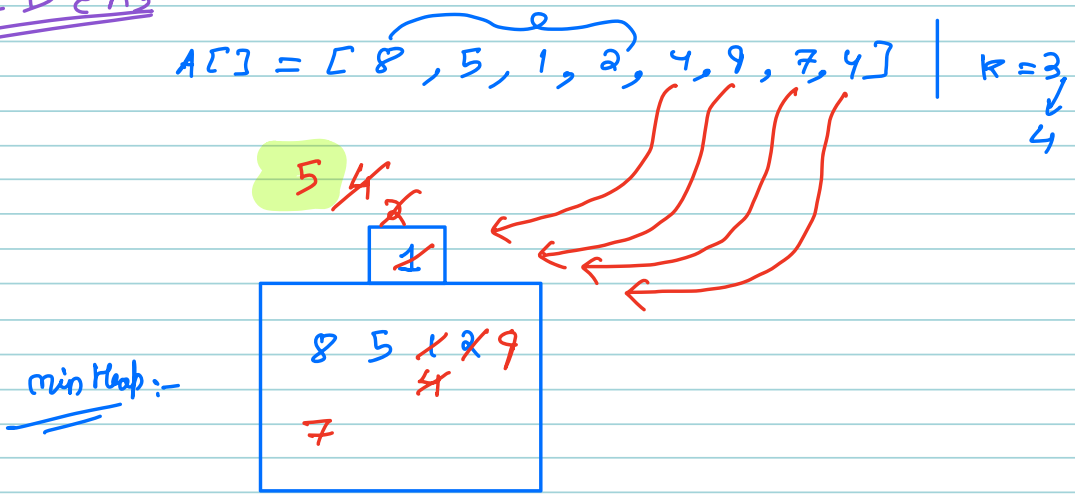
≈ O(K log N)

SC — O(1)

↳ OPTIMIZATION

IDEA:

A[] = [8, 5, 1, 2, 4, 9, 7, 4] | K = 3



* By doing this we are able to maintain K largest element from 0 to ith position

PSEUDO CODE

① Store First K element [0 to K-1] in Min Heap. — TC - O(K)

② For each element from (K to N-1):- — TC - (n-k) log K

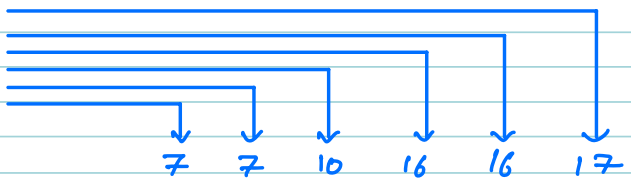
```
if ( currElement > MinHeap.top ){
    extract min();
    insert ( currElement, Heap);
}
```

return Heap.top();

TC — O(n log K)
SC — O(K)

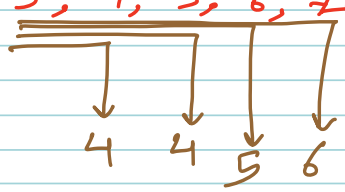
Question :- Find K^{th} largest element for all windows of an array starting from 0 index.

	0	1	2	3	4	5	6	7	
A[] = [	10	18	7	5	16	19	3	17]	K = 3



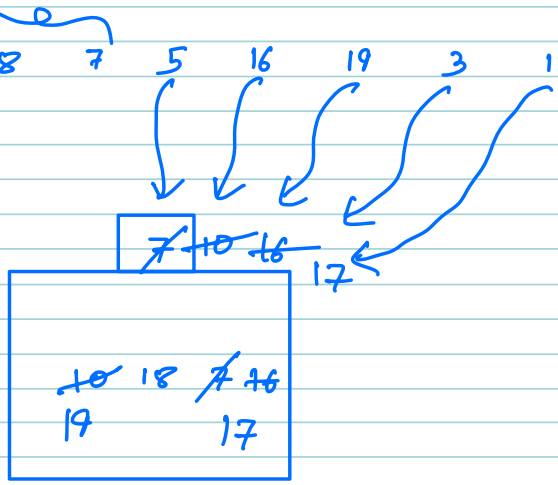
Quiz4 :- Find K^{th} largest element for all windows of an array starting from 0 index.

arr[] = [5, 4, 3, 6, 7] | K = 2



DRY RUN

A = [10 18 7 5 16 19 3 17] | K = 3



Ans = [7 , 7 , 10 , 16 , 16 , 17]

PSEUDO CODE

① Store First k element [0 to k-1] in Min Heap.

Print (Heap.top());

② For each element from (k to N-1):-

```
if ( currElement > minHeap.top ){
        extract min();
        insert ( currElement, Heap);
}
Print ( Heap.top());
```

Question :- Given a nearly sorted array. You need to sort an array.

Nearly Sorted Array :- Every element is shifted away from its correct position by atmost k-steps.

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

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

Brute force :- Sort the given Array.

↳ OPTIMIZATION

↳ Using Heap, as question is not total unsorted but unsorted elements can be k distance away only

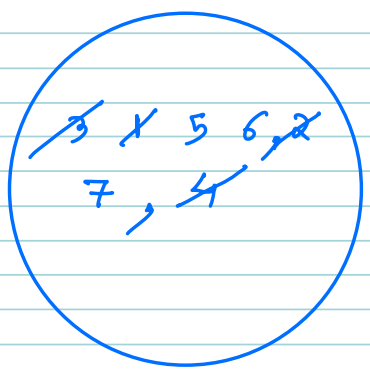
DRY RUN

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

(indices: 0 1 2 3 4 5 6)

Ans = [1 , 2 , 3 , 4 , 5 , 6 , 7]

Min Heap :-



0th Index → 0 to 3

1st Index → 0 to 4

2nd Index → 0 to 5

3rd Index → 0 to 6

4th Index → 1 to 6

5th Index → 2 to 6

6th Index → 3 to 6

Q what is the max size of Heap at any instance.

↳ k+1

PSEUDO CODE

```
Pq → min heap()
for (i → 0 to n-1) {
        pq.add(A[i]);
        if (i >= k) {
                Print(pq.top());
                pq.poll();
        }
}
```

getMin() from heap till its emptg.

TC — O(n log(k))
SC — O(k)

Question :-

Flipkart is currently dealing with the difficulty of precisely estimating and displaying the expected delivery time for orders to a specific pin code.

The existing method relies on historical delivery time data for that pin code, using the median value as the expected delivery time.

As the order history expands with new entries, Flipkart aims to enhance this process by dynamically updating the expected delivery time whenever a new delivery time is added. The objective is to find the expected delivery time after each new element is incorporated into the list of delivery times.

End Goal: With every addition of new delivery time, requirement is to find the median value.

Why Median ?

The median is calculated because it provides a more robust measure of the expected delivery time

The median is less sensitive to outliers or extreme values than the mean. In the context of delivery times, this is crucial because occasional delays or unusually fast deliveries (outliers) can skew the mean significantly, leading to inaccurate estimations.

Problem statement :- Given a running Stream of Integer, Find Median for all input.

{5, 10, 2, 1, 4} Median → {1 2 4 5 10} → 4

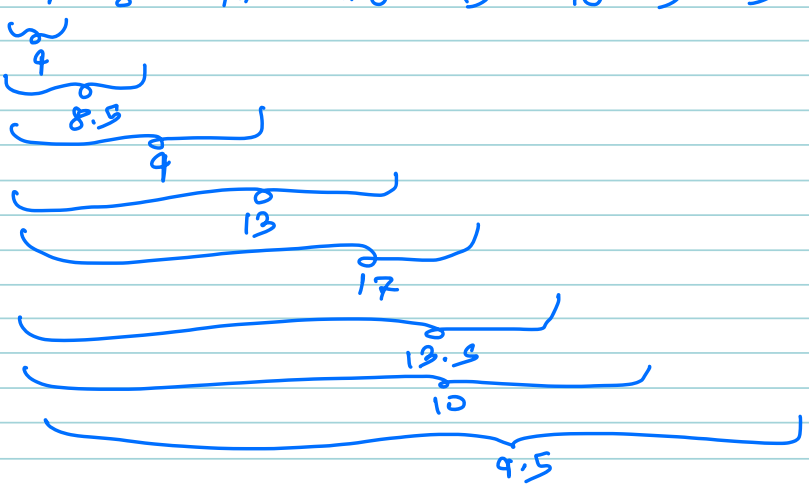
{5, 10, 2, 3, 4} Median → {1 2 3 4 5 10}

↳ $\frac{3+4}{2} = 3.5$

Quiz :- The Median of [1, 2, 4, 3]

↳ [1 2 3 4] ⇒ $\frac{2+3}{2} = 2.5$

I/p → 9 8 17 20 25 10 5 3

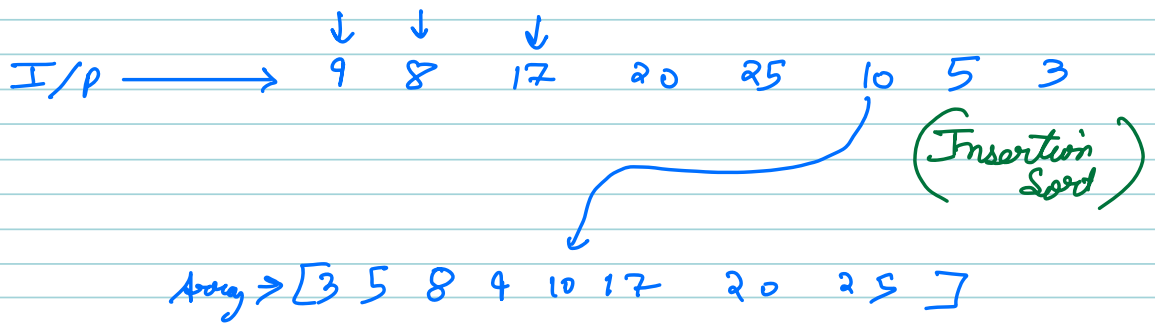


Brute force :- For every Incoming element, Sort the resultant array.

TC — $O(N^2 \log N)$

↳ OPTIMIZATION

IDEA :- Try to place Each incoming element at its correct position in Sorted array.



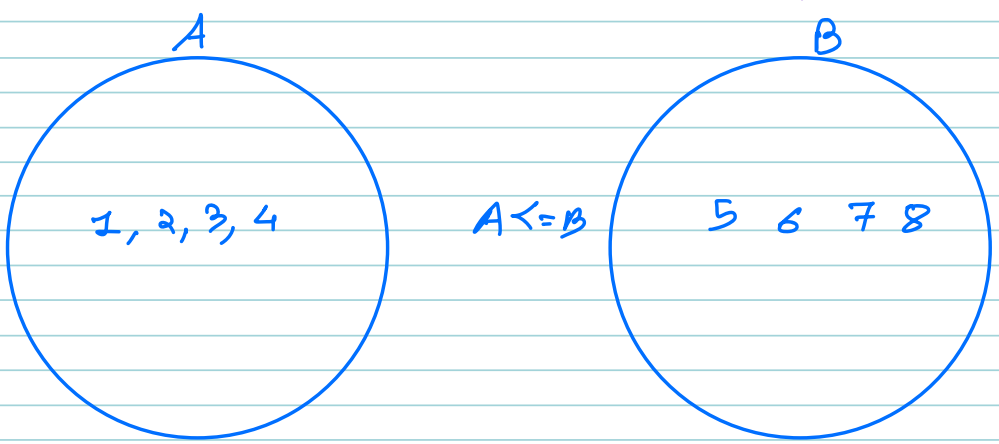
TC — $O(N*N)$

IDEA 3:-

Case I :- Even No. of elements.

1, 5, 7, 4, 3, 6, 2, 8

↳ median $\Rightarrow$ $\frac{4^{th} \text{ element} + 5^{th} \text{ element}}{2}$



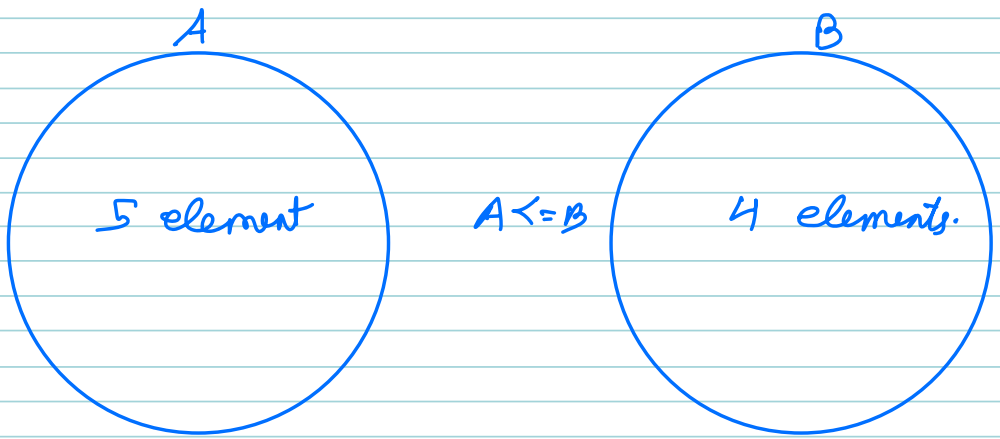
* we will split according to condition:-

All element in A <= All element in B

Q If we have splitted, then what will be median :-

↳ $\frac{\max(A) + \min(B)}{2}$

Case 1 :- odd no. of elements. [9 elements]



Median = Max(A)

Conclusion :- ① Elements in A <= Elements in B

② Even Case :- Way to figure out :-

↳ Size(A) - Size(B) = 0

↳ Median = $\frac{Max(A) + Min(B)}{2}$

③ Odd Case :- Way to figure out

↳ Size(A) - Size(B) = 1

↳ Median = Max(A)

DRY RUN

I/P :— 9 8 17 20 25 10 5 3

Q which kind of Heap should be A & B

A → Max Heap B → Min Heap.

Q How I can figure out in which heap it should go?

↳ Compare with Max of A

Q:- How we will make sure elements diff b/w two Heaps should be atmost 1.

↳ If diff in heap size >1, then move from larger size Heap to smaller size Heap.

I/P :— 9 8 17 20 25 10 5 3

A (Max Heap) B (Min Heap)

~~9~~, 8, ~~9~~, ~~10~~, 5, 3 A <= B ~~9~~, 17, 20, ~~25~~, ~~10~~, 10

Ans = [9 , 8.5, 9 , 13 , 17, 13.5, 10 , 9.5]

→ Insert
→ Balancing
→ Find Median

PSEUDO CODE

```
Void runningMedian ( int arr[] ){

    MaxHeap maxh;  ⟶ A
    MinHeap minh;  ⟶ B
    maxh.insert (arr[0]);
    Print (arr[0]);
    for ( i = 1 ; i < n ; i++ ){
        int ele = arr[i];
        if ( ele < maxh.getMax() ){
            maxh.insert (ele);
        } else {
            minh.insert (ele);
        }

        if ( maxh.size() - minh.size() >= 2 ){
            // Transfer maxh ele from maxh to minh.
        } else if ( minh.size() - maxh.size() >= 2 ){
            // Transfer min from minh to maxh
        }

        int s = minh.size() + maxh.size()
        if ( s % 2 == 1 ){
            // Ans will be from heap with large size.
```

Insert ←

Balance ←

Find Median.

```
} else {
    Print ( maxh.getMax() + minh.getMin() / 2 )
}
}
```

TC — O(n log n)

SC — O(n)