HEAPS 2

"We achieve more when we chase the dream instead of the competition."

Simon Sinek

Good Evening

Today's content

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

Sort the Array

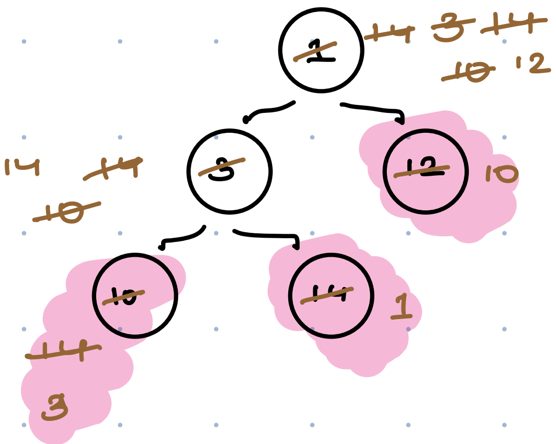
A[] = { 1 3 12 10 14 }

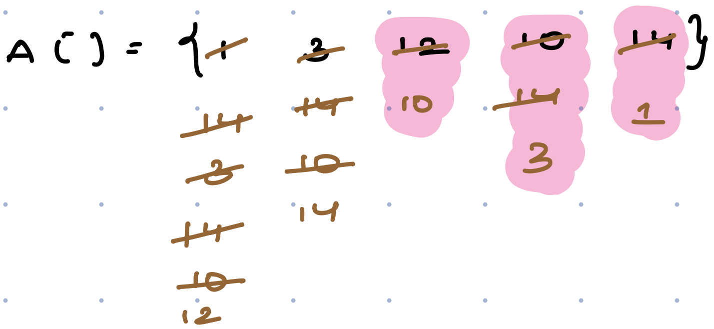
01. Arrays.sort(A) → { 1 3 10 12 14 }

TC: O(nlogn)

02. Build a min heap

↳ Extract the minimum element & store it in ans array





Ans = 1 3 10 12 14

TC: O(n + nlogn)

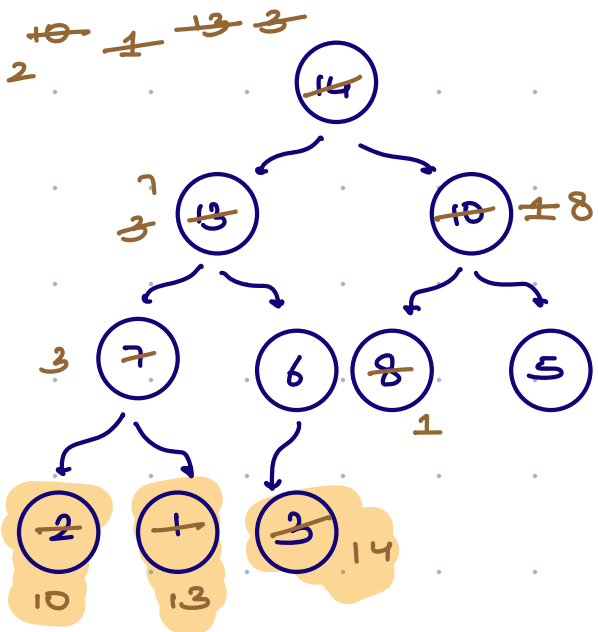
SC: O(n)

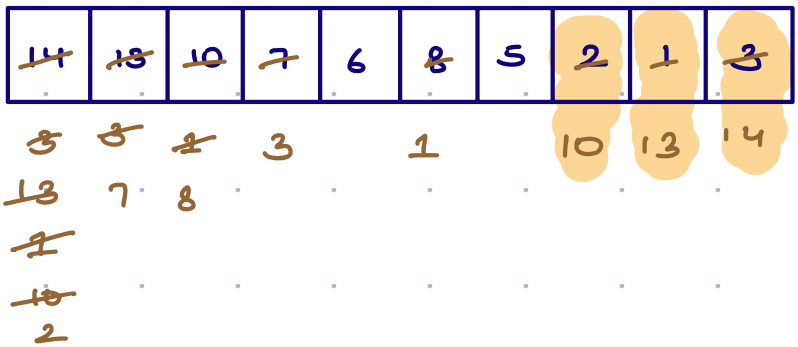
→ Optimise the SC: O(1)

* Use a max heap

→ Build a max heap

→ Store largest element in last & virtually remove it for next iteration.





Tc: $O(n + n\log n) = O(n\log n)$

Sc: $O(1)$

```
void heapsort(AL<I> A){
    maxheap(A);
    j = n-1;
    while(j>0){
        swap(A,0,j)
        j=j-1;
        downheapify(A,0,j);
    }
}
```

```
void maxheap(AL<I> heap){
    for(i = n/2 - 1 ; i >= 0 ; i--){
        downheapify(heap, i, n)
    }
}

void downheapify(heap, i, n)
{
    while(2i+1 < n){
        x = max(heap.get(i), heap.get(2i+1); heap.get(2i+2))
        if(x == heap.get(i)) return;
        else if(x == heap.get(2i+1)){
            swap(heap, i, 2i+1)
            i = 2i+1
        }
        else{
            swap(heap, i, 2i+2);
            i = 2i+2;
        }
    }
}
```

Is heapsort an inplace sorting Algo? → Yes

Is heapsort a stable sorting Algo? → No

Q Given A[n]. Find k^{th} largest element.

A[] → { 8 5 1 2 4 9 7 }

k = 2 → 8

k = 4 → 5

A[] → { 1 2 3 4 5 }

k = 5 → 1

01. Sort the array & A[n-k]

TC : O(nlogn)

02 Use Binary Search

TC : O(nlog(max-min))

03 Use max heap

01. Build a max heap → O(n)

02. Extract the maximum k times → O(klogn)

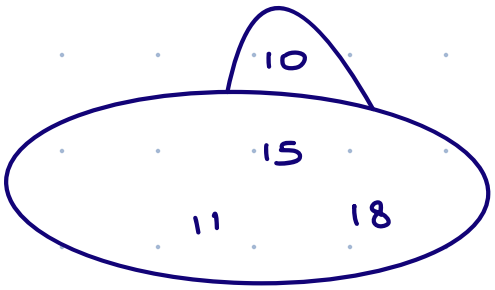
TC : O(n + klogn)

SC : O(n)

04 Use min heap

B_1	B_2	B_3	B_4	B_5	B_6	B_7
8	10	15	7	11	3	18

} Select 4 best batsmen



min heap

4th best batsman = 10

pq.peek();

* Build a minheap of size K
* Iterate on rest of elements (n-K) & remove the smallest element from peek

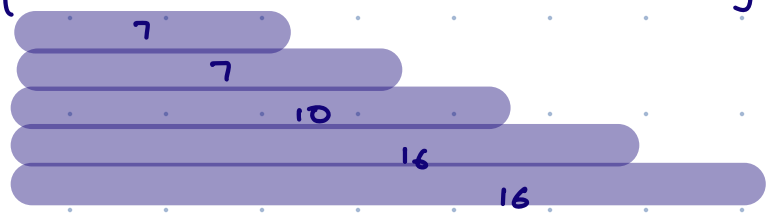
```
PriorityQueue<I> pq = new PriorityQueue();

for (i=0; i<K; i++){
    pq.add(A[i]);
}

for (i=K; i<n; i++){
    int curr = A[i]
    if (curr > pq.peek()){
        pq.remove();
        pq.add(curr);
    }
}
return pq.peek();
```

TC : O(K + (n-K) log K)

SC : O(K)

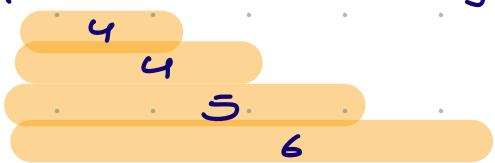
Q Find k^{th} largest element for all windows from 0 to i where $i \geq k-1$

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



Ans[] = { 7 7 10 16 16 }

A[] = { 5 4 1 6 7 } k = 2



Ans[] = { 4 4 5 6 }

```
PriorityQueue<I> pq = new PriorityQueue();
for (i=0; i<k; i++){
    pq.add(A[i]);
}
ans.add(pq.peek());
```

```
for (i=k; i<n; i++){
    int curr = A[i]
    if (curr > pq.peek()){
        pq.remove();
        pq.add(curr);
    }
    ans.add(pq.peek());
}
return ans;
```

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

SC: O(k)

10:30 pm + 5 more mins

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

mid of sorted array

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

A[ ] = { 1 2 4 3 }

↓ sort

A[ ] = { 1 2 3 4 }

median = $\frac{2+3}{2}$ = 2.5

Idea 1 → For every coming element, insert it in array & then sort it.

Return the middle element / Average of two middle elements

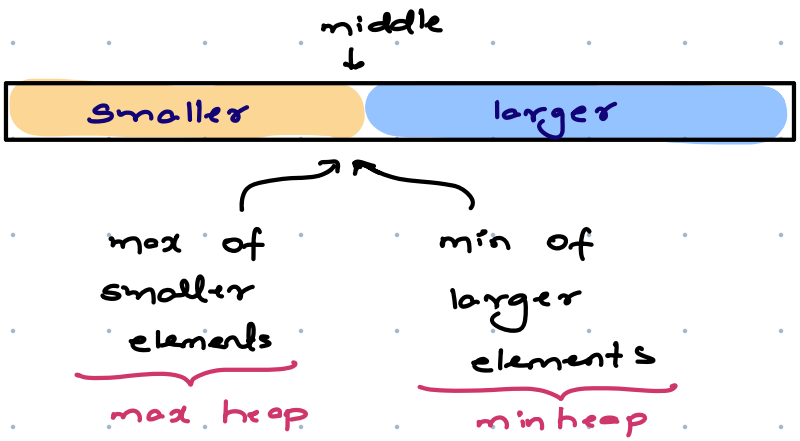
TC : O(n * nlogn)

Idea 2 Insertion sort

↳ place upcoming element at its correct position

TC : O(n^2)

Idea 3



Odd elements

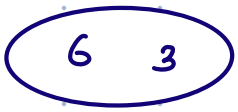
A[] = {6}

max heap

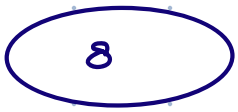
min heap

Ans = 6

A[] = {6 8 3}



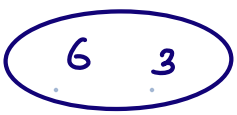
max heap



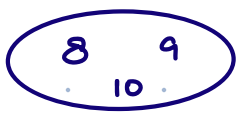
min heap

Ans = 6

A[] = {6 8 3 9 10}



max heap



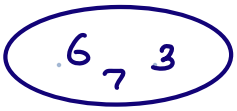
min heap

Ans = 8

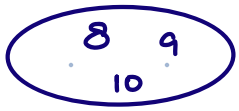
when count is odd

ans = peek element of heap having greater size.

A[ ] = { 6 8 3 9 10 7 }



max heap

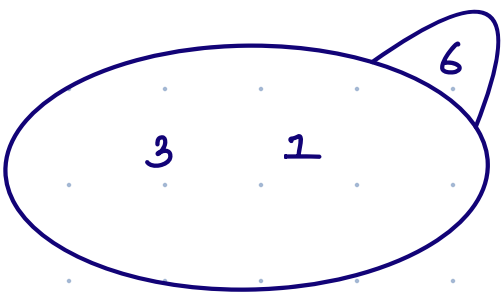


min heap

* even no. of ele

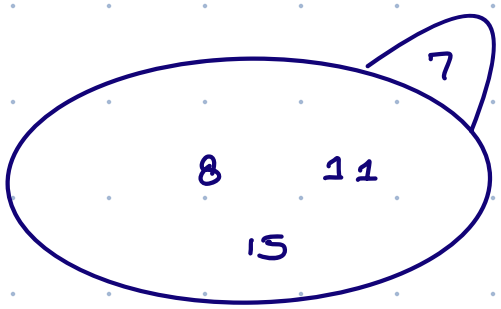
$$ans = \frac{max.peek() + min.peek()}{2}$$

A[ ] = { 6 3 8 7 11 1 15 }



maxi (max heap)

smaller



mini (min heap)

larger ele

Ans = {6 4.5 6 6.5 7 6.5 7 }

```
PriorityQueue<I> mini = new PriorityQueue();

PriorityQueue<I> maxi = new PriorityQueue(Collections.reverseOrder());

maxi.add(A[0])

print(maxi.peek());
```

```
for (i=1; i<n; i++){
    if (A[i] > maxi.peek()){
        mini.add(A[i]);
    }
    else{
        maxi.add(A[i]);
    }
    int diff = Math.abs(mini.size() - maxi.size());
    if (diff > 1) { balance(mini, maxi); }
    if (maxi.size() > mini.size()) print(maxi.peek());
    else if (mini.size() > maxi.size()) print(mini.peek());
    else {
        print( (maxi.peek + mini.peek) / 2.0 )
    }
}

void balance (mini, maxi){
    if (maxi.size() > mini.size()){
        mini.add(maxi.remove());
    }
    else
        maxi.add(mini.remove());
}
```

Tc : O(nlogn)

Sc : O(n)