First Negative Integer 2

```
public static void firstNegetiveInteger(int[] arr, int n, int k) {
    // que contain index of -ve elements only
    Queue<Integer> que = new LinkedList<>();
    int i = 0;
    while (i < k) { // k times only (for first window)
        if ( arr[i] < 0 ) {
            que.add(i);
        }
        i++;
    }

    while ( i < n ) {
        // first negetive element of current window
        if ( que.size() > 0 ) {
            System.out.print( arr[que.peek()] + " " );
        } else {
            System.out.print( 0  + " ");
        }

        // remove all elements which are out of window
        while ( que.size() > 0 && que.peek() < (i - k + 1) ) {
            que.poll();
        }

        // keep adding -ve elements
        if ( arr[i] < 0 ) {
            que.add(i);
        }
        i++;
    }

    if ( que.size() > 0 ) {
        System.out.print( arr[que.peek()] + " " );
    } else {
        System.out.print( 0  + " ");
    }
}
```

to insert -ve elements in queue for first window

to print first -ve no.

to remove out of window elements

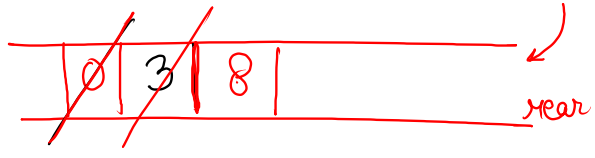
k=3

arr = [-8, 2, 3, -6, 10, 3, 1, 4, -2]
0 1 2 3 4 5 6 7 8

i

queue

front



ans = -8 -6 -6 -6 0 0 -2

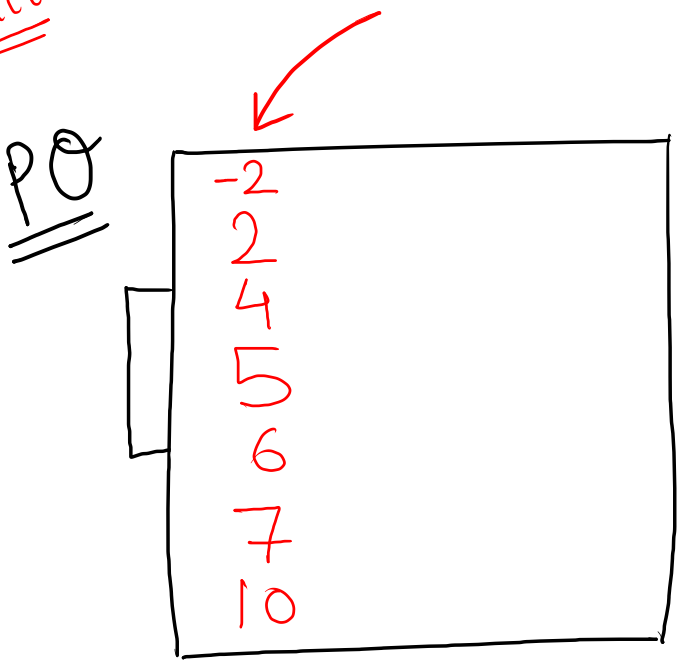
3 < 6 - 3 + 1

3 < 4

⟹ Priority Queue (Heap)

(any element that will be placed in PQ will always be in sorted order)

default



In Java

default PQ is ascending order

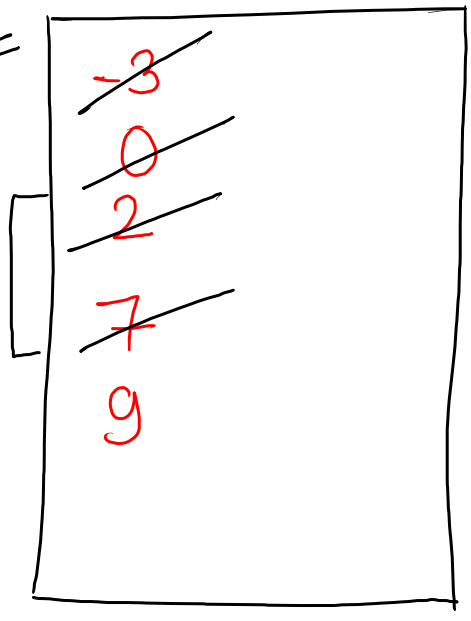
In C++

default PQ is descending order

Note:- we can only access the top element

Ex:-

PQ



arr = [2, -3, 0, 9, 7]

Note:-

In PQ, duplicay is allowed

Syntax

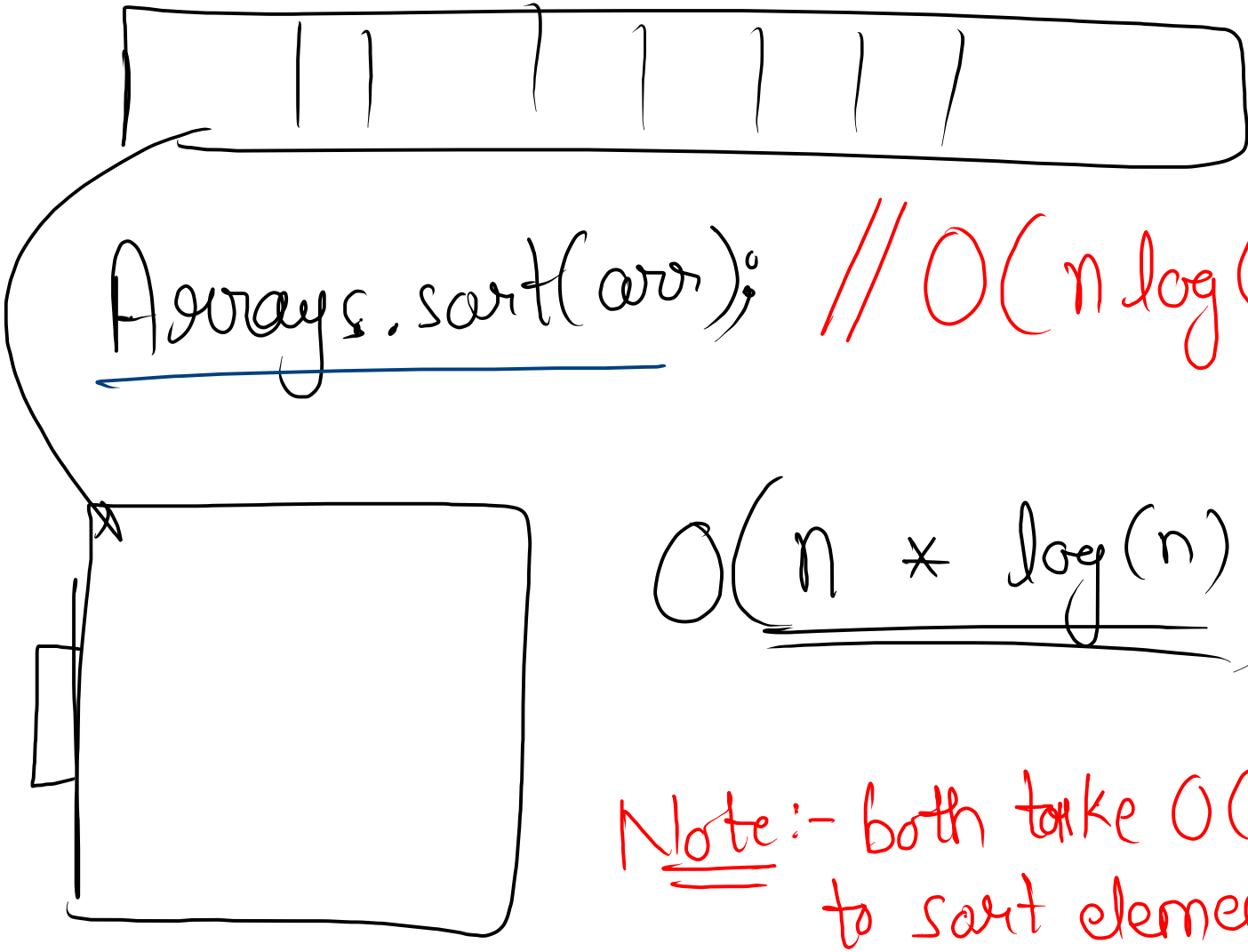
```
PriorityQueue<Integer> pq = new PriorityQueue<>();
```

Inbuilt functions

[Note:- all functions in PQ takes O(log(n))]

```
pq.add(x);      // add element in PQ
pq.remove();    } //remove & return
pq.poll();      }
pq.peek();      // access top element
```

```
pq.size() / pq.isEmpty();
```



Arrays.sort(arr); // O(n log(n))

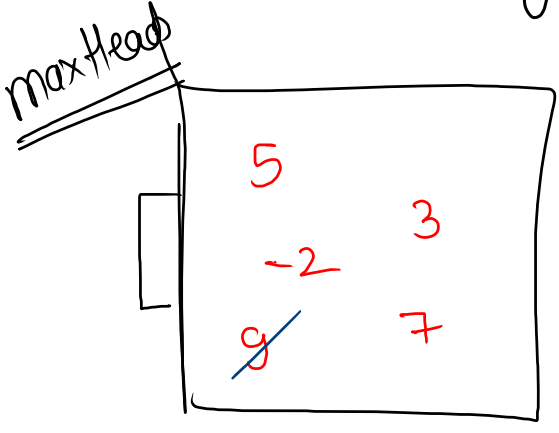
O(n * log(n))

Note :- both take O(n log(n)) to sort elements

Note:-

Priority Queue is always called

- minHeap is when PQ sorts in ascending order
- maxHeap is when PQ sorts in descending order.



peek element is 9

Lambda function in PQ

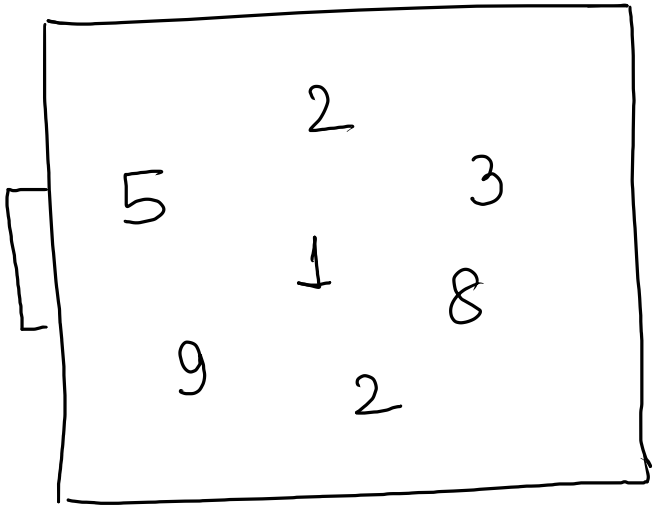
```
Arrays.sort(arr, (a,b) -> {
    return b-a;
});
```

```
PriorityQueue<Integer> pq = new PriorityQueue<>((a,b) -> {
    return b-a;
});
```

priority queue basics

arr = [5, 2, 1, 3, 8, 9, 2]

minHeap
PQ



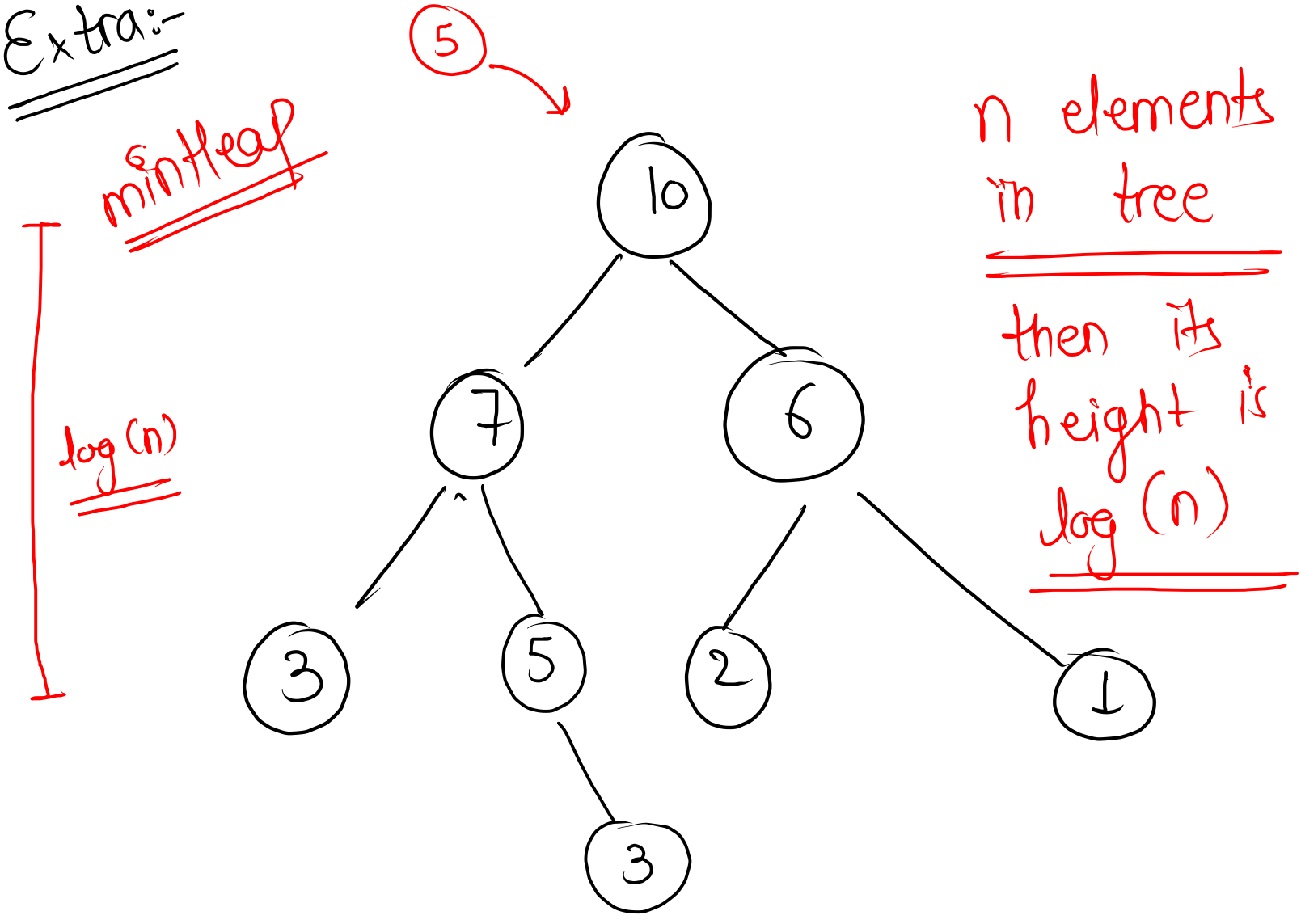
ans:- 5
2
1
1
1
1
1

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int t = scn.nextInt();
    int[] arr = new int[t];
    for (int i = 0; i < t; i++) {
        arr[i] = scn.nextInt();
    }
    PQbasic(arr, t);
}
public static void PQbasic(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i = 0; i < n; i++) {
        pq.add( arr[i] );
        System.out.println( pq.peek() );
    }
}
```

$$T.C = n\log(n) + n\log(n)$$
$$= 2n\log(n)$$

Extra:-


minHeap
log (n)
5
10
7
6
3
5
2
1
3
n elements in tree
then its height is log (n)

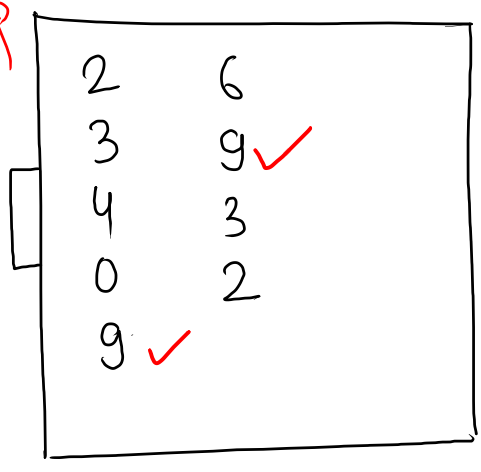
Maximum Product of Two Elements in an Array

exp :- $(arr[i] - 1) * (arr[j] - 1);$

arr =

2	3	4	0	9	6	9	3	2

maxHeap



int a = pq.poll() // 9

int b = pq.poll() // 9

Code

T.C = O(n log(n))

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    maxProd(arr, n);
}
public static void maxProd(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> {
        return b - a;
    });
    for (int i = 0; i < n; i++) {
        pq.add(arr[i]);
    }
    int a = pq.poll();
    int b = pq.poll();
    System.out.println( (a - 1) * (b - 1) );
}
```

Reduce Array Size to the half 1

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

arr = [3, 3, 3, 3, 5, 5, 5, 2, 2, 7]

Note:- remove element only until it's size is half or less than half

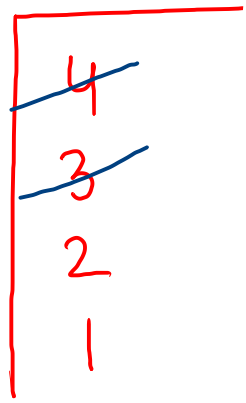
return, mini no. of operations required

map

arr ele → freq
3 → 4
5 → 3
2 → 2
7 → 1

PQ

maxHeap



Code

```java
public static int reduceArraySizeToHalf(int[] arr, int n) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < n; i++) {
        if ( map.containsKey(arr[i]) == false ) {
            map.put( arr[i], 1 );
        } else {
            int freq = map.get(arr[i]);
            map.put( arr[i], freq + 1 );
        }
    }

    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> {
        return b - a;
    });
    for (int i : map.values()) {
        pq.add(i);
    }
    int size = n;
    int count = 0;
    while ( size > n / 2 ) {
        count++;
        int rem = pq.peek();
        size = size - rem;
        pq.poll();
    }
    return count;
}
```

$$T.C = n + n\log(n) + n\log(n)$$
$$= O(n\log(n))$$

map — arr ele → freq

3	4
5	3
2	2
7	1

PQ maxHeap

4
3
2
1

size = ~~10~~ ~~8~~ 3

count = ~~0~~ ~~1~~ 2

rem = ~~4~~ 3

minimum digits

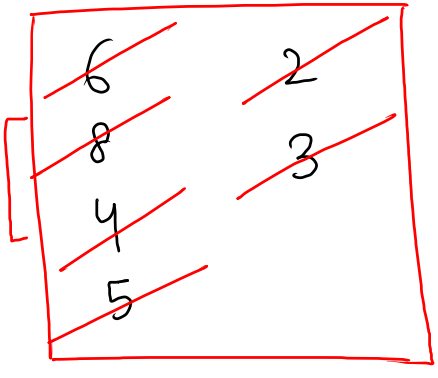
arr = [6 , 8 , 4 , 5 , 2 , 3]

num1 = 2 4 6

num2 = 3 5 8

num1 = num1 * 10 + rem

= (2 * 10) + 4



num1 = 2 4 6

num2 = 3 5 8