Heaps

→ Max Heap.

① Find kth Smallest ele in A given Array.

arr = {10, ②, 3, 8, -4, -2, 6}   k = 3,

↓ 3rd   → Ans.   ↓ 1st   ↓ 2nd

M-1 → Brute Force.

→ Selection Sort (k passes) then arr[k-1].

→ TC :- O(kn).

M-2 → Builtin Sort → Quick Sort / Merge Sort.

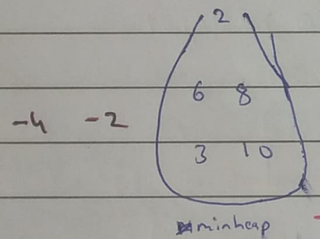
→ T.C = O(n log n).

→ S.C = O(log n) / O(n).

arr[k-1]

M-3 → Using heap / pq → But Which heap?

If we use min heap, then.



Add → nlogn.

remove → k logn.

T.C = O(n logn)

S.C = O(n).

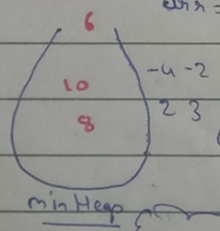
Follow Up. → minHeap.

② Find kth Largest ele. in A given Array

LeetCode Q.No. (215).

arr = {10, 2, 3, 8, -4, -2, ⑥}   k=3.

1st   2nd   3rd.   → Ans.



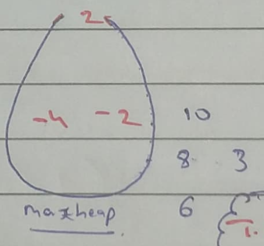
T.C → ~~n(log k)~~ O(n log k)

Auxiliary Space → O(k)

```
for(i=0 to i<n){
    pq.add(arr[i]);
    if(pq.size()>k){
        pq.remove();
    }
}
return pq.peek();
```

If we use max heap, then

=> Restrict the size of Maxheap to 'k'.



Add → n log k

remove → (n-k) log k

total → (2n-k) log k.

T.C => O(n log k)

=> Code :-

```
for (int i=0; i<n; i++){
    maxPQ.add(arr[i]);
    if(maxPQ.size()>k){
        maxPQ.remove();
    }
}
return maxPQ.peek();
```

→ Good Interview Question.

③ Sork A 'k' Sorted Array (Sort a nearly Sorted Array)

⇒ k = 3

$$arr = \{\overset{0}{⑥}, \overset{1}{⑤}, \overset{2}{3}, \overset{3}{2}, \overset{4}{8}, \overset{5}{10}, \overset{6}{9}\}$$

→ Every ele is <= k times shifted from its Actual position

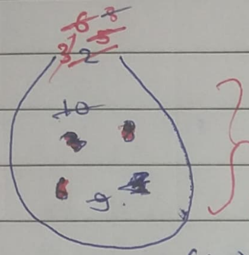
Sort → $= \{\overset{0}{2}, \overset{1}{3}, \overset{2}{⑤}, \overset{3}{⑥}, \overset{4}{8}, \overset{5}{9}, \overset{6}{10}\}$

6 → 0 - 3 ⇒ 3<= k.   3 → 2 - 1 ⇒ 1 <= k.

5 → 2 - 1 ⇒ 1 <= k.

⇒ we will use MinHeap i

arr = {6, 5, 3, 2, 8, 10, 9}.



minheap(k).

k remaining ele will Come at end of Array.

ele At 0th ~~e~~idx ~~[illegible]~~ max^m Can shifted k index. So min^m ele from this window. ~~will be at~~ its Actual position is 0th idx of Array.

Ans = {2, 3, 5, 6, 8, 9, 10}.

⇒ Code :- int idx = 0.

```
for(i=0 to i<n){
    pq.add(arr[i]);
    if(pq.size() > k){
        arr[i] = arr[idx];
        arr[idx] = pq.remove();
        idx++;
    }
}.

while(pq.size() > 0){
    arr[idx] = pq.remove();
    idx++;
}
```

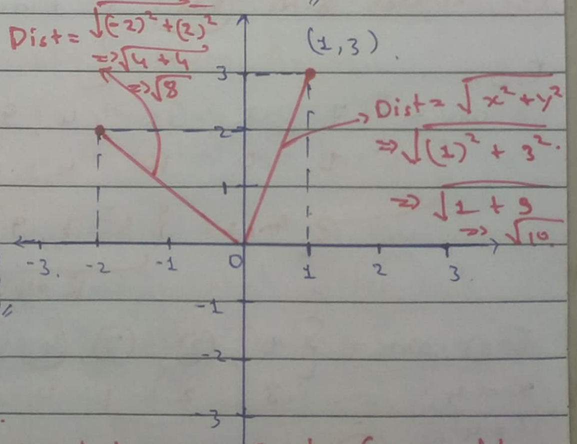
④ LeetCode Q.No. 973 { K Closest Points to Origin. }

Ex①:-

points = {{1, 3}, {-2, 2}}

k = 1.

→ Ans ⇒ ~~{[illegible]}~~ {-2, 2}



→ k Smallest Distance → Maxheap.

Ex②

arr :- {{3, 3}, {5, -1}, {2, 4}, {1, 0}, {3, 2}}.

$\boxed{k = 2}$ 18   26   20   ①   ⑬

→ Ans ⇒ {{1, 0}, {3, 2}}.

map < Distance, Coordinates >
int   Arr. ~~[illegible]~~ → Triplet.

Need to use Custom Comparable
→ To Sort on the basis of Distn.

```
=> Code :- int[][] kClosest(int[][] points, int k){

    ~~minHeap~~
    P ~~pq = new PQ<>(~~
                  → Object.
    PQ<Triplet> pq = new PQ<>(Collections.reverseOrder());
    for(i=0 to i<n){
        int x = points[i][0];
        int y = points[i][1];
        ind ~~x~~d = x*x + y*y;
        pq.add(new Triplet(d, x, y));
        if(pq.size() >k)
            pq.remove();
    }.
    Store Max Heap k ele in 2-D Array And Return.
}.
```

Imp

```
Class Triplet implements Comparable<Triplet>{
    int d;
    int x;
    int y;
    Triplet(int d, int x, int y){
        this.d = d;
        this.y = y;
        this.x = x;
    }.

    public int CompareTo(Triplet t){
        return this.d - t.d;
    }
}
```

(5) LeetCode Q.No. (658) { Find k Closest Elements to x. } → smallest → MaxHeap.

Ex①.

arr = { 7, 8, 18, 4, 9, 6 }.   k = 3   x = 8

  1   0   10   4   1   2

→ Ans :- {8, 7, 9} (k ele's.)

Closeness → a, b, x.

→ a is Closer to x if

$|x-a| < |x-b|$.

& if $|x-a| == |x-b|$ then if $a<b$ then a is closer.

Ex② arr = { 1, ②, ③, ④, ⑤, 6, 7 }

x = 4,

k = 4,

  3   2   1   0   1   2   3.

② < 6

→ MaxHeap<Pair> → ele, |x-ele| → Comparable

```
=> Code :- find Closest Ele (int[] arr, int k, int x){
    PQ<Pair> pq = new PQ<>(Collections.reverseOrder());

    for(int i = 0 to i<n){
        pq.add(new Pair(arr[i], Math.abs(x - arr[i])));
        if(pq.size() > k)
            pq.remove();
    }
    Store remaining k ele in Heap to Array List & return.
}

Class Pair implements Comparable<Pair>{
    int ele;
    ind dist;
    Pair(int ele, int dist){
        this.ele = ele;
        this.dist = dist;
    }

    Public int compareTo(Pair p){
        if(this.dist == p.dist){
            return this.ele - p.ele;
        }

        else return this.dist - p.dist;
    }
}
```

T.C => O(n log k)

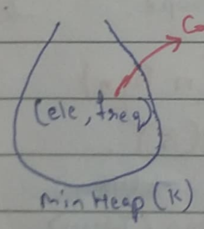
A.S => O(k)

(6) LeetCode Q.No. (347) { Top k Frequent elements. } → K Largest freq Element → minHeap.

Ex:-
arr = {1, 2, 1, 3, 1, 3, 3, 3, 2, 3, 4, 5,}.
k = 3.
→ Return those 'k' ele which occurs the most in Given Array.

Ex Space = O(n)
T.C = O(n log k)

In the Question :- It is Given, each distint ele has a unique freq.



Comparable based of freq

(1, 3).
(2, 2)
(3, 5)
(4, 1)
(5, 1)

map

Steps :- (1) Find freq of All ele using Map.

(2) Create minHeap<Pair> which stores pair of (ele, freq). use Custome Comparable ~~to~~ On freq. So, that it will Sort on basis of freq.

(3) Add All Pairs of map to Heap if size of Heap > k then remove.

(4) → You will Get top K ele with Hight freq.

H.W :- LC 1636.

7 LeetCode Q.No. 1046 { Last Stone Weight. }

At Every ~~f~~ Step we are taking 2 Largest Stones. → Max Heap.

=> Brute Force :-
→ Use ArrayList & Sorting.

Time → $n\log n + (n-1)\log(n-1) + (n-2)\log(n-2) \ldots$

$\sum_{r=n}^{1} r \log \approx \boxed{n^2 \log n}$

r → n to 1.

=> Using MaxHeap.
→ Here, we have at max 'n' ele in heap. So insertion, removal of each ele will take 'logn'
→ T.C = O(n logn).

Ex :-

arr = {2, 7, 4, 1, 8, 1}
↓ 8 - 7 = 1
{2, 4, 1, 1, 1}.
4 - 2 = 2 ↓
{2, 1, 1, 1}.
2 - 1 = 1 ↓
{1, 1, 1}
↓ 1 - 1 = 0
{1}

Steps :- (1) Add all the ele ~~in~~ of Array to maxHeap. [max^n ele At top].

(2) Run A loop untill heap size become equal to 1.

(3) Then every time remove two top(Greater) ele. ~~x~~ ~~y~~

(4) If they are equal then they will destroy themselve.. No need to add

(5) if they are not equal the add there difference (y - x) in Heap.

(6) If heap size is 0 it mean All the stones destroyed return 0 else return heap peek ele.

8 Minimum Cost to Connect All Ropes.

→ I have 'n' Ropes of ~~Size~~ Some lengths
→ I can connect 2 Ropes At a time & the Cost of joining is the sum of length of ropes.

arr = {2, 7, 4, 1, 8}.

2 ~~8~~ 7 4 1 8
2 15 4 1
2 19 1
21 1
Cost => 15 + 19 + 21 + 22.
=> 77
max.

2 7 4 1 8
3 7 4 8
7 7 8
14 8
Cost => 3 + 7 + 14 + 22.
=> 46
min.

[Logic is to ~~[illegible]~~ every time join 2 smallest length rope.]

T.C = O(n log n)

S.C = O(n).

Create ~~[illegible]~~ minHeap <Integer> ~~Add All ele to Heap.~~ every time remove two smallest ropes x & y. cost ~~+=~~ x+y. & Add (x+y) to Heap.

(9) LeetCode Q.No. (295) { Find Median from DATA Stream }

⇒ 8, 6, 1, 3, 13, 18, -6.

↳ -6, 1, 3, (6), 8, 13, 18.
↳ Median

⇒ 10, 80, 100, 40

↳ 10, (40), (80), 100
Avg. (60) → median.

⇒ DATA Stream } → Every Time One ele ~~ee~~ will be added. you need to return median.

Stream		Median
8	→	8
8, 6	→	7
8, 6, 1	→	6
8, 6, 13	→	4.5
8, 6, 13, 13	→	6.

M - 1

→ Use ArrayList every time Add ele from back

→ Then sort using Buildin Sort

→ Then find Median.

T.C ⇒ $1 \log 1 + 2 \log 2 + 3 \log 3 + \ldots n \log n$

$$\Rightarrow \sum_{r=1}^{n} r \log r.$$

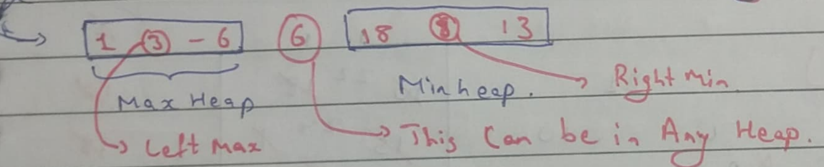
$$\Rightarrow n^2 < \sum_{r=1}^{n} r \log r < n^2 \log n.$$

M - 2 → Improved Approch :-

↳ Do Not use Buildin Sort.
Use Insertion Sort Algo.

⇒ Time → $1 + 2 + 3 + \ldots n = O(n^2)$.

{8 6 1 3 18 13 -6.} **M - 3** → Best Approch → Using Heaps. T.C = $O(n \log n)$



⇒ Code :- → $O(1)$ } → Total ⇒ $O(n)$.

```
p double findMedian() {
      if(maxHeap.size() == minHeap.size()){
            return (maxHeap.peek() + minHeap.peek()) / 2.0;
      }
      else if (maxHeap.size() > minHeap.size()){
            return maxHeap.peek();
      else
            return minHeap.peek();
}
```

→ O(logn) for One Call.
So for 'n' Calls => O(n log n).

```
p void addNum(int num){
    if(maxHeap.size()==0)   maxHeap.add(num);
    else{
        if(num < maxHeap.peek())   max Heap.add(num);
        else   minHeap.add(num);
    }.

    // Balance the Heaps:-
    if(maxHeap.size() == minHeap.size()+2){.
        int top = maxHeap.remove();
        minHeap.add(top);
    }

    ~~else~~ if(minHeap.size() == maxHeap.size()+2){
        int top = minHeap.remove();
        maxHeap.add(top);
    }
}
```

(10) LeetCode Q.No. (632) { Smallest Range Covering Elements from k Lists. }

Range
↓
[a,b] → b-a Should be minimum.

heap<Triplet> → ele, row, col.

Ex:-

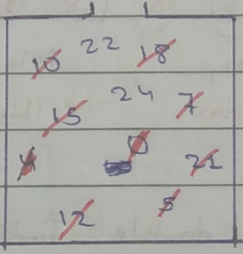
[4, 10, 15, 24, 26]
[0, 7, 12, 21]
[5, 18, 22, 30]

ele max

minRange= [0, IMax]

21
4 10
0
7 5
18 12 15

✓{0,5} ✓{4,7} {5,10}✗ {7,18}✗
{10,18}✗ {12,18}✗ {15,21}✗
{18,24}✗ {21,24}✗



MinHeap.

Max.
~~5~~
~~4~~
~~10~~
~~18~~
~~21~~
24

If there are total 'n' ele in nums,

Extra Space : O(k)
Time Complexity : O(n log k).

```
=> Code :- p  int[]  smallestRange (List<List<Integer>> nums){
    int[]  ans  =  {0 ,  Integer.MAX_VALUE};
    PriorityQueue<Triplet>  pq  =  new  PQ<>();   // MinHeap.
    int  k  =  nums.size();

    int  max  =  Integer.MIN_VALUE;
    for (int  i = 0 ;  i<k ;  i++){
        int  ele  =  nums.get(i).get(0);
        pq.add(new  Triplet(ele, i, 0));
        max  =  Math.max(max, ele);
    }.

    while (true){
        Triplet  top  =  pq.remove();
        int  ele  =  top.ele ,  row  =  top.row ,  col = top.col;
        // Update the minimum Range
        if (max - ele  <  ans[1]  -  ans[0]){
            ans[0]  =  ele;
            ans[1]  =  max;
        }
        if (col== nums.get(row).size()-1)          break;
        int  next  =  nums.get(row).get(col+1);
        max  =  Math.max(max, next);
        pq.add(new  Triplet(next, row, col+1));
    }.

    return  ans;
}.
```