Name: Devesh

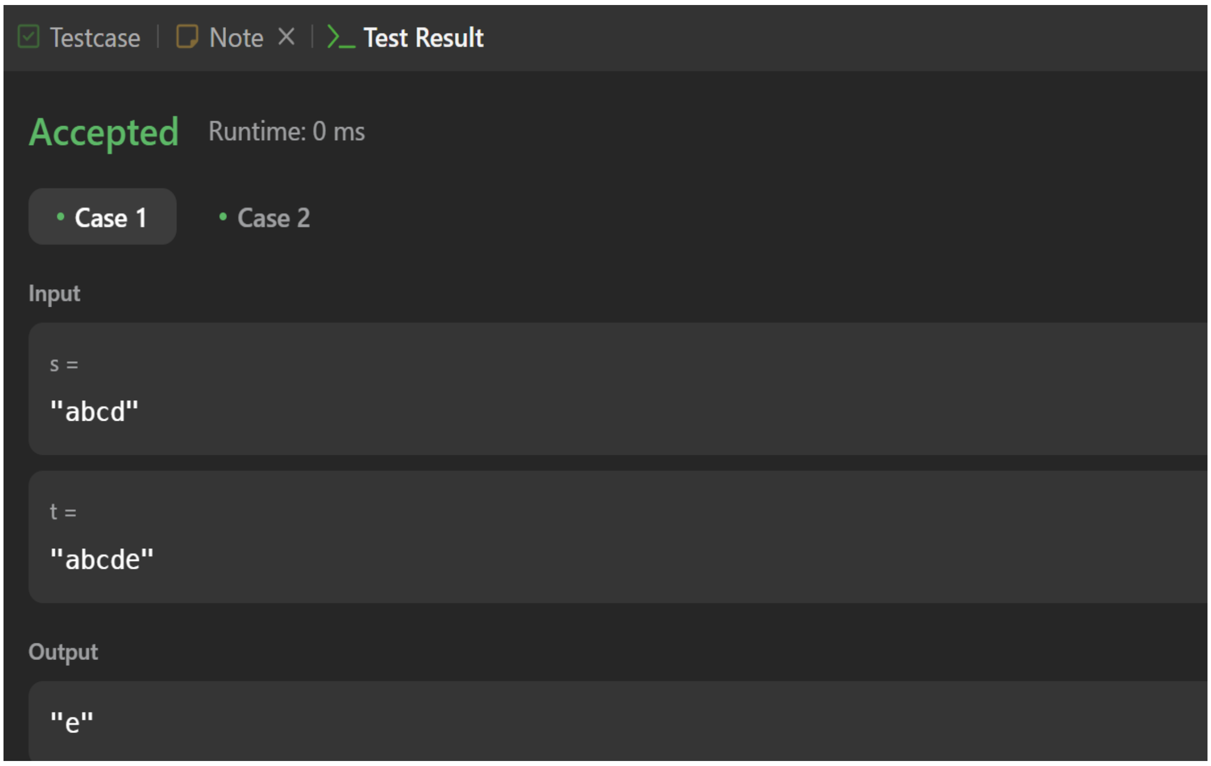
UID: 22BCS16690

Class – 605 -B

Q 1 Find the difference

```
class Solution {
   public char findTheDifference(String s, String t) {
      char miss = 0;
      int len = t.length();
      for(int i = 0; i < len; i++) {
         if(i < s.length()) {
            miss ^= s.charAt(i);
         }
         miss ^= t.charAt(i);
      }
      return miss;
   }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

```
s =
"abcd"
t =
"abcde"
```

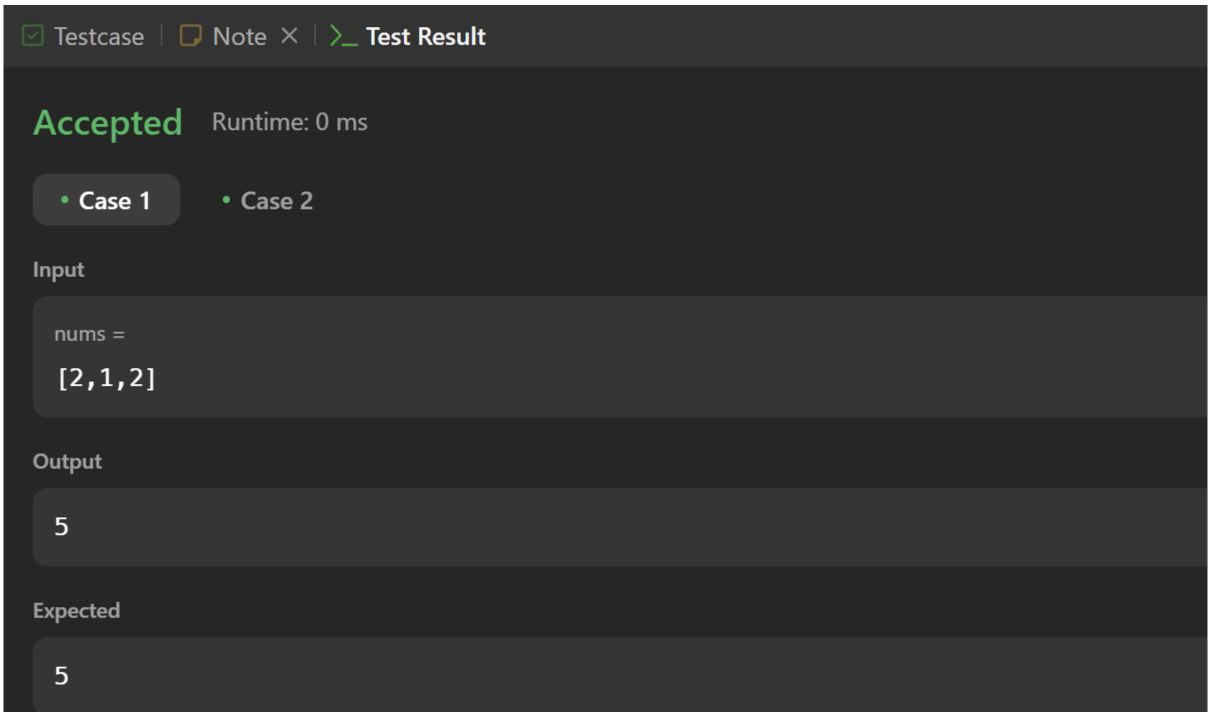
Output

```
"e"
```

Q 2 Largest Perimeter Triangle

```
class Solution {
    public int largestPerimeter(int[] nums) {
        Arrays.sort(nums);
        int n=nums.length;
        for(int i=n-1;i>=2;i--){
            if(nums[i-2]+nums[i-1]>nums[i]){
                return nums[i-2]+nums[i-1]+nums[i];
            }
        }
        return 0;
    }
}
```

OUTPUT:

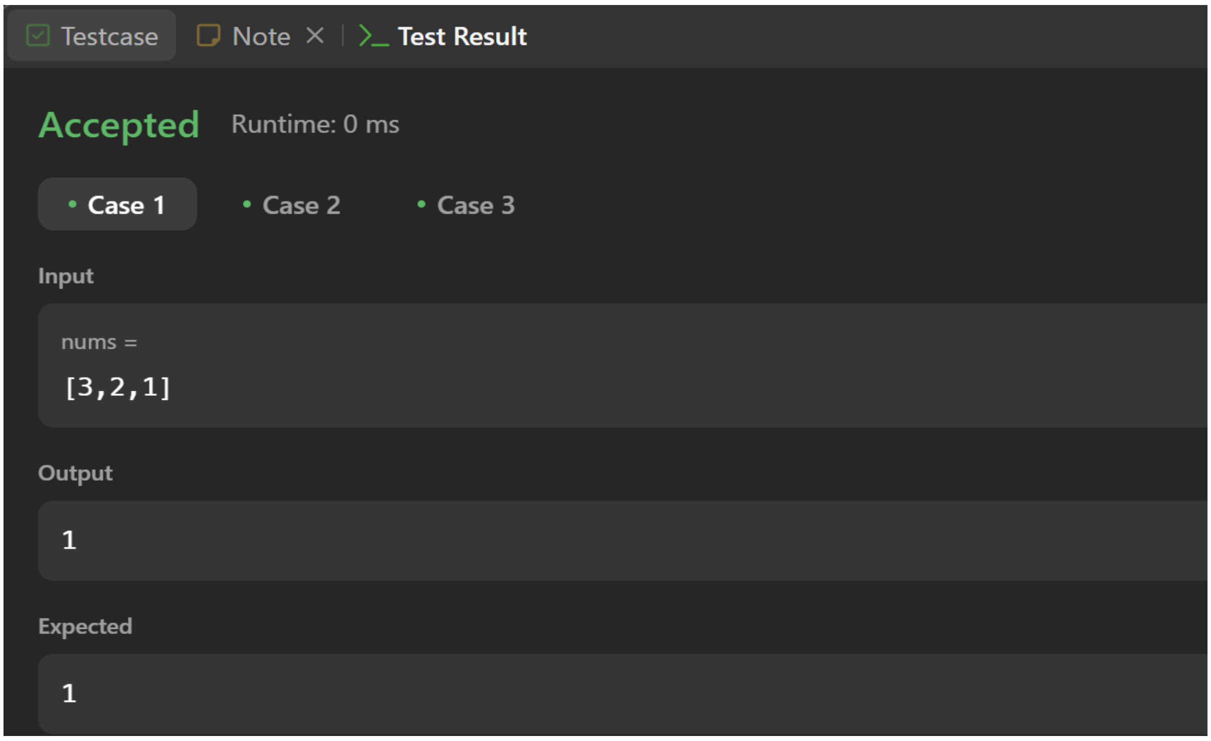


```
Accepted   Runtime: 0 ms

Input
nums =
[2,1,2]

Output
5

Expected
5
```

Q3 Third Maximum Number

```
class Solution {
    public int thirdMax(int[] nums) {
        Set<Integer> s = new HashSet<>();
        for (int n : nums) s.add(n);
        if (s.size() < 3) return Collections.max(s);
        s.remove(Collections.max(s));
        s.remove(Collections.max(s));
        return Collections.max(s);
    }
}
```

OUTPUT:

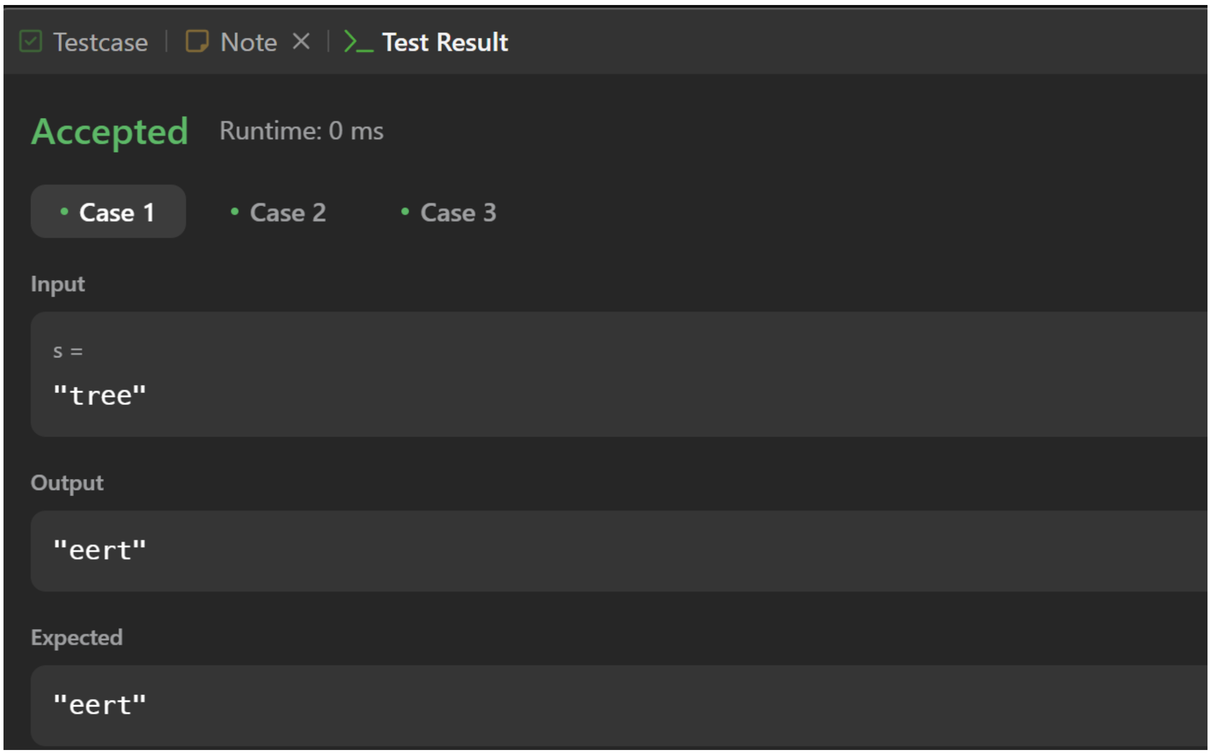


```
Accepted    Runtime: 0 ms

Input
nums =
[3,2,1]

Output
1

Expected
1
```

Q 4 Sort Characters By Frequency

```
class Solution {
    public String frequencySort(String s) {
        HashMap<Character, Integer> map = new HashMap<>();
        for (char ch : s.toCharArray()) {
            map.put(ch, map.getOrDefault(ch, 0) + 1);
        }
        PriorityQueue<Character> pq = new PriorityQueue<>((a, b) -> map.get(b) - 
map.get(a));
        for (char ch : map.keySet()) {
            pq.add(ch); }
        StringBuilder sb = new StringBuilder();
        while (!pq.isEmpty()) {
            char c = pq.poll();
            int frequency = map.get(c);
            for (int i = 0; i < frequency; i++) {
                sb.append(c);
            }
        }
        return sb.toString();
    }
}
```

OUTPUT:

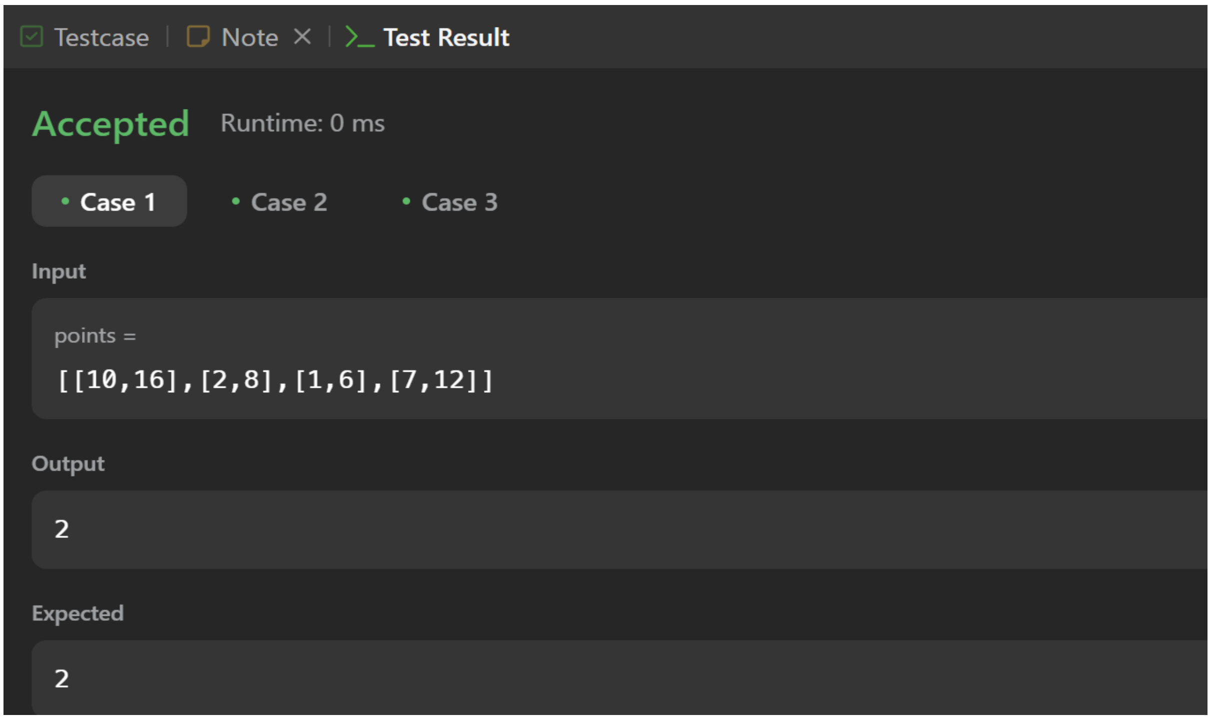


```
Accepted   Runtime: 0 ms

Input
s =
"tree"

Output
"eert"

Expected
"eert"
```

Q 5 Minimum Number of Arrows to Burst Balloons

```
class Solution {
    public int findMinArrowShots(int[][] segments) {
        Arrays.sort(segments, (a, b) -> Integer.compare(a[1], b[1]));
        int ans = 0, arrow = 0;
        for (int i = 0; i < segments.length; i ++) {
            if (ans == 0 || segments[i][0] > arrow) {
                ans ++;
                arrow = segments[i][1];
            }
        }
        return ans;
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
points =
[[10,16],[2,8],[1,6],[7,12]]
```

Output

```
2
```

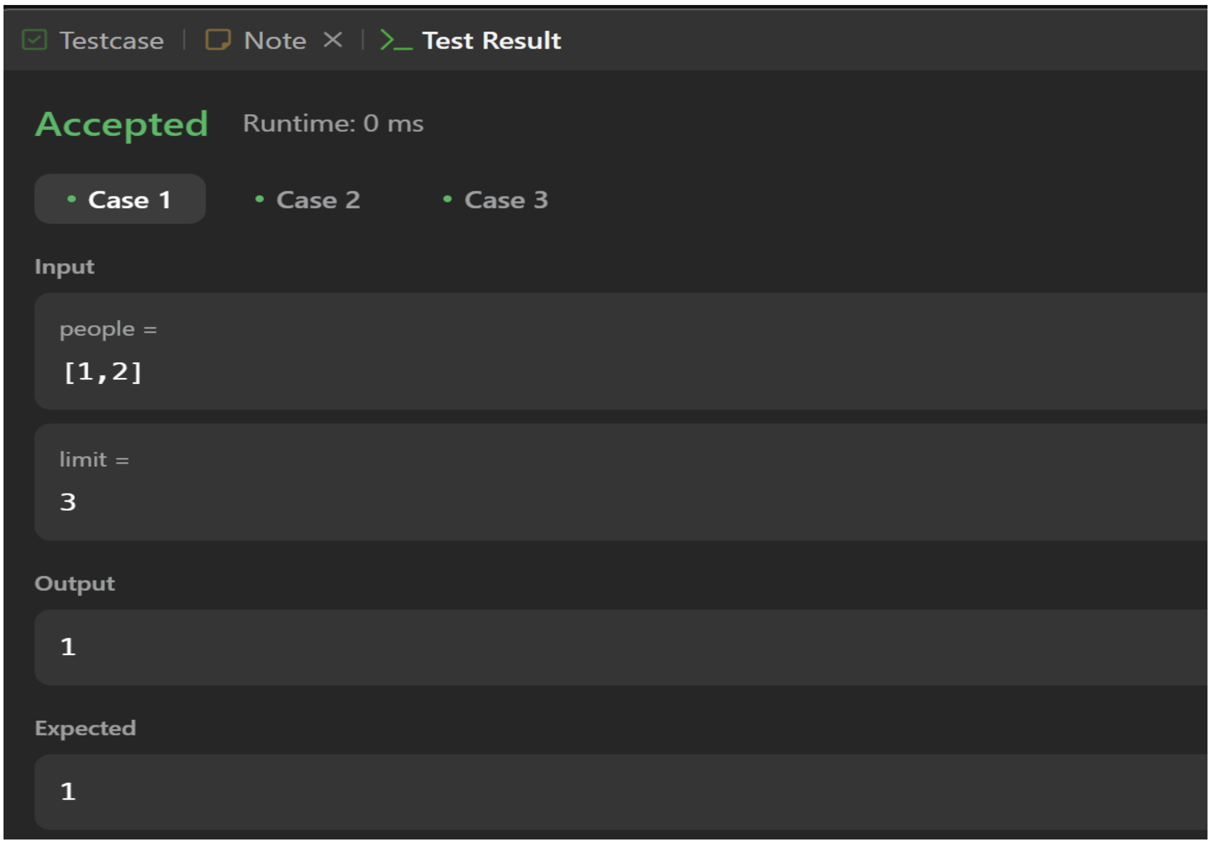
Expected

```
2
```

Q 6 Boats to Save People

```
class Solution {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people);
        int high = people.length-1;
        int low = 0;
        int numOfBoats = 0;
        while(low <= high){
            if(people[low] + people[high] <= limit){
                low++;
            }
            high--;
            numOfBoats++;
        }
        return numOfBoats;
    }
}
```

OUTPUT:



Testcase | Note × | >_ Test Result

Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input

```
people =
[1,2]
```

```
limit =
3
```

Output

```
1
```

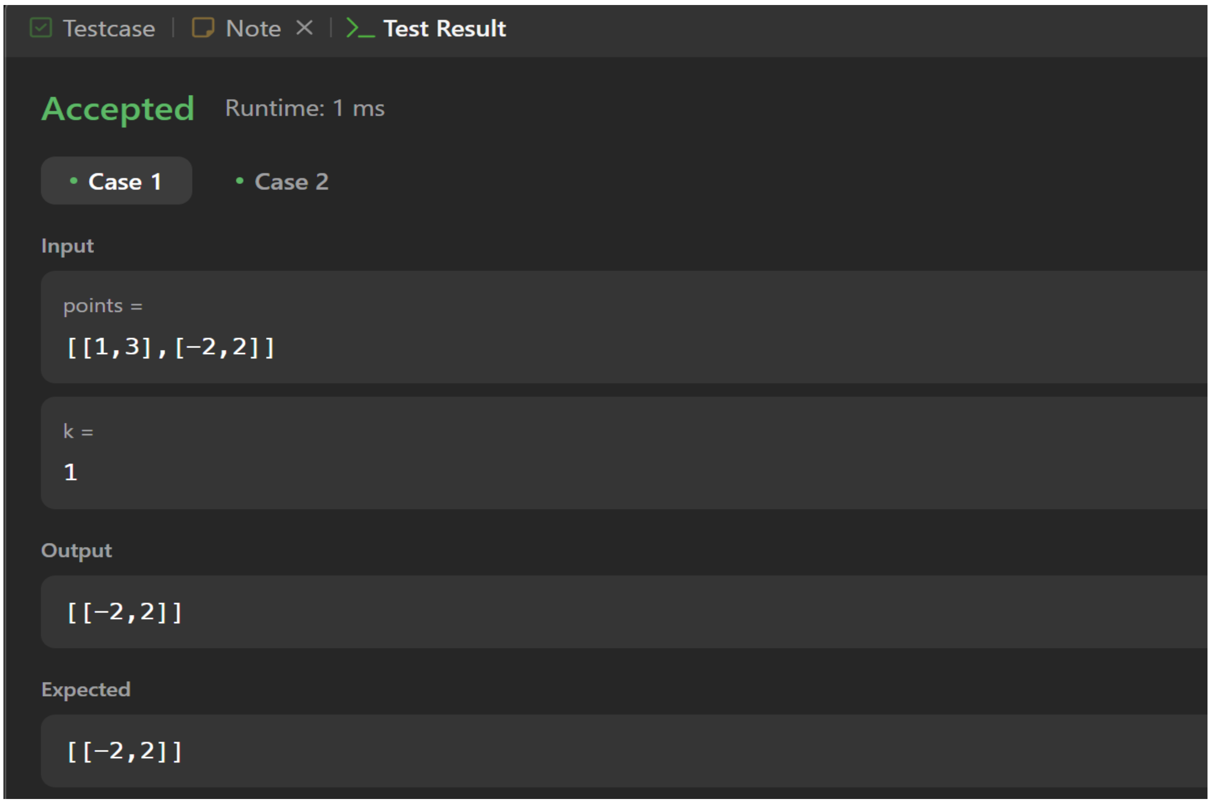
Expected

```
1
```

Q 7 K Closest Points to Origin

```
class Solution {
    public int[][] kClosest(int[][] points, int k) {
        PriorityQueue<int[]> pq = new PriorityQueue<>(
            (p1, p2) -> Double.compare(getDistance(p1), getDistance(p2))
        );
        for (int i = 0; i < points.length; i++) {
            pq.add(points[i]);
        }
        int[][] result = new int[k][2];
        for (int i = 0; i < k; i++) {
            result[i] = pq.poll();
        }
        return result;
    }
    private double getDistance(int[] point) {
        return Math.sqrt(point[0] * point[0] + point[1] * point[1]);
    }
}
```

OUTPUT:



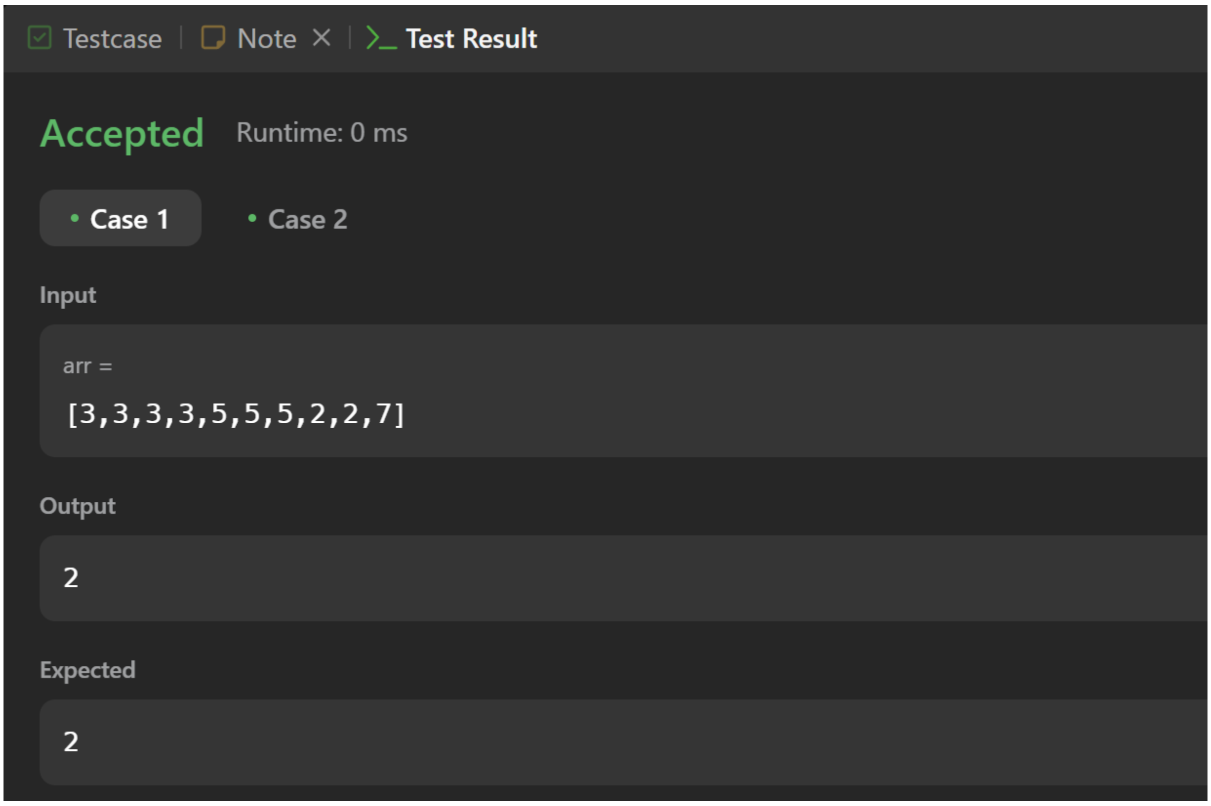
```
Accepted    Runtime: 1 ms

Case 1    Case 2

Input
points =
[[1,3],[-2,2]]

k =
1

Output
[[-2,2]]

Expected
[[-2,2]]
```

Q 8 Reduce Array Size to The Half

```
  class Solution {
  public int minSetSize(int[] arr) {
  Map<Integer, Integer> countMap = new HashMap<>();
  PriorityQueue<Integer> countValues = new   PriorityQueue<>(Comparator.reverseOrder());
      for (int num : arr) countMap.put(num, countMap.getOrDefault(num, 0) + 1);
```

```
        for (int value : countMap.values()) countValues.offer(value);
        int size = arr.length; int result = 0;
        while (size > arr.length / 2) {
            size -= countValues.poll();
            result++;
        }
        return result;
    }
}
```

OUTPUT:



Testcase | Note × | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

arr =

```
[3,3,3,3,5,5,5,2,2,7]
```

Output

```
2
```

Expected

```
2
```