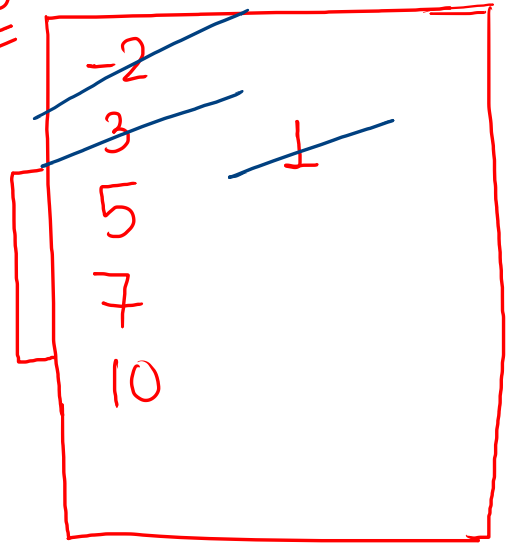
⟹ Priority Queue (any elements store in PQ are always sorted in a sequence)

Java

PQ

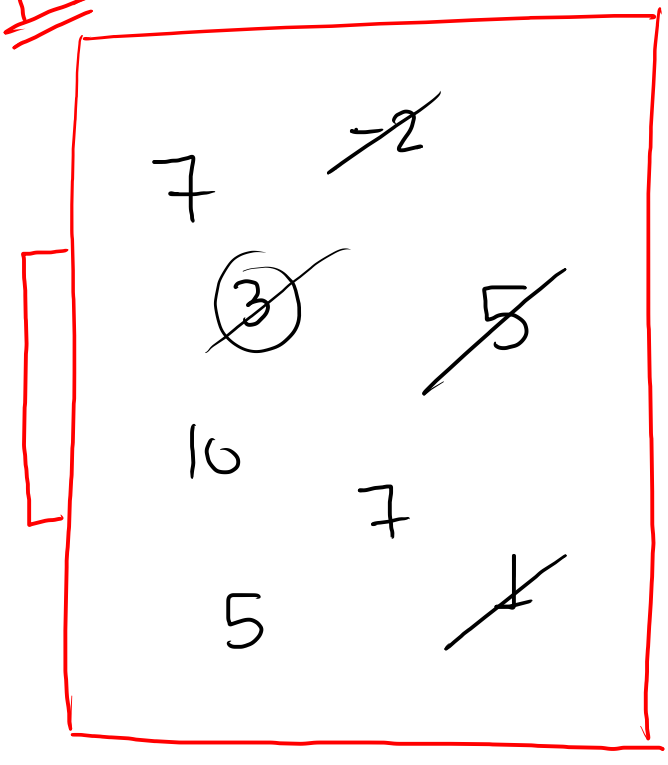



```
Arrays.sort(arr, (a, b) -> {
        return a-b; }
        return b-a;

});
```

input = 7, 5, 10, -2, 3, 1

PQ



input = 5, 7, -2, 3
10, 7, 5, 1
output = 5, -2, 1, 3

Note:- duplicacy is allowed

Syntax

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

Inbuilt f^n (all f^n's take logN time)

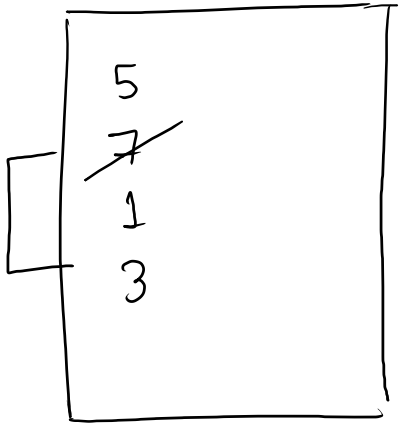
- `pq.add(x);` // add element in PQ
- `pq.remove();` } used to remove element from PQ
- `pq.poll();` } used to remove element from PQ
- `pq.peek();` // get top element without removing it
- `pq.size()` , `pq.isEmpty()`

Note:- Priority Queue is also called as

minHeap when PQ is in ascending order

maxHeap when PQ is in descending order

maxHeap (pq)



i/p = 5, 7, 1, 3

o/p = 7, 5

priority queue basics

PQ

t=7

5	-2
7	1
2	
8	
0	

default

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int t = scn.nextInt();

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

    for (int i = 0; i < t; i++) {
        int num = scn.nextInt();
        pq.add(num);
        System.out.println( pq.peek() );
    }
}
```

o/p = 5, 5, 2, 2, 0, -2, -2

$2t * \log(N)$

T.C = $N \log(N)$

Arrays.sort(arr); $\longrightarrow$ $O(n \log n)$

TC for n elements; $\longrightarrow$ $n * \log(n) + n * \log(n)$

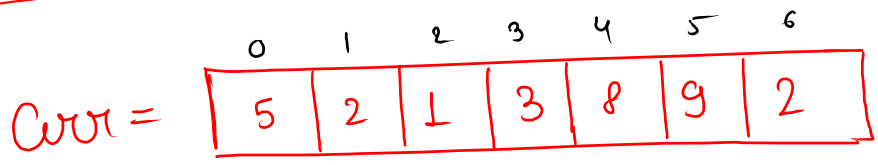
$\longrightarrow$ $2n \log(n)$

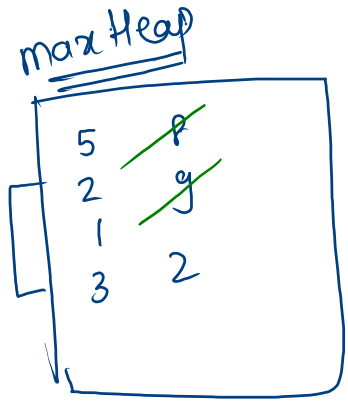
$O(n \log(n))$

where

n $\rightarrow$ no. of elements

Maximum Product of Two Elements in an Array





max value = (num1 − 1) * (num2 − 1)

num1 → largest no.

num2 → 2nd largest no.

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

max Heap

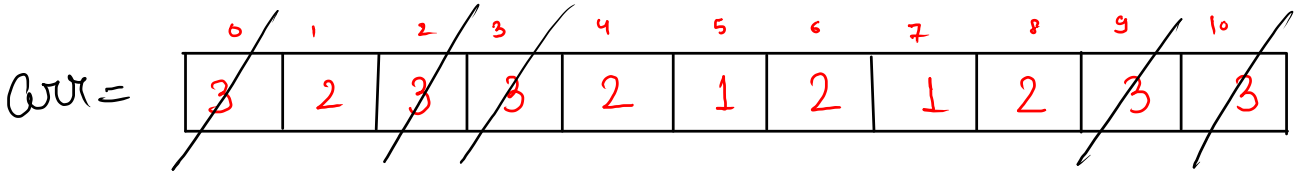
Code

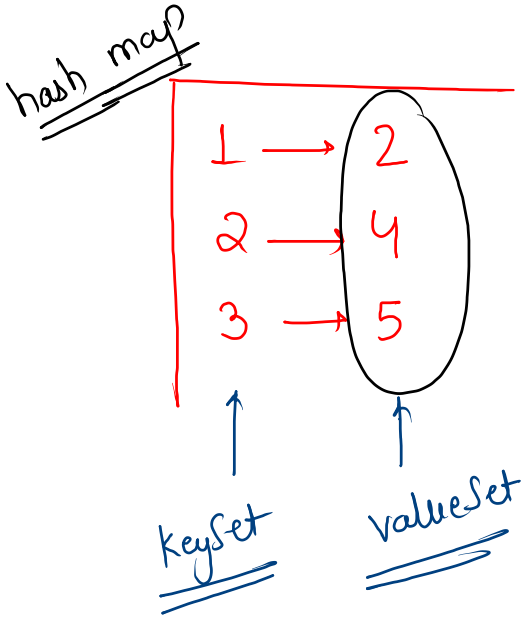
T.C = O(nlog(n))

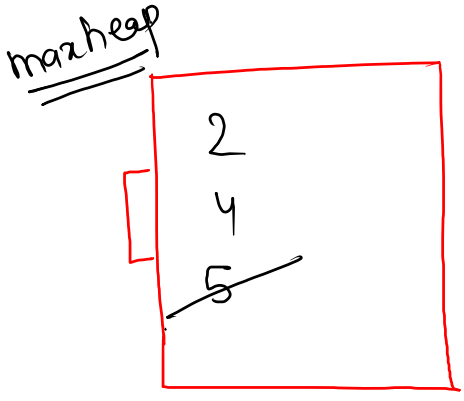
```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    System.out.println(maxProd(arr));
}
public static int maxProd(int[] arr) {
    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> {
        return b - a;
    });
    for (int i : arr) {
        pq.add(i);
    }
    int num1 = pq.poll();   // 9
    int num2 = pq.poll();   // 8
    return (num1 - 1) * (num2 - 1);
}
```

Reduce Array Size to the half 1

size = 11







Psudo code

1) create HM
2) store freq of all elements
3) create PQ
4) store all values of HM in PQ

   size of array = n;

5) loop until size become half

   5.1) n = n - pq.poll();

   count++;

Code

```java
public static int reduceSizeToHalf(int[] arr, int n) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < arr.length; 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 ) {
        size = size - pq.peek();
        pq.poll();
        count++;
    }
    return count;
}
```

N

NlogN

NlogN

$T.C = O(2N\log N + N)$

$\simeq O(N\log N)$

$S.C = O(N)$