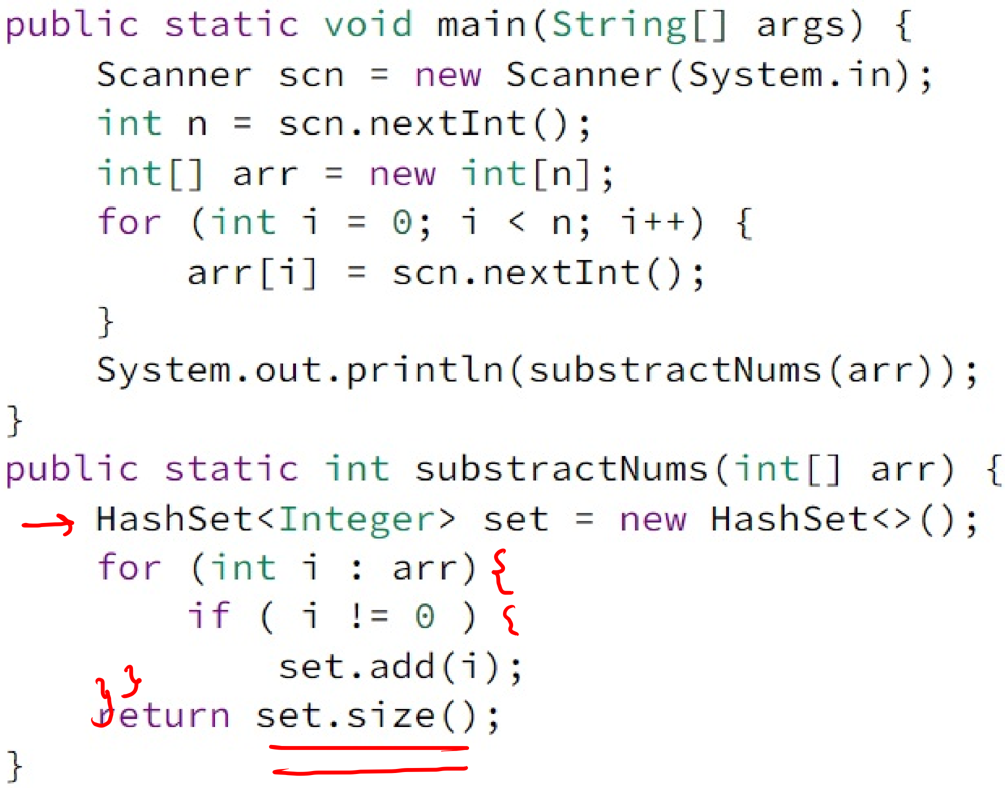
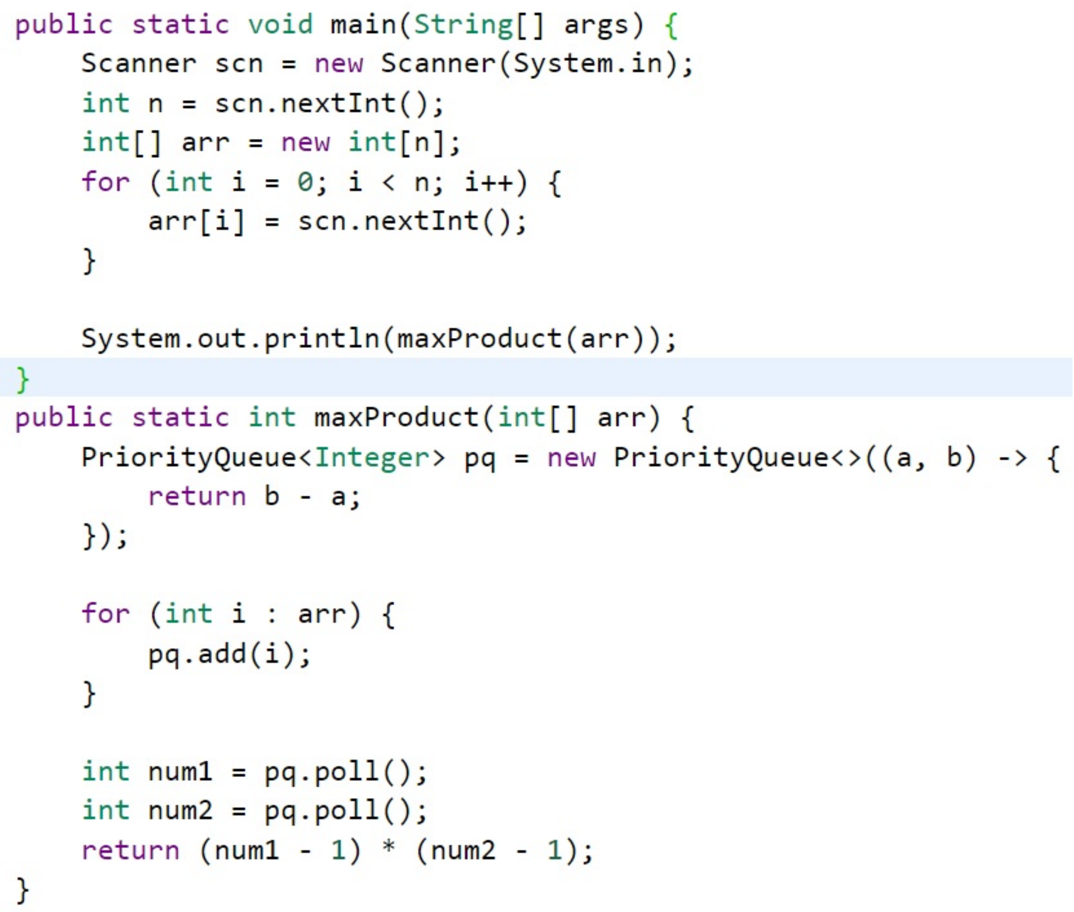
subtract numbers 1



```
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(substractNums(arr));
}
public static int substractNums(int[] arr) {
    HashSet<Integer> set = new HashSet<>();
    for (int i : arr) {
        if ( i != 0 ) {
            set.add(i);
    }}
    return set.size();
}
```

best DS to remove duplicates

Maximum Product of Two Elements in an Array



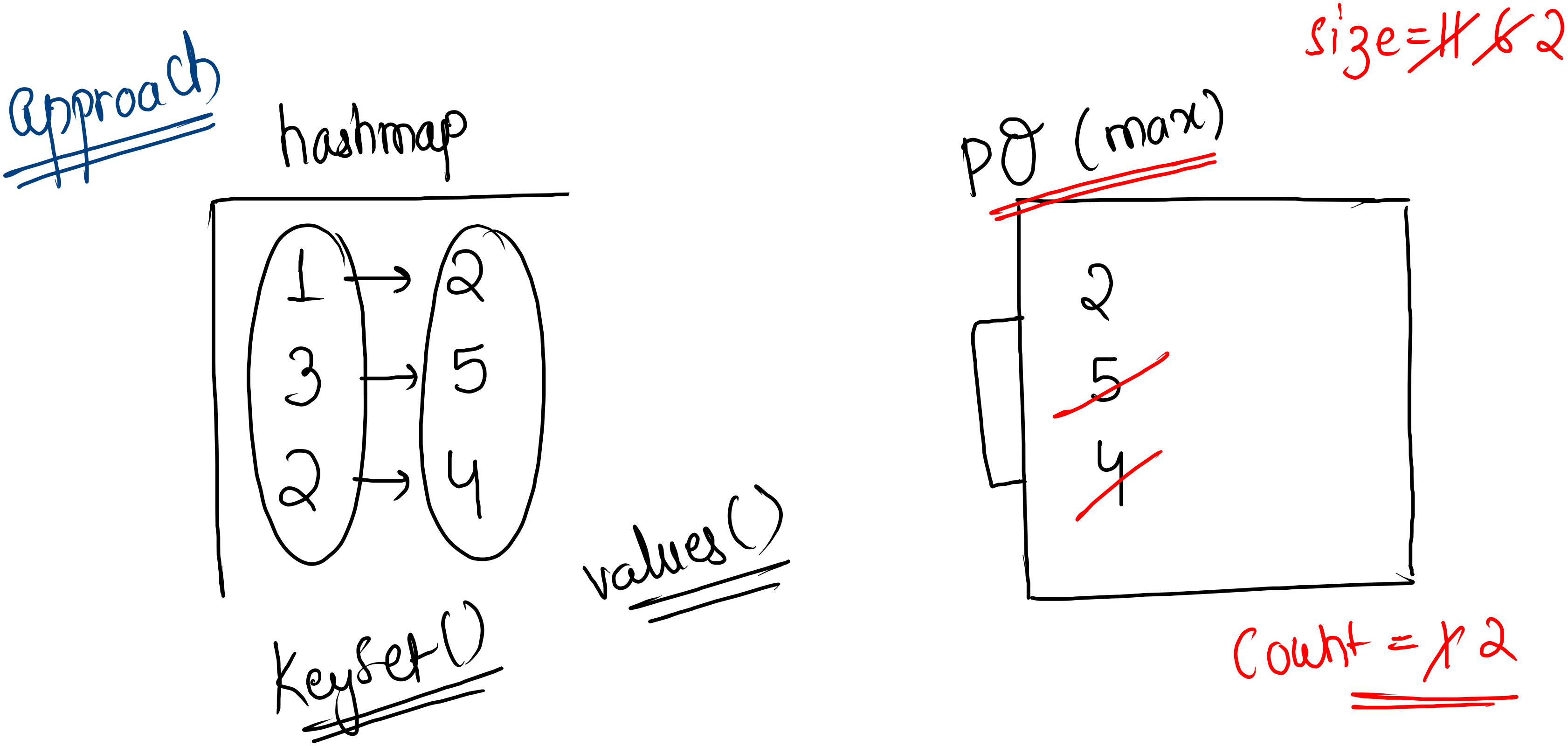
```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    System.out.println(maxProduct(arr));
}
public static int maxProduct(int[] arr) {
    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);
}
```

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

N → size of array

Reduce Array Size to the half 1

	0	1	2	3	4	5	6	7	8	9	10
arr	3	2	3	3	2	1	2	1	2	3	3



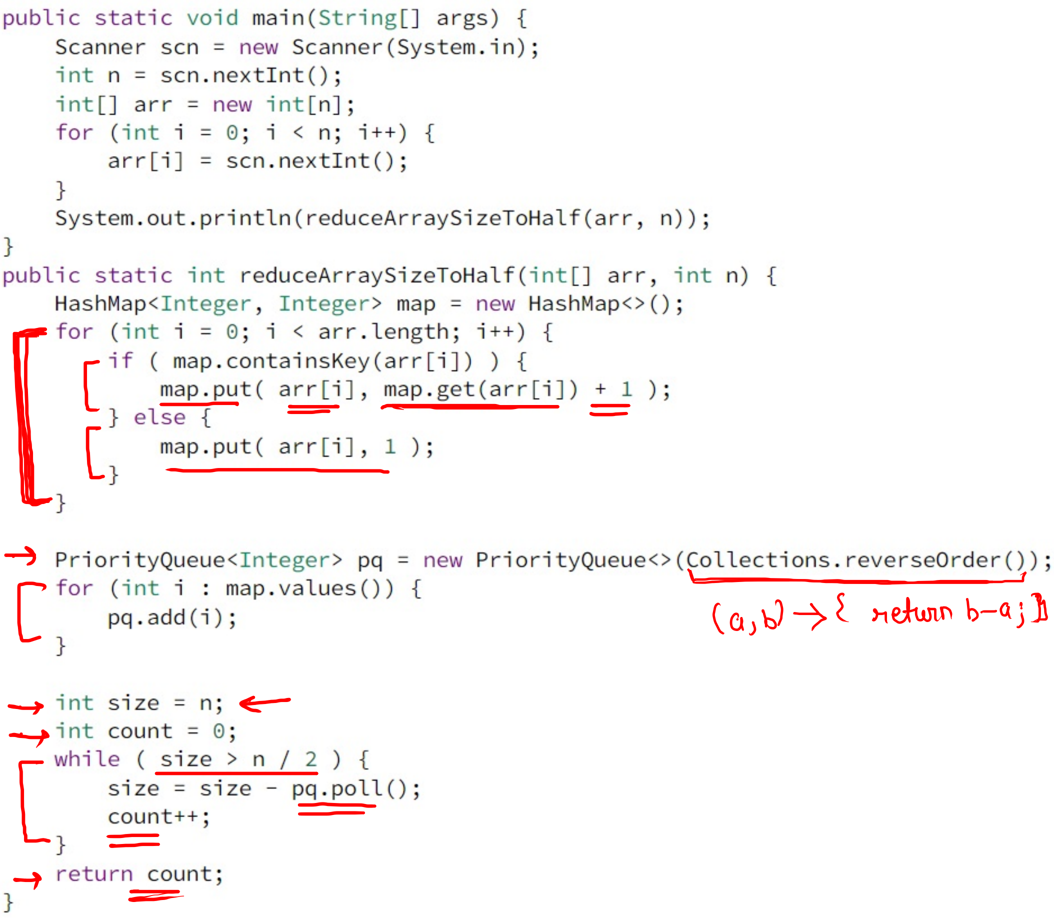
psudo code

```
1) Declare HM
2) Create freq HM
3) Declare PQ (max)
4) Store "values" in PQ
5) iterating until size of array <= half
    5.1) get top element from PQ
    5.2 remove from size of array
    5.3) keep counting
6) print count
```

Code

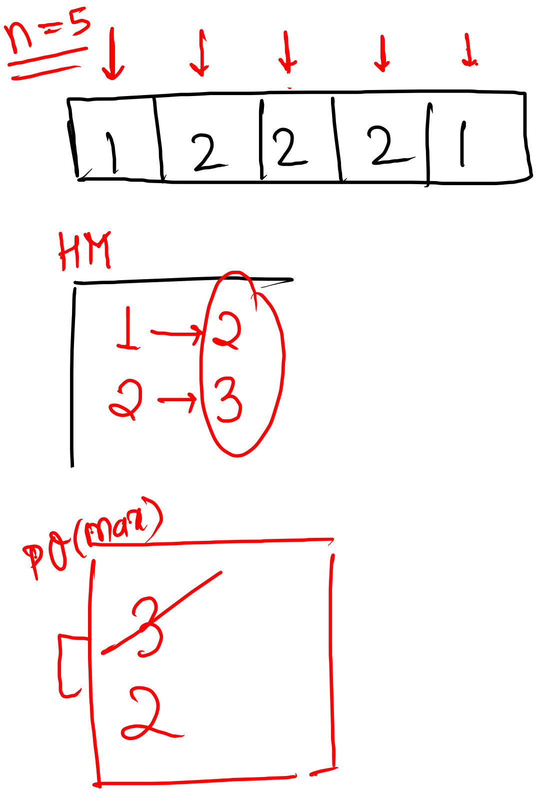
T.C = O(N log N) , N = no. of distinct elements

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

    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
    for (int i : map.values()) {
        pq.add(i);
    }

    int size = n;
    int count = 0;
    while ( size > n / 2 ) {
        size = size - pq.poll();
        count++;
    }
    return count;
}
```

(a,b) → { return b-a; }



size = ~~5~~ 2   5 > 2

count = ~~0~~ 1   2 > 2

weakest rows

(Imp)

arr

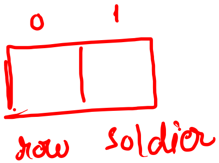
0	1	1	1	1	0	0
1	1	1	0	0	0	0
2	1	1	1	1	1	0
3	1	1	1	0	0	0
4	1	1	0	0	0	0
5	1	1	1	1	0	0

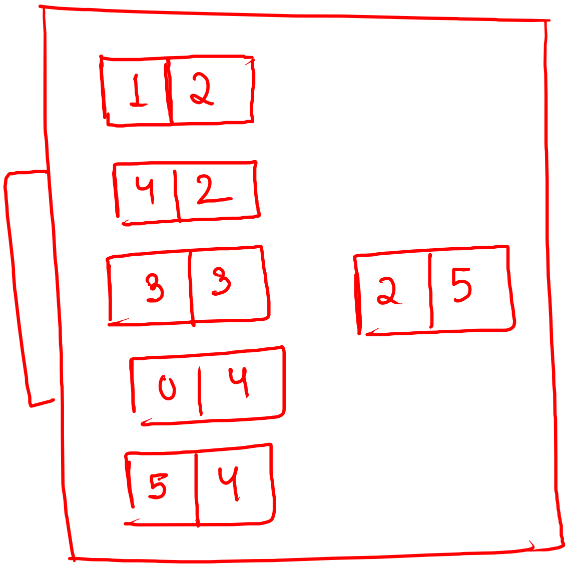
m×n

K = 3

row		soldiers
1	→	2
4	→	2
3	→	3
0	→	4
5	→	4
2	→	5

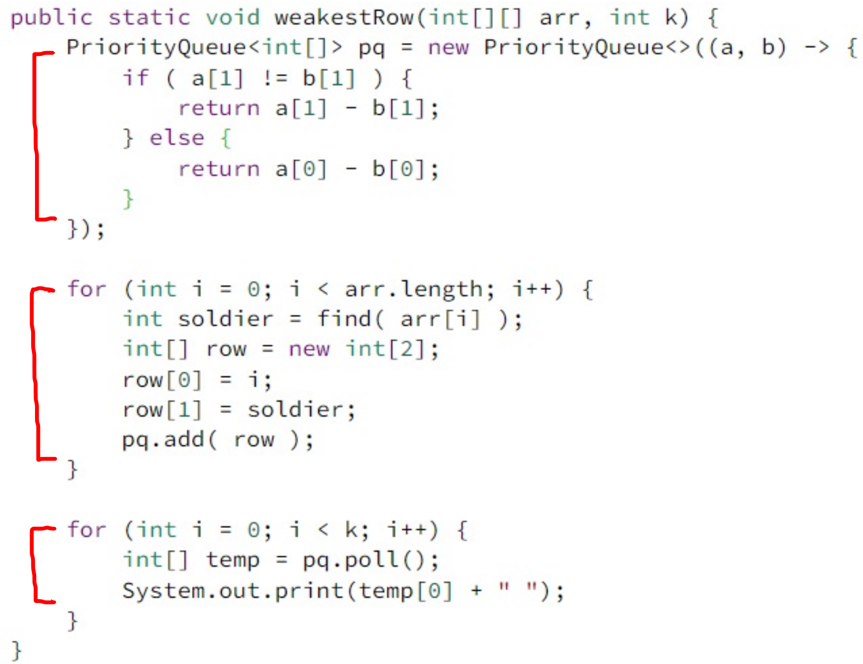
PQ





```
(a,b) -> {
    if( a[1] != b[1] )
        return a[1] - b[1];
    else
        return a[0] - b[0];
}
```

Code

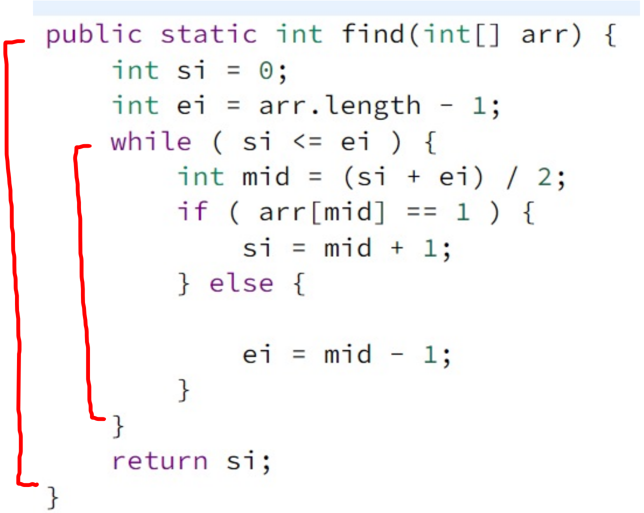


```
public static void weakestRow(int[][] arr, int k) {
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
        if ( a[1] != b[1] ) {
            return a[1] - b[1];
        } else {
            return a[0] - b[0];
        }
    });

    for (int i = 0; i < arr.length; i++) {
        int soldier = find( arr[i] );
        int[] row = new int[2];
        row[0] = i;
        row[1] = soldier;
        pq.add( row );
    }

    for (int i = 0; i < k; i++) {
        int[] temp = pq.poll();
        System.out.print(temp[0] + " ");
    }
}
```

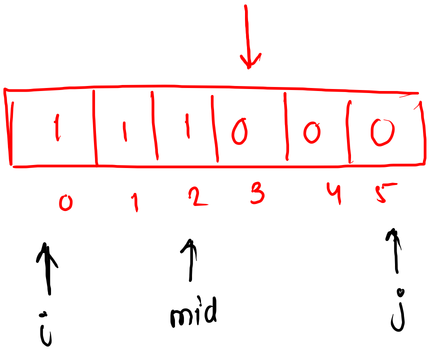
S.C = O(M)

M = rows

T.C = O(N logN)



```
public static int find(int[] arr) {
    int si = 0;
    int 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;
}
```


1
1
1
0
0
0
0
1
2
3
4
5
i
mid
j