* CBT (Complete Binary Tree)
* **Heap** :-
  - ↳ for every node value >= Both children
    - ↳ MAX HEAP
  - ↳ for every node value <= Both children
    - ↳ **MIN HEAP**.

Functions :-

1) insert() ⇒ $O(\log N)$
2) getMin() | getMax() ⇒ $O(1)$
3) deleteMin() | deletMax() ⇒ $O(\log N)$

⇒ **Heap DS**

Q. Given N distinct array elements. Find K smallest elements. K < N

A[10] : { 8, 3, 10, 4, 11, 2, 7, 6, 5, 1 }

K = 4

{ 1, 2, 3, 4 }

A[8] : { -3, 6, 2, 0, 8, 7, 10, 4 }

K = 3

⇒ { -3, 0, 2 }

Approach 1:-

⇒ In every iteration, get the smallest element & swap it with ith index. Repeat this for k times.

TC : $O(K \cdot N)$

SC : $O(1)$

Approach 2:- Sorting

TC : $O(N \log N)$

SC : Depends on sorting Algo.

Approach 3: Min Heap

i) Create a Min Heap. ⟶ $O(N \log N)$

ii) getMin()
deleteMin() } K times.

TC: $O(N\log N + K + K\log N)$

SC: $O(N)$
↳ Heap creation (Array)

Approach 4:- Max Heap

A[10] : {8, 3, 10, 4, 11, 2, 7, 6, 5, 1}

k = 4

~~8~~, 3, ~~10~~,
~~7~~, 4, 2
~~6~~ ~~5~~ 1

Max Heap

1) Create Max Heap of first (K) elements.

2) • If ele > getMax() ⇒ ele can't be the ans.

• If ele < getMax() ⇒ ele can be the ans.

↳ insert (ele)

→ deleteMax()

TC :- $O(K \log K + (N-K) \log K)$

$O(N \log K)$

SC : $O(K)$

MEDIAN :-

A[5] : { 2, 9, 6, 4, 5 }

~~↑~~

Sort

{ 2, 4, 5, 6, 9 }

$\underbrace{2, 4}_{1^{st}\ \text{Half}}, 5, \underbrace{6, 9}_{2^{nd}\ \text{Half}}$

1st Half < 2nd Half

A[6] : { -1, 10, 3, 6, 9, 2 }

↓ Sort

{ -1, 2, $\underbrace{3, 6}$, 9, 10 }

Avg of 3 & 6 = $\frac{9}{2}$ = 4

Q: Given an Array, find the median of all Prefix subarrays.

Subarrays starting from index = 0.

A[5] : { 9 , 6 , 3 , 10 , 4 }

Prefix subarrays	Median
{9}	9
{9, 6}	$\frac{6+9}{2} = \frac{15}{2} = 7$
{9, 6, 3}	6
{9, 6, 3, 10}	$15/2 = 7$
{9, 6, 3, 10, 4}	6

Approach 1:-

Sort every Prefix subarray & get the median.

TC : O(N × NlogN)

SC : O(N)

Approach 2 :-

A[5] : { 9 , 6 , 3 , 10 , 4 }

↑ Median

i) { 9 } ⇒ 9

ii) { 6 , 9 } ⇒ 15/2 = 7

iii) { 3 , 6 , 9 } ⇒ 6

iv) { 3 , 6 , 9 , 10 } ⇒ 7

v) { 3 , 4 , 6 , 9 , 10 } ⇒ 6

⇒ Add the new element at it's correct position in previous sorted prefix subarray.

⇒ TC :- $O(N^2)$

SC :- $O(1)$

Optimized approach :-

Odd size

A[9] : { 3 , 1 , 6 , 10 , 14 , 2 , 17 , 12 , 9 }

↓ SORT

{ 1 , 2 , 3 , 6 , 9 , 10 , 12 , 14 , 17 }

1st Half | ↑ Median | 2nd Half

1st Half elements < 2nd Half elements.

Observation 1:-

All the elements in 1st Half < All the elements in 2nd

→ Max element in 1st < Min element of 2nd Half.

⇒ Include median in the 1st Half.

{1, 2, 3, 6, 9, 10, 12, 14, 17}

2) Size of 1st Half − Size of 2nd Half = 1

3) Median = max of the 1st Half.

Even Size

A[10] : {3, 4, 16, 12, 10, 14, 8, 9, 2, 1}

Sort

{1 2 3 4 8 9 10 12 14 16}

1st Half 1st median 2nd median 2nd Half

1) Max element in 1st < Min element of 2nd Half.

2) Size of 1st Half − Size of 2nd Half = 0

3) Median = $\dfrac{\text{Max of 1st half} + \text{Min of 2nd Half}}{2}$

A : { 4, 9, 6, 2, 1, 10, 9, 7, 3, 5 }

	9	6	2	1	10	9	7	3	5
	> max of 1st half	> max of 1st half	< max of 1st half	< max of 1st half	> max of 1st half	> max of 1st half	> max of 1st half	< max of 1st half	< max of 1st half

1st Half	2nd Half.
4, 2, 1, 3, 5	9, 10, 9, 7, 6

insert()
getMax()
deleteMax()

Max Heap

insert()
getMin()
deleteMin()

Min Heap.

Median :- $4, \frac{4+9}{2}, 6, \frac{4+6}{2}, 4, \frac{4+6}{2}, 6, \frac{6+7}{2}, 6, \frac{5+6}{2}$

Note :-

1) Max of 1st half < Min of 2nd half.
2) Size of 1st half - Size of 2nd half = 0 or 1
3) if (size(1st) == size(2nd))

$$\text{median} = \frac{\max(1^{st}) + \min(2^{nd})}{2}$$

else

$$\text{median} = \max(1^{st})$$

Code :-

```
int[] running_median(int ar[], int N){
    int ans[N]
    MaxHeap <int> maxH;
    MinHeap <int> minH;
    maxH.insert(ar[0]);
    ans[0] = ar[0];
    for(i=1; i<N; i++){
        if(ar[i] < maxH.getMax())
            maxH.insert(ar[i])
        else
            minH.insert(ar[i])
        if(maxH.size() < minH.size()){
            // Transfer min from minH
            // to maxH
            ele = minH.getMin();
            minH.deleteMin();
            maxH.insert(ele);
        }
        else if(maxH.size() - minH.size() > 1){
            // Transfer max from maxH
            // to minH
            ele = maxH.getMax()
            maxH.deleteMax()
            minH.insert(ele)
        }
        int s = i+1 // Size of both the Heaps.
        if(s % 2 == 0){
            ans[i] = (maxH.getMax() + minH.getMin()) / 2
        }
        else{
            ans[i] = maxH.getMax()
        }
    }
    return ans;
}
```

Step 1 :- Insert()

Step 2 Balancing the Heaps.

Step 3 :- Median

TC: O(N*log N)

SC: O(N)