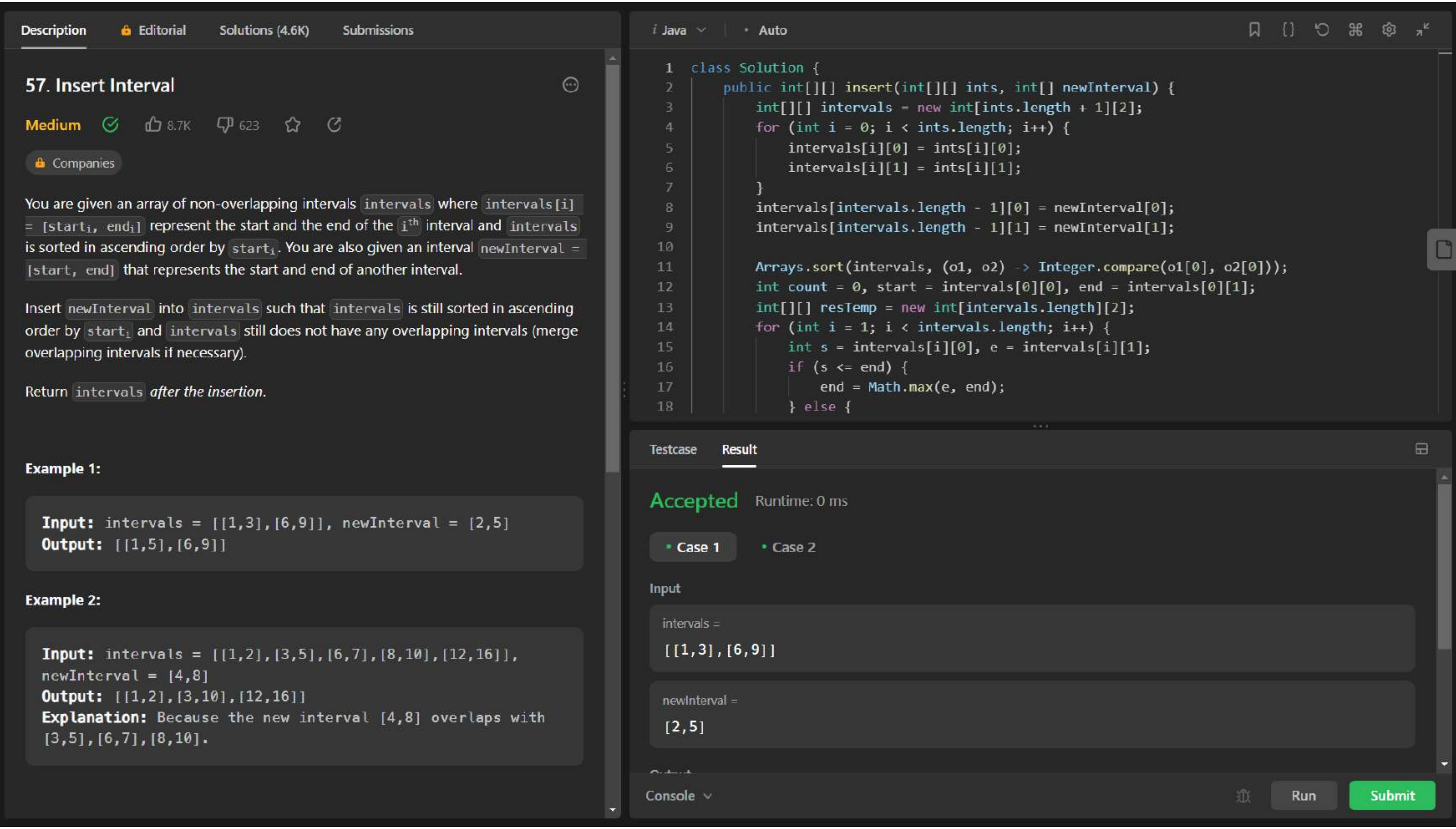


57. Insert Interval

Medium 8.7K 623

Companies

You are given an array of non-overlapping intervals `intervals` where `intervals[i] = [start_i, end_i]` represent the start and the end of the i^{th} interval and `intervals` is sorted in ascending order by `start_i`. You are also given an interval `newInterval = [start, end]` that represents the start and end of another interval.

Insert `newInterval` into `intervals` such that `intervals` is still sorted in ascending order by `start_i` and `intervals` still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return `intervals` *after the insertion.*

Example 1:

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]],
newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with
[3,5],[6,7],[8,10].
```

```java
class Solution {
    public int[][] insert(int[][] ints, int[] newInterval) {
        int[][] intervals = new int[ints.length + 1][2];
        for (int i = 0; i < ints.length; i++) {
            intervals[i][0] = ints[i][0];
            intervals[i][1] = ints[i][1];
        }
        intervals[intervals.length - 1][0] = newInterval[0];
        intervals[intervals.length - 1][1] = newInterval[1];

        Arrays.sort(intervals, (o1, o2) -> Integer.compare(o1[0], o2[0]));
        int count = 0, start = intervals[0][0], end = intervals[0][1];
        int[][] resTemp = new int[intervals.length][2];
        for (int i = 1; i < intervals.length; i++) {
            int s = intervals[i][0], e = intervals[i][1];
            if (s <= end) {
                end = Math.max(e, end);
            } else {
```

Testcase   Result

Accepted   Runtime: 0 ms

Case 1   Case 2

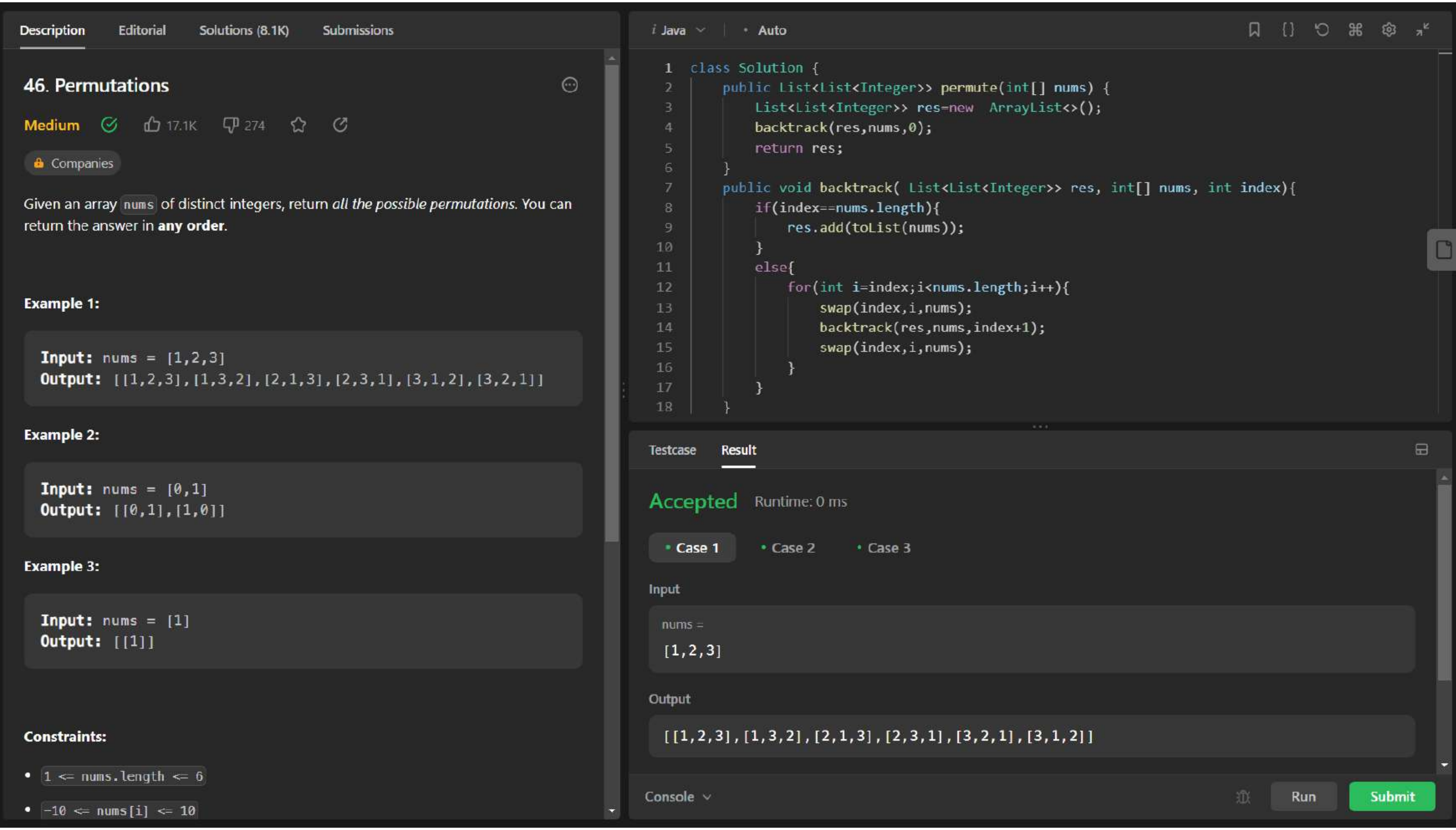
Input

intervals =

```
[[1,3],[6,9]]
```

newInterval =

```
[2,5]
```



46. Permutations

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

```
Input: nums = [1]
Output: [[1]]
```

Constraints:

- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> res=new  ArrayList<>();
        backtrack(res,nums,0);
        return res;
    }
    public void backtrack( List<List<Integer>> res, int[] nums, int index){
        if(index==nums.length){
            res.add(toList(nums));
        }
        else{
            for(int i=index;i<nums.length;i++){
                swap(index,i,nums);
                backtrack(res,nums,index+1);
                swap(index,i,nums);
            }
        }
    }
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =

```
[1,2,3]
```

Output

```
[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,2,1],[3,1,2]]
```