⇒ Priority Queue (also called as Heap)

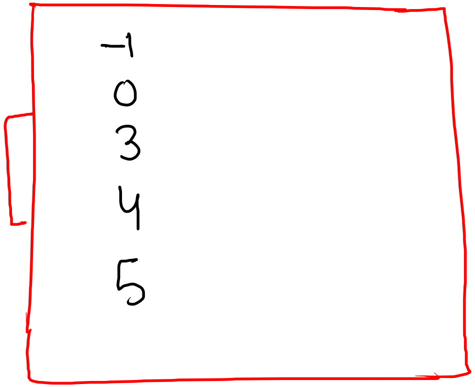
3, 5, -1, 4, 0

In Java

1) default pq is min PQ

means top element will always be smallest

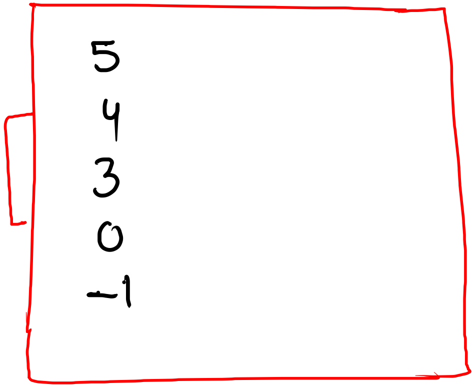
min PQ

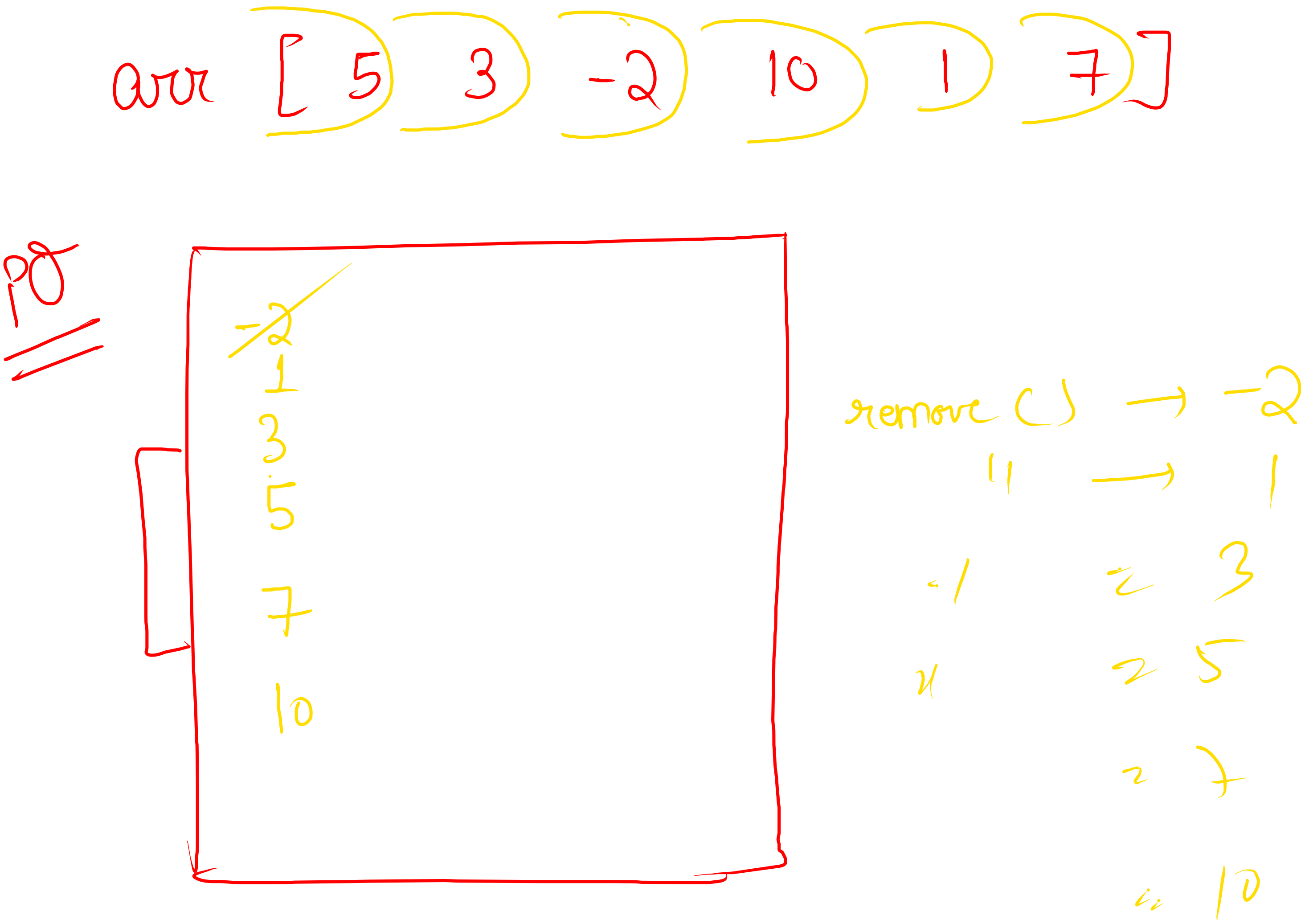


2) max PQ

means top element will always be largest

max PQ




arr [5 3 -2 10 1 7]
PQ
-2
1
3
5
7
10
remove() → -2
" → 1
" " 3
" " 5
" 7
" 10

Syntex :- (default PQ) → ascending order

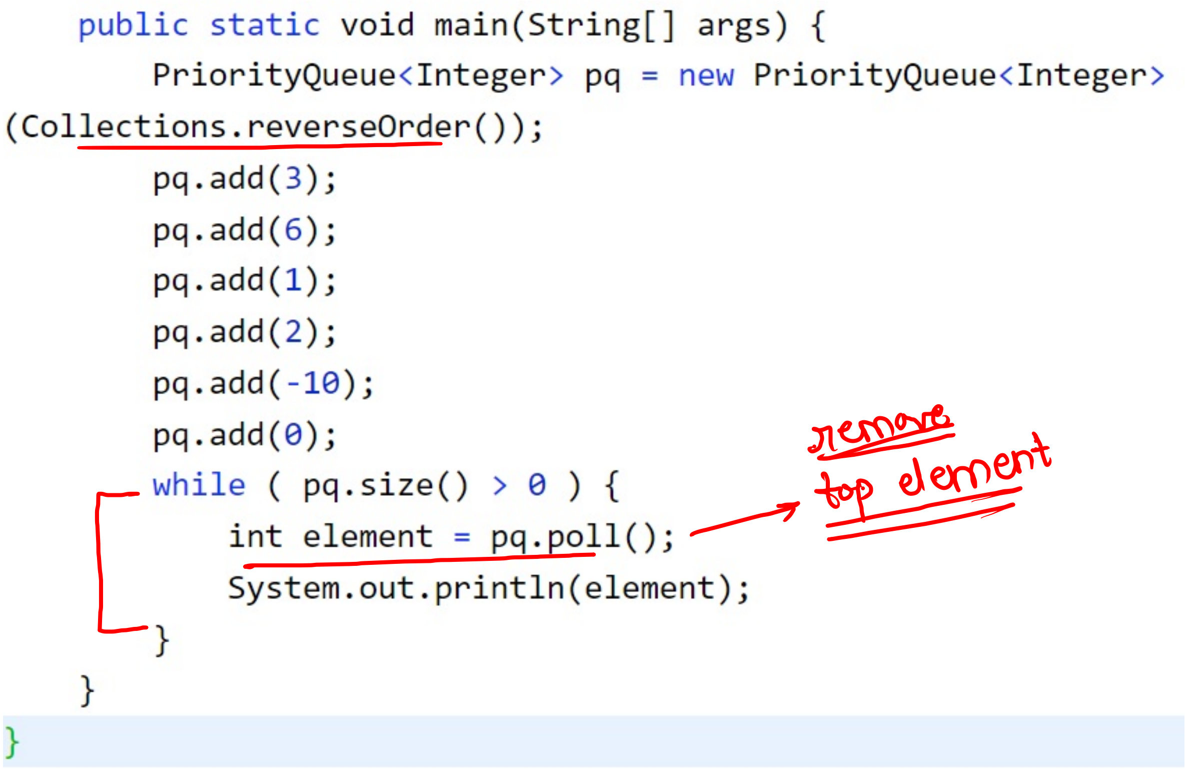
```
PriorityQueue< DataType> pq = new PriorityQueue< >();
```

functions :-

```
pq.add(x);     // to add element
pq.offer(x);
pq.remove();   // to remove element
pq.peek();     // return top element
pq.poll();     // return and remave
               //   top element.
```

→ desending order PQ

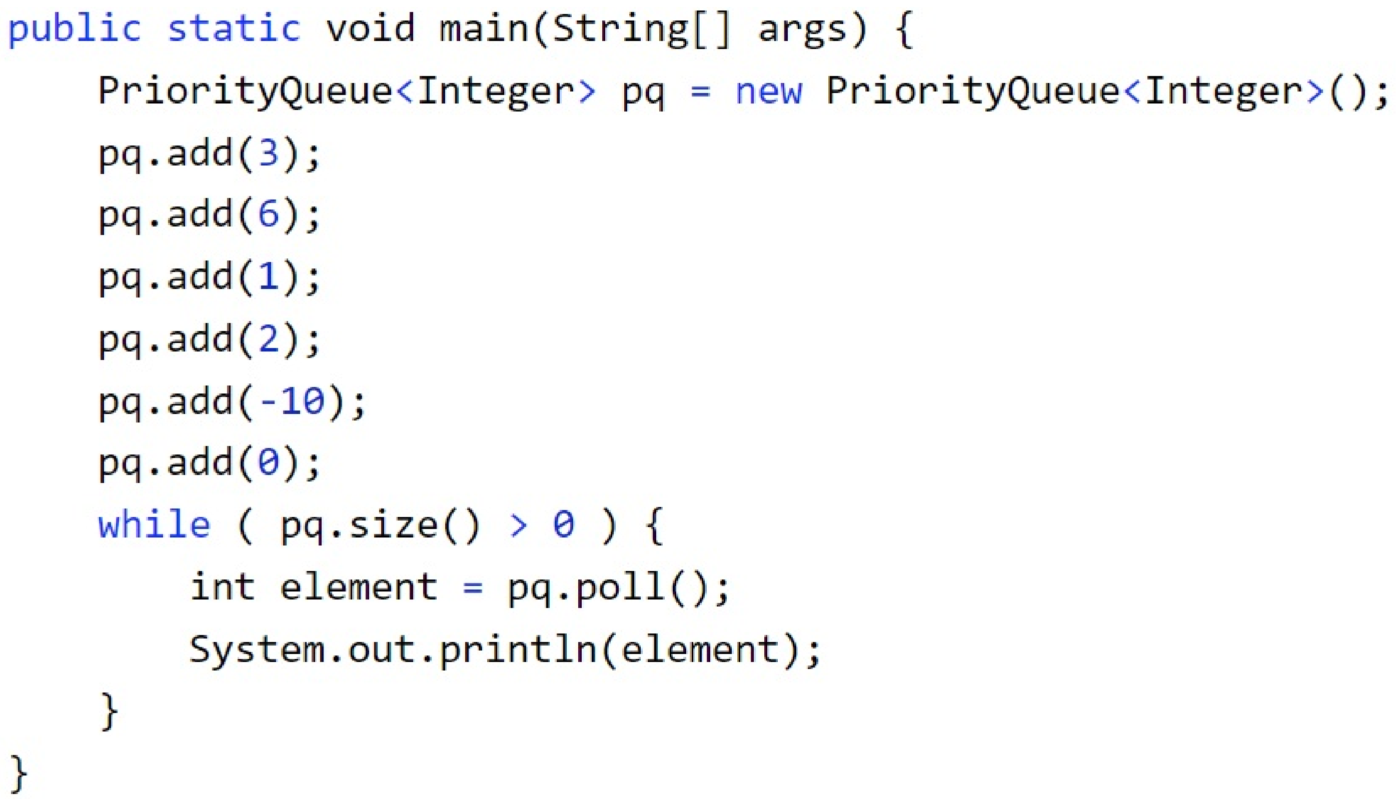
PriorityQueue< DataType > pq =
new PriorityQueue< > (Collections. reverseOrder());



```
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<Integer>
(Collections.reverseOrder());
        pq.add(3);
        pq.add(6);
        pq.add(1);
        pq.add(2);
        pq.add(-10);
        pq.add(0);
        while ( pq.size() > 0 ) {
            int element = pq.poll();
            System.out.println(element);
        }
    }
}
```

remove top element

default PQ in asc. order



```
public static void main(String[] args) {
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    pq.add(3);
    pq.add(6);
    pq.add(1);
    pq.add(2);
    pq.add(-10);
    pq.add(0);
    while ( pq.size() > 0 ) {
        int element = pq.poll();
        System.out.println(element);
    }
}
```

Note:-

we can easily modify the nature of PQ

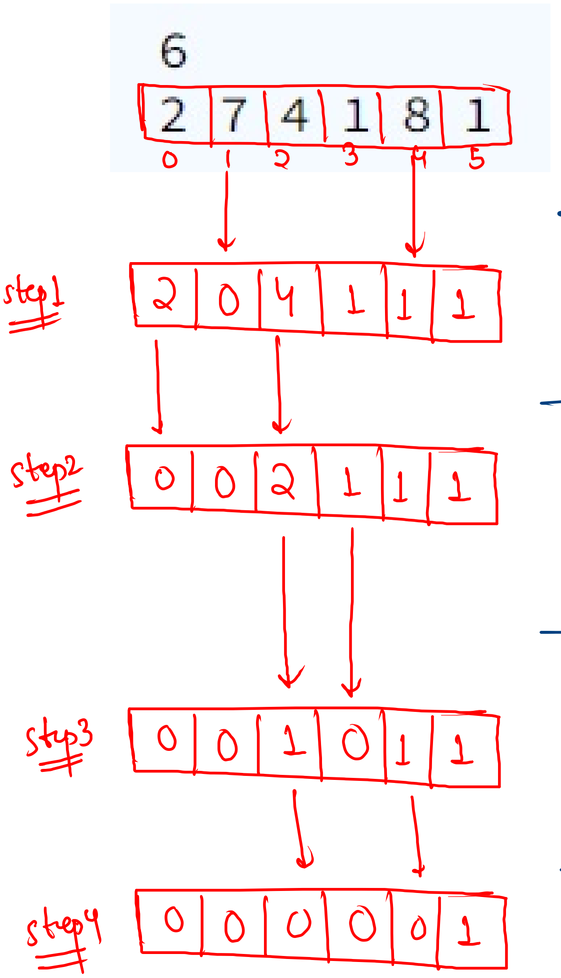
↳ Comparator & Comparable } ✗

↳ lambda function ✓

syntex :-

```
PriorityQueue< DataType>  pq  =  new PriorityQueue< >( (a,b)->{
        // return  a-b ;  // asc. order
           return  b-a ;  // desc. order
  });
```

Break stone



6

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

step1

2	0	4	1	1	1

step2

0	0	2	1	1	1

step3

0	0	1	0	1	1

step4

0	0	0	0	0	1

max1 = ~~8~~ 1

max2 = ~~7~~ 0

max1 = ~~4~~ 2

max2 = ~~2~~ 0

max1 = ~~2~~ 1

max2 = ~~1~~ 0

max1 = ~~1~~ 0

max2 = ~~1~~ 0

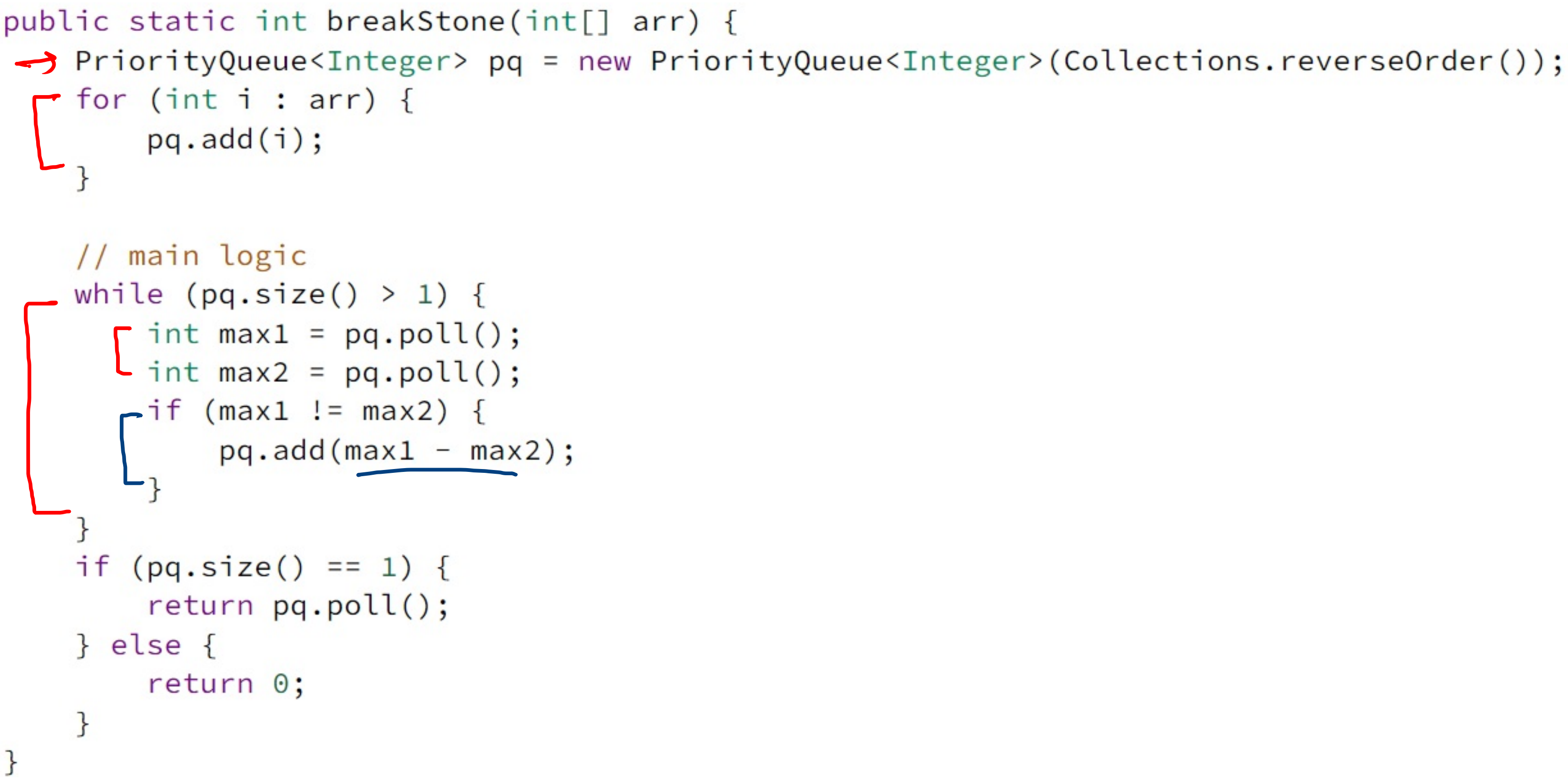
remaining stone = 1

Observation

1) each time we need largest element

★★★ 2) put back element after calculating

⇒ code



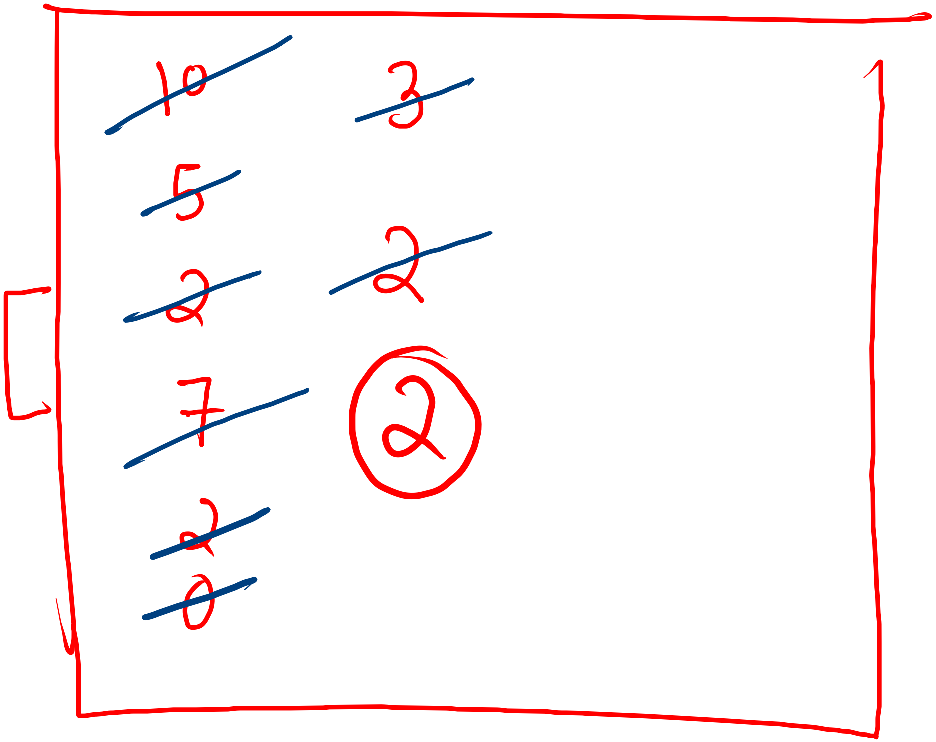
```java
public static int breakStone(int[] arr) {
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>(Collections.reverseOrder());
    for (int i : arr) {
        pq.add(i);
    }

    // main logic
    while (pq.size() > 1) {
        int max1 = pq.poll();
        int max2 = pq.poll();
        if (max1 != max2) {
            pq.add(max1 - max2);
        }
    }
    if (pq.size() == 1) {
        return pq.poll();
    } else {
        return 0;
    }
}
```

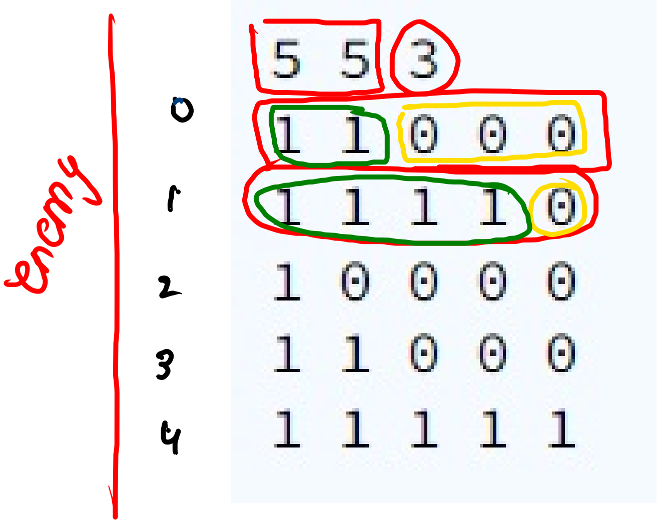
arr = [10, 5, 2, 7, 2, 0]

dry run

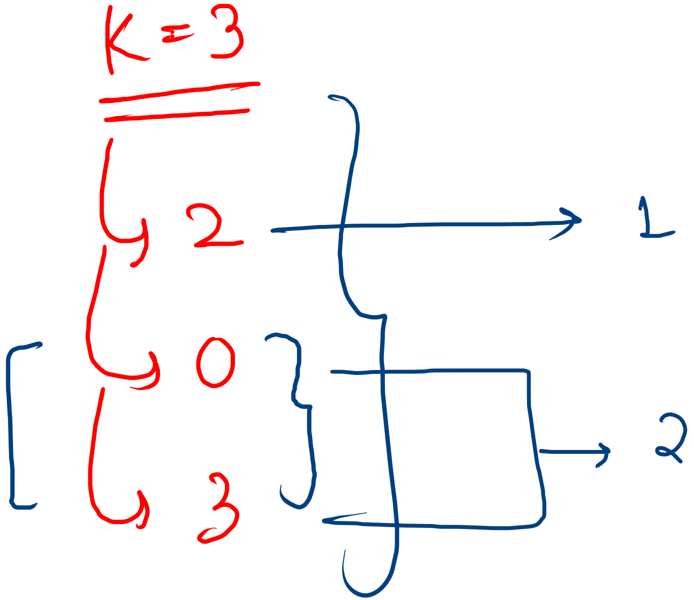
max PQ

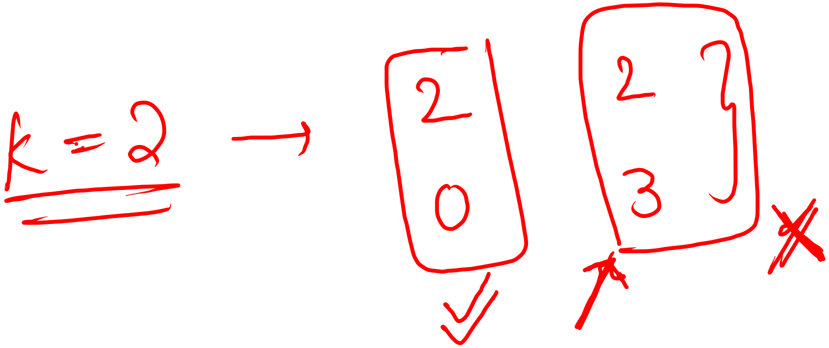


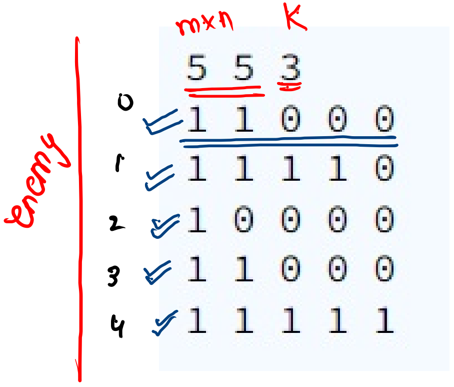
weakest rows



```
    5 5 3
0   1 1 0 0 0
1   1 1 1 1 0
2   1 0 0 0 0
3   1 1 0 0 0
4   1 1 1 1 1
```



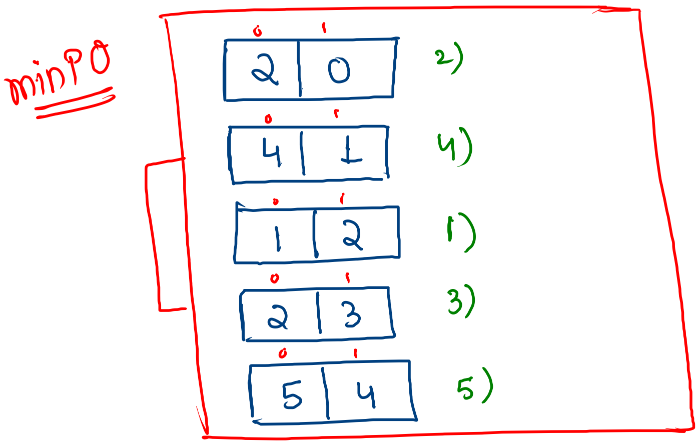




```
5 5 3
1 1 0 0 0
1 1 1 1 0
1 0 0 0 0
1 1 0 0 0
1 1 1 1 1
```

K=3

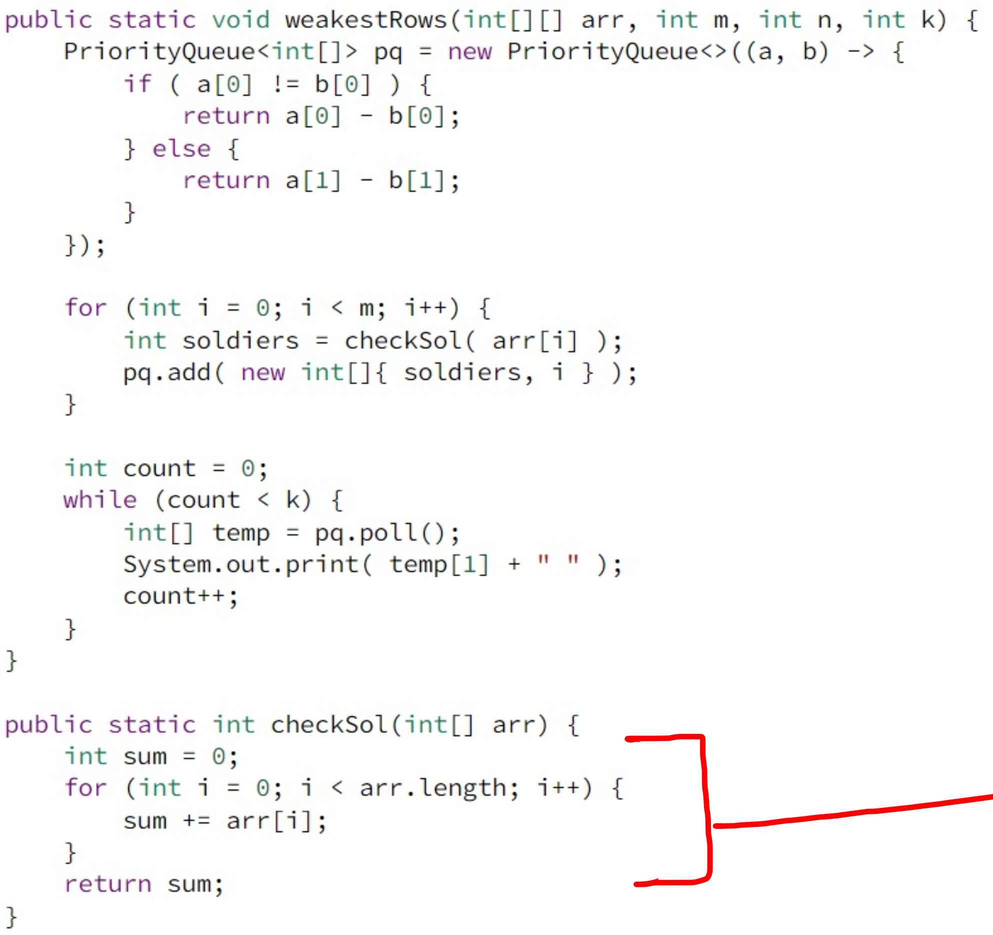
[no. of soldiers, index]



modify PQ, so that we will have smallest no. of soldiers first.

and if same no. of soldiers then sort acc. to index.

Code

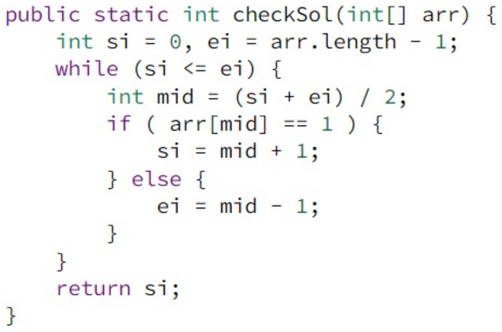


```java
public static void weakestRows(int[][] arr, int m, int n, int k) {
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
        if ( a[0] != b[0] ) {
            return a[0] - b[0];
        } else {
            return a[1] - b[1];
        }
    });

    for (int i = 0; i < m; i++) {
        int soldiers = checkSol( arr[i] );
        pq.add( new int[]{ soldiers, i } );
    }

    int count = 0;
    while (count < k) {
        int[] temp = pq.poll();
        System.out.print( temp[1] + " " );
        count++;
    }
}

public static int checkSol(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
    return sum;
}
```



```java
public static int checkSol(int[] arr) {
    int si = 0, ei = arr.length - 1;
    while (si <= ei) {
        int mid = (si + ei) / 2;
        if ( arr[mid] == 1 ) {
            si = mid + 1;
        } else {
            ei = mid - 1;
        }
    }
    return si;
}
```