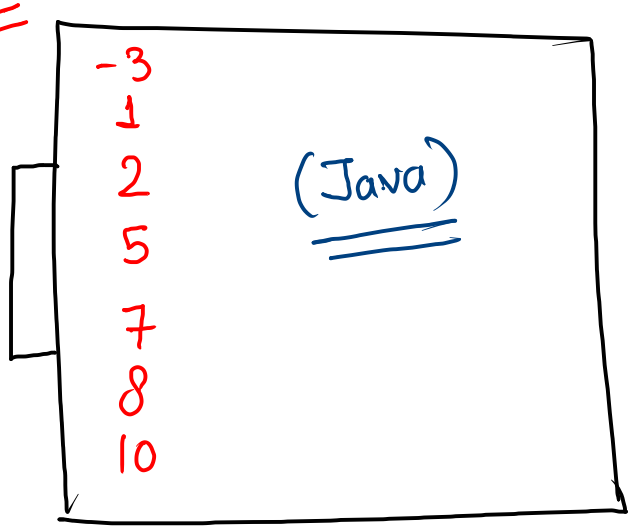
⇒ Priority Queue (any element that will be placed in PQ will always be in sorted order)

↳ Heap

Note:- ↳ also dynamic nature
↳ no indexing present

2, 7, 1, -3, 10, 5, 8

PQ



Note:- (default)

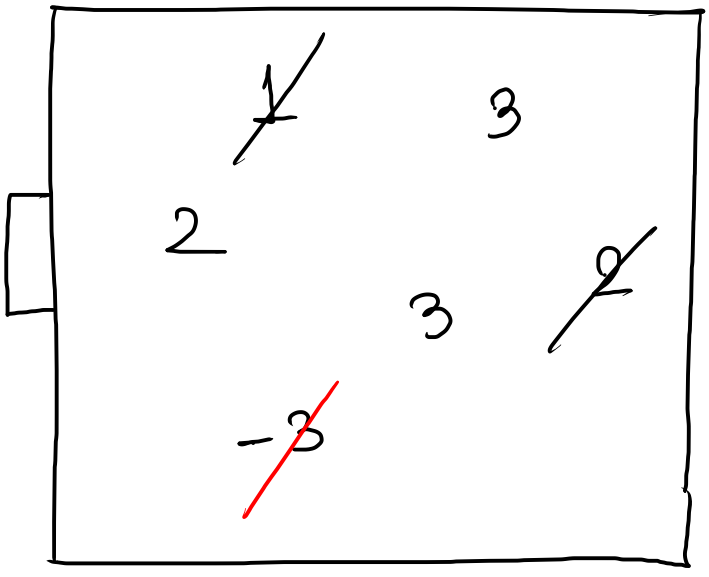
↳ by default, PQ will sort elements in increasing order (Java)

↳ by default, PQ will sort elements in decreasing order (C++)

Note:- we can only access the top element

PQ

add → 2, 3, −3, 3, 1, 2



Note:- In PQ, repetation is allowed

Remove → −3, 1, 2, 2

Syntax

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

Imp

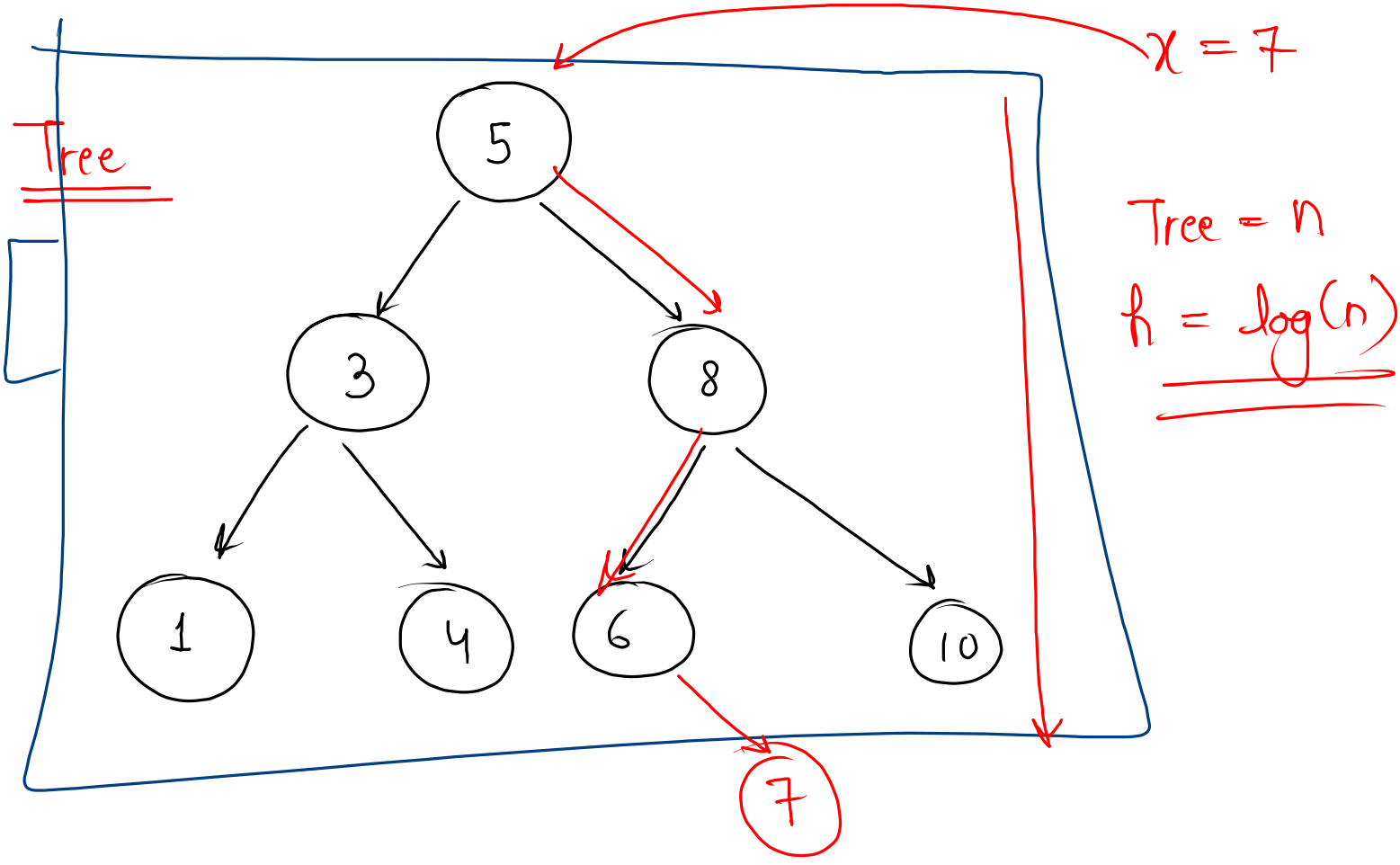
> Note:- every inbuilt fn T.C = log(n)
>
> where n is size of pq

Inbuilt functions

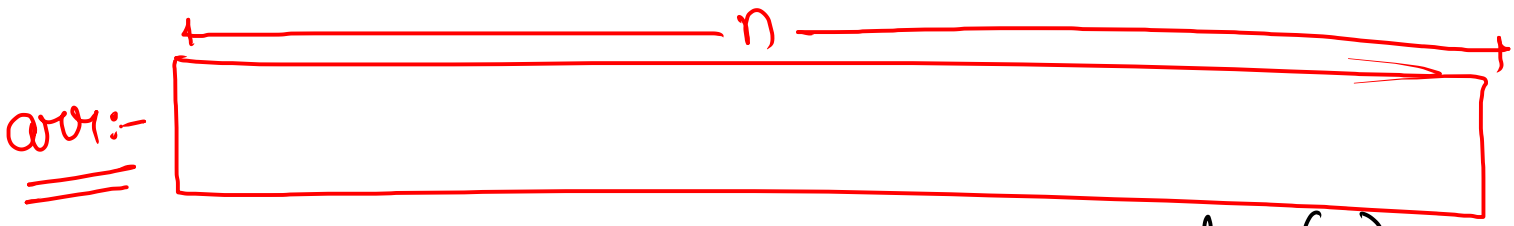
- `pq.add(x)` // add element in PQ
- `pq.remove()` } // remove and return top element of PQ
- `pq.poll()` }
- `pq.peek()` // return top element of PQ

`pq.size()` / `pq.isEmpty()` ✓ evergreen

Behind the scene of PO (Ignore)



Note:- every question that can be solved using PQ, can also be solved using sorting & vice versa



sort :- $n \log(n)$

PQ

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

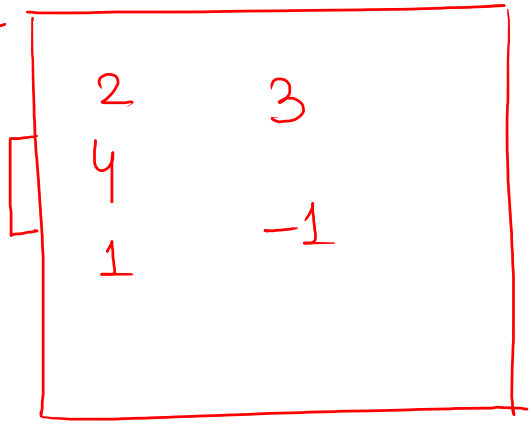
Imp :- Lambda function

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

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

priority queue basics

PQ



t = 5

- 4 → 4
- 2 → 2
- 1 → 1
- 3 → 1
- -1 → -1

code

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

$S.C = O(n)$

```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();
    }
    solve(arr, t);
}
public static void solve(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i : arr) {
        pq.add( i );
        System.out.println( pq.peek() );
    }
}
```

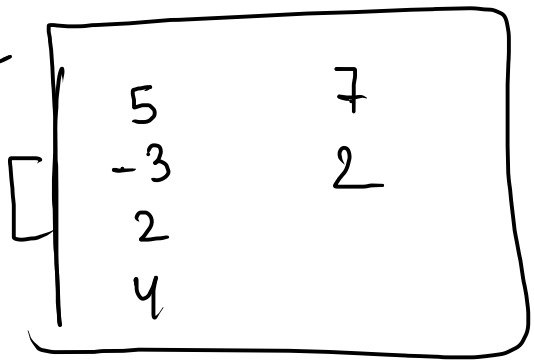
Maximum Product of Two Elements in an Array

arr = [5, -3, 2, 4, 7, -2]

val = (arr[i] - 1) * (arr[j] - 1) } max

val = (7 - 1) * (5 - 1) = 24

PQ



Note :-

PQ with ascending order nature :- min heap

PQ with descending order nature :- max heap

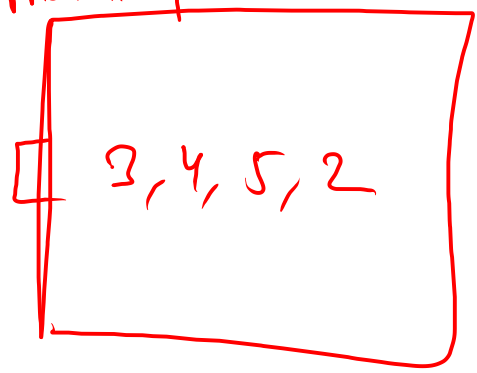
Code

```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();
    }
    System.out.println(solve(arr, t));
}
public static int solve(int[] arr, int n) {
    // max heap
    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> {
        return b - a;
    });
    for (int i : arr) {
        pq.add(i);
    }

    int num1 = pq.poll();
    int num2 = pq.poll();
    return (num1 - 1) * (num2 - 1);
}
```

3, 4, 5, 2

max heap



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

$= O((n+2) \log(n))$

$= O(n \log(n))$

Lambda fn never effects the T.C

minimum digits

arr = [5 , 2 , 3 , 0 , 9 , 7 , 3]

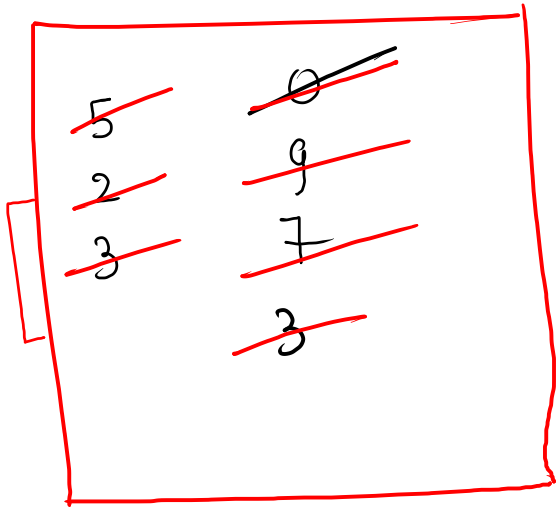
num1 = 0 3 7

num2 = 2 5 9

num1 = num1 * 10 + peek

num2 = num2 * 10 + peek

min heap

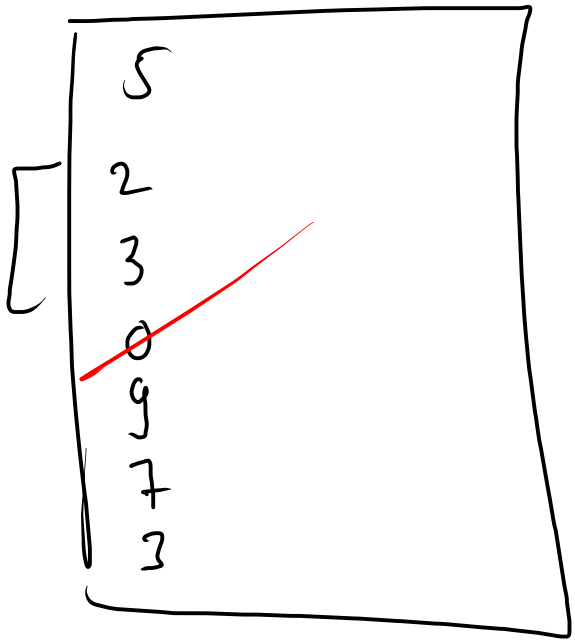


num1 = 0 3 5 9

num2 = 2 3 7

arr = [5 (0), 2 (1), 3 (2), 0 (3), 9 (4), 7 (5), 3]

Size = 7



pq.size = odd

num1 =

num2 =

pq.size = even

code

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

```java
public static long minimumSum(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i : arr) {
        pq.add(i);
    }

    long num1 = 0;
    long num2 = 0;
    while ( pq.size() > 0 ) {
        int rem = pq.poll();
        if ( pq.size() % 2 == 0 ) {
            num1 = num1 * 10 + rem;
        } else {
            num2 = num2 * 10 + rem;
        }
    }

    return num1 + num2;
}
```