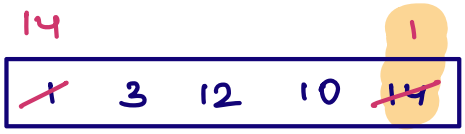
HEAPS 2

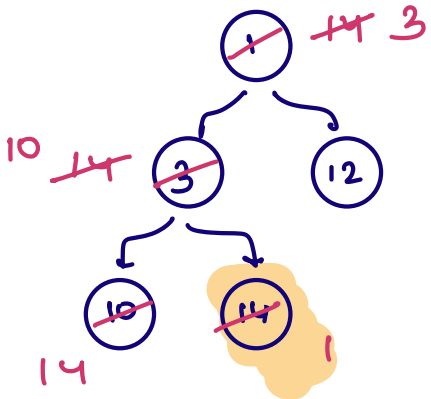
Good Evening

Today's content

01. Heap Sort
02. K^{th} largest element
03. Sort nearly sorted array
04. Median of stream of integers

Sort the array





01. Build a min heap

⇓

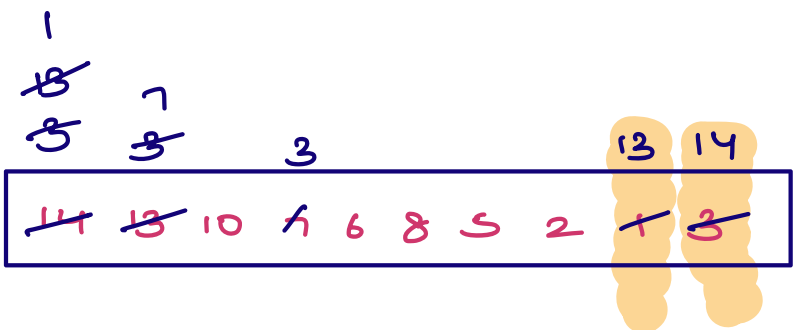
Extract min → ans array

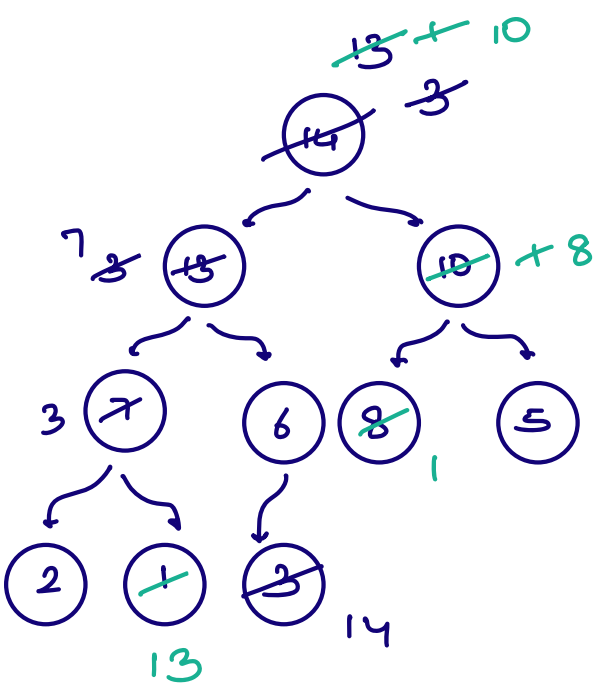
TC : O(n + nlogn) ≈ O(nlogn)

SC : O(n)

Optimise the SC : O(1)

02. Use max heap





Heapsort : TC : O(n + nlogn)

SC : O(1)

Heap Sort

01. Build a max heap → O(n)

```
j = N - 1

while ( j > 0) {
    swap ( ar[0], ar[j] )
    j = j - 1
    heapify (ar, 0 , j )
}
```

O(nlogn)

* Is heap sort an inplace sorting algo? Yes

* Is heapsort a stable sorting algo? No

HeapSort	Quick Sort	Merge Sort
TC: O(nlogn)	TC: O(nlogn)	TC: O(nlogn)
SC: O(1)	SC: O(1)	SC: O(n)

Q1 Given arr[N]. Find K* largest ele

arr[] = { 8 5 1 2 4 9 7 }
indices: 0 1 2 3 4 5 6

K=2 → 8

K=5 → 4

sort[] = { 1 2 4 5 7 8 9 }
indices: 0 1 2 3 4 5 6

01. Sort the array & return arr[n-k]

TC: O(nlogn)

02. Binary Search

TC: O(nlog(max-min))

03. Using a max heap

01. Build a max heap → O(n)

02. Extract the max K times

klogn

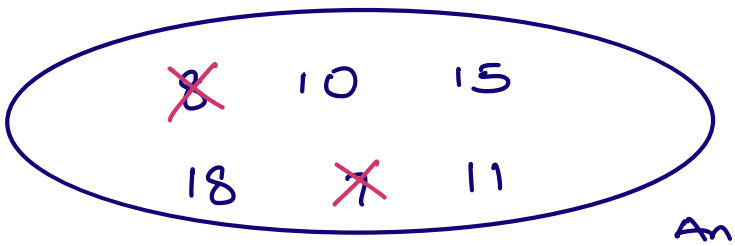
TC: O(n + klogn)

SC: O(1)

04. Using min heap

Select 4 badsmen

b_1	b_2	b_3	b_4	b_5	b_6	b_7
↓	↓	↓	↓	↓	↓	↓
8	10	15	7	11	3	18

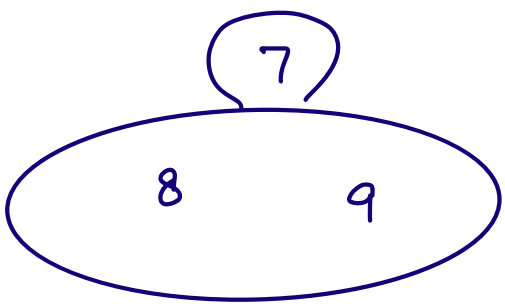


min heap

Ans = min.getfirst();

8	5	1	2	4	9	7	1

k = 3



minheap

peak ele → k' largest ele

→ Build a minheap of size k with first k ele

→ Iterate on remaining n-k ele

O(k)

```
for (i = k; i < n; i++)
    if (curr ele > peek ele of min heap)
        extractmin();
        insert curr ele:
}
}
return minheap.extractmin();
```

TC: $O(k + (n-k)\log k)$

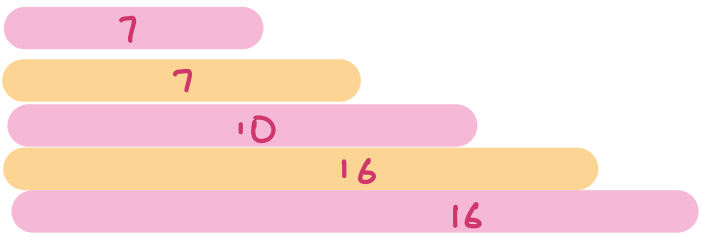
SC: $O(k)$

Q Find k^{th} largest ele for all the windows from

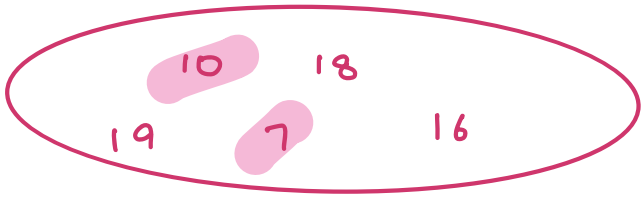
0 to i &

$i \geq k-1$

arr[ ] = { 10 18 7 5 16 19 3 } k=3



Min heap



Ans = 7 7 10 16 16

Q Given a nearly sorted array. Sort the array

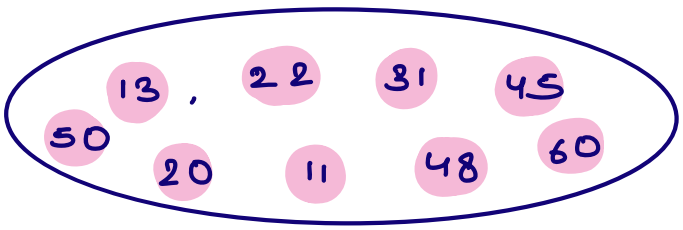
Every ele is shifted away from its correct position by atmost K - steps

K=4

	0	1	2	3	4	5	6	7	8
arr [ ] = {	13,	22,	31,	45,	11,	48,	20,	60,	50}

Sort [ ] = { 11, 13, 20, 22, 31, 45, 48, 50, 60}

* Extract the min ele → min heap ⟶ K+1



min heap

Ans: 11, 13, 20, 21, 31, 45, 48, 50, 60

* Pseudocode

01. Build min heap of k+1 elements

02. Iterate on remaining ele

    ↳ extractmin() & print it.

    ↳ add curr ele

03. while (heap.size() > 0)

    ↳ extractmin() & print it.

TC: $O(n-k-1 \log(k+1))$

$\simeq O(n-k \log k)$

SC $\simeq O(k)$

10:10 → 10:20 pm

Running median

Q Given an infinite stream of integers. Find the median of current set of elements.

median → middle element in sorted array

arr		median
6	⟶	6
6, 3	⟶	4.5
6, 3, 8	⟶	6
6, 3, 8, 11	⟶	7
6, 3, 8, 11, 20	⟶	8

Idea 1 → For every element, insert & sort the array

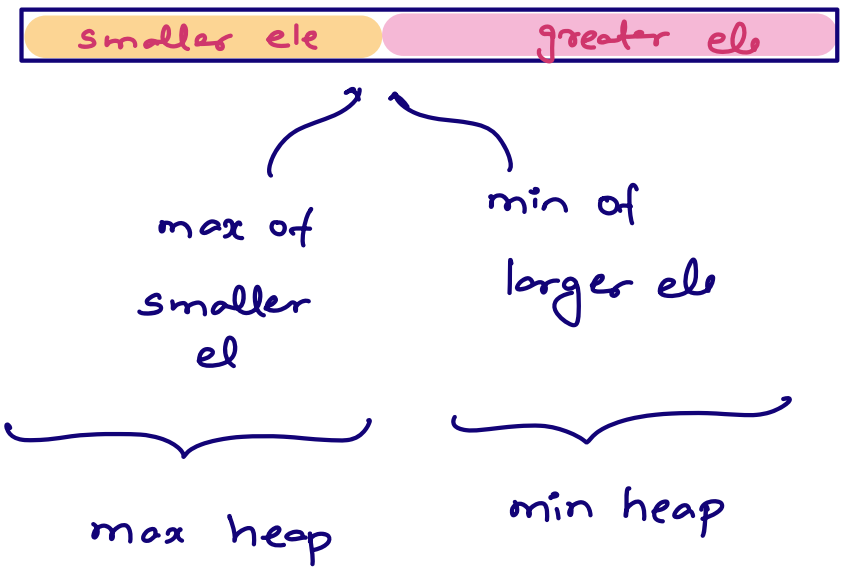
→ middle ele / avg of two middle ele

TC: O(n * nlogn)

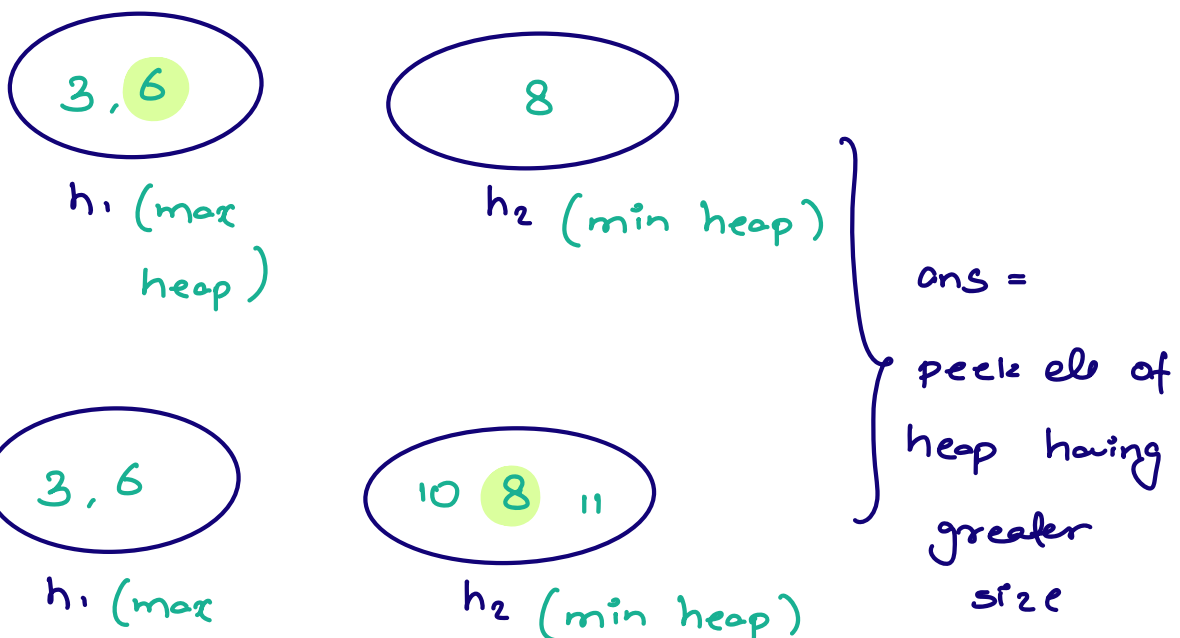
Idea 2 → Use insertion sort

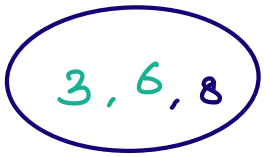
TC: $O(n^2)$

Idea 3

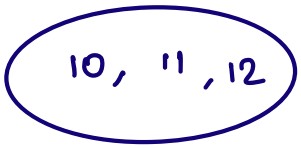


6 3 8





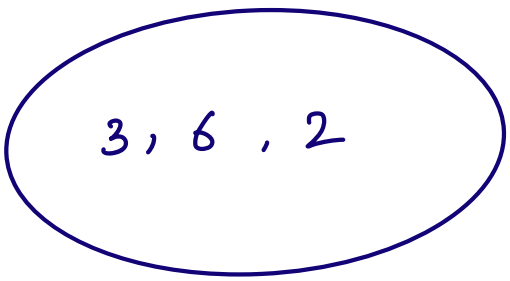
h_1 (max heap)



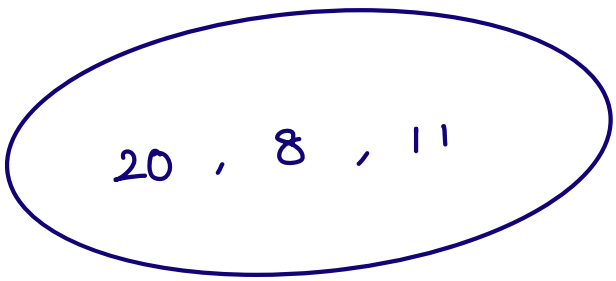
h_2 (min heap)

$$ans = \frac{\text{max of } h_1 + \text{min of } h_2}{2}$$

arr[] = { 6, 3, 8, 11, 20, 2, 10, 18, 3 ... }



h_1 (max heap)



h_2 (min heap)

Ans: 6, 4.5, 6, 7, 8, 7

h_1 = max heap

h_2 = min heap

h1.add(ar[0]);

```
for (i=1 ; i<n; i++)
{
    if (ar[i] > h1.getmax()){
        h2.add(ar[i]);
    }
    else {
        h1.add(ar[i]);
    }
    int diff = |h1.size() - h2.size()|
    if (diff > 1) {balance(h1, h2)};
    if (h1.size() > h2.size()) print(h1.getmax());
    if (h2.size() > h1.size()) print(h2.getmin());
    else {
        print( (h1.getmax() + h2.getmin()) / 2.0 );
    }
}
```

TC: O(nlogn)

SC: O(n)