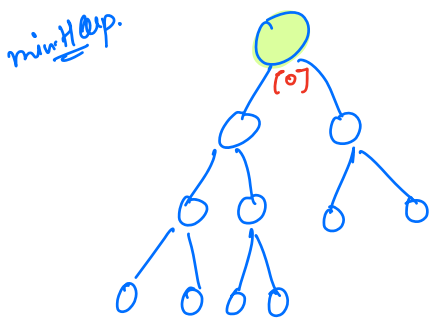
→ Heap Sort

→ K^{th} largest element

→ Sort nearly sorted array

→ median of stream of integers.

Sort an array.



A1.→ Build a min-Heap

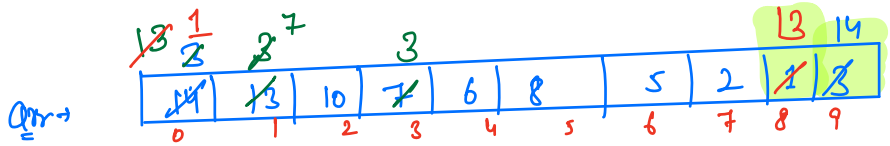
⇓

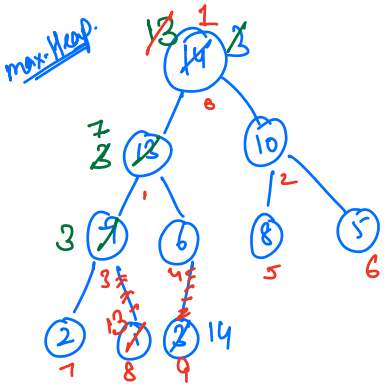
extractMin() → in O(1) array

T.C → $N + N * \log N = O(N \log_2 N)$

S.C → $O(N)$

Q: Can we optimize the space.?





Heap Sort

① Build max-Heap → O(N)

②
```
j = N-1
while ( j > 0 ) {
    swap ( 0, j )
    j--;
    heapify ( arr[], 0, j );
}
```

T.C → N + N log N → O(N log N)
S.C → O(1)

Arrays.sort(arr[]);
↓
Tim sort (Quick sort + Insertion sort)
↓
T.C → Nlogn to N^2.

* Heap sort - not stable.
  ↳ In-place sorting.

Q) Given arr[N]. Find K^{th} largest element.

arr[] → | 8 | 5 | 1 | 2 | 4 | 9 | 7 |
0 1 2 3 4 5 6

[K=3]

① Sort the array & return arr[N-K]

| 1 | 2 | 4 | 5 | 7 | 8 | 9 |
0 1 2 3 4 5 6

N=7, K=3.

T.C → O(N log N)

② Binary Search [K^{th} smallest element] → {Revise & #todo}

③ using max-heap.

Build a max-heap → Extract max K-1 times.

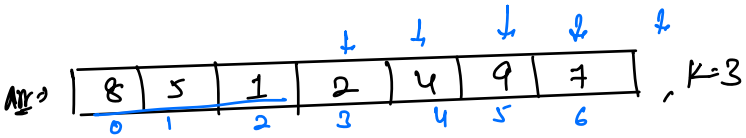
{ T.C → O(N + K·log N)
S.C → O(1) }

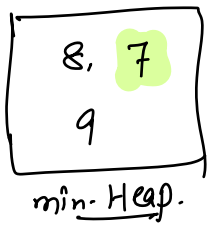
④ using min-heap.

b1	b2	b3	b4	b5	b6	b7
8	10	6	2	15	13	7

Select 4 batsman:

8, 10, 13, 15 ⇒ min Heap of size K.





getMin() to get K^{th}-largest

① Build min-Heap with first K elements. — O(K)

② iterate on the remaining elements, (N-K)

for every element check

```
if (curr ele > min of all elements in heap.)
{
    extractMin()
    insert (curr element)
}
```

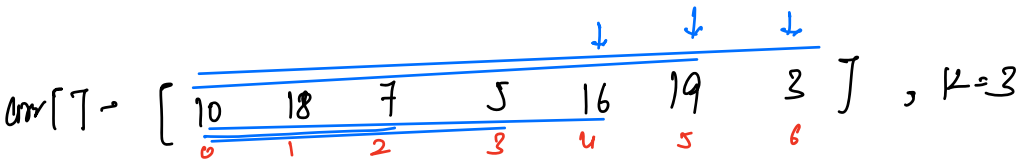
③ ans → getMin()

T.C → O(K + (N-K) log K)
S.C → O(K)

Q: for, K^{th} smallest element -

Build a max-heap of size - K.

Q) k^{th} largest element for all the windows of $(0, i)$ $(i = k-1)$



ans: [7, 7, 10, 16, 16]

ans: [7, 7, 10, 16, 16]

Q: Given a nearly sorted array. You need to sort the array.

⇓

Every element is shifted away from its correct position by atmost k-steps.

arr → [13, 22, 21, 45, 11, 20, 48, 30, 50] (k = 4)

idea-1 → Sort the array. T.C → O(N log N)

idea-2 → Minimum element can lie from 0 to k^{th} index.

min-Heap of size k+1

[48, 50, 45, 60]

[11, 13, 20, 22, 21, 45, 48, 50, 60]

① Build min-Heap with first (k+1) elements.

②
```
for( i = k+1 ; i < N ; i++){
    extractMin() → put it ans[] array
    insert( arr[i] )
}
while( minHeap is not Empty){
    extractMin() → put it ans[] array
}
return ans[];
```

T.C → O(k + (n-k) log k)
S.C → O(k)

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

↓ middle element in sorted array

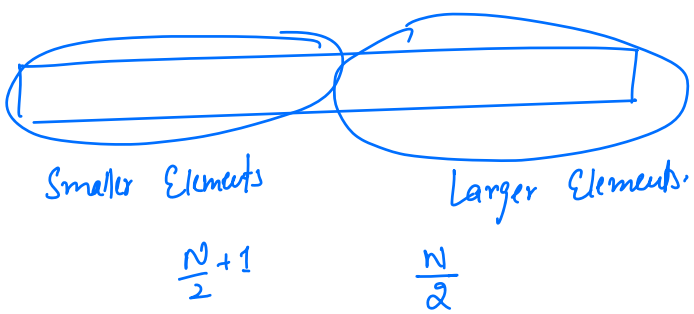
	median
6. →	6
6, 3 →	4.5
6, 3, 8 →	6
6, 3, 8, 11 →	7
6, 3, 8, 11, 20 →	8

idea-1. → for every incoming value, include the value & sort the array. Find the middle element/ avg of 2 middle elements.

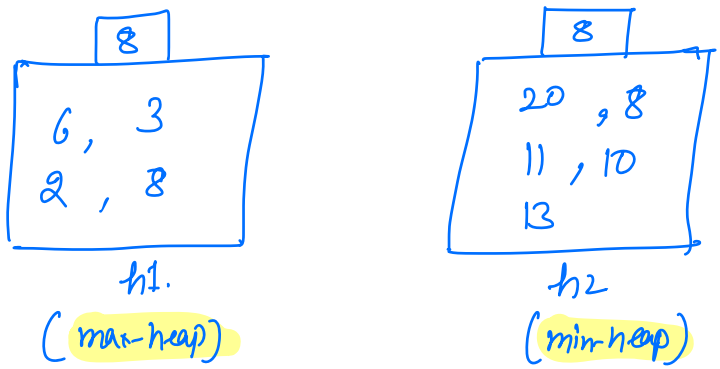
T.C → $O(N^2 \log N)$

idea-2. Every time, find the correct position of upcoming element. ie. Apply Insertion sort.

T.C → $O(N^2)$



arr → [6, 3, 8, 11, 20, 2, 10, 8, 13, 50, 4, - -



[6, 4.5, 6, 7, 8, 7, 8, 8, 8. - -] $|h1.size() - h2.size()| \le 1$

```
h1, h2, h1.insert(arr[0]);
[max-heap] [min-heap]
for( i = 1 ; i < N ; i++){
        if( arr[i] > h1.peek() ){
            h2.insert(arr[i]);
        }else{
            h1.insert(arr[i]);
        }
    diff = | h1.size() - h2.size() |
    if (diff > 1)    balance(h1, h2)
    if( h1.size() > h2.size() ) { print(h1.peek())}
    else if (h2.size() > h1.size() ) { print (h2.peek())}
     else    print( (h1.peek() + h2.peek())/2.0 );
}
```

{ T.C → O(N log N)
  S.C → O(N) }