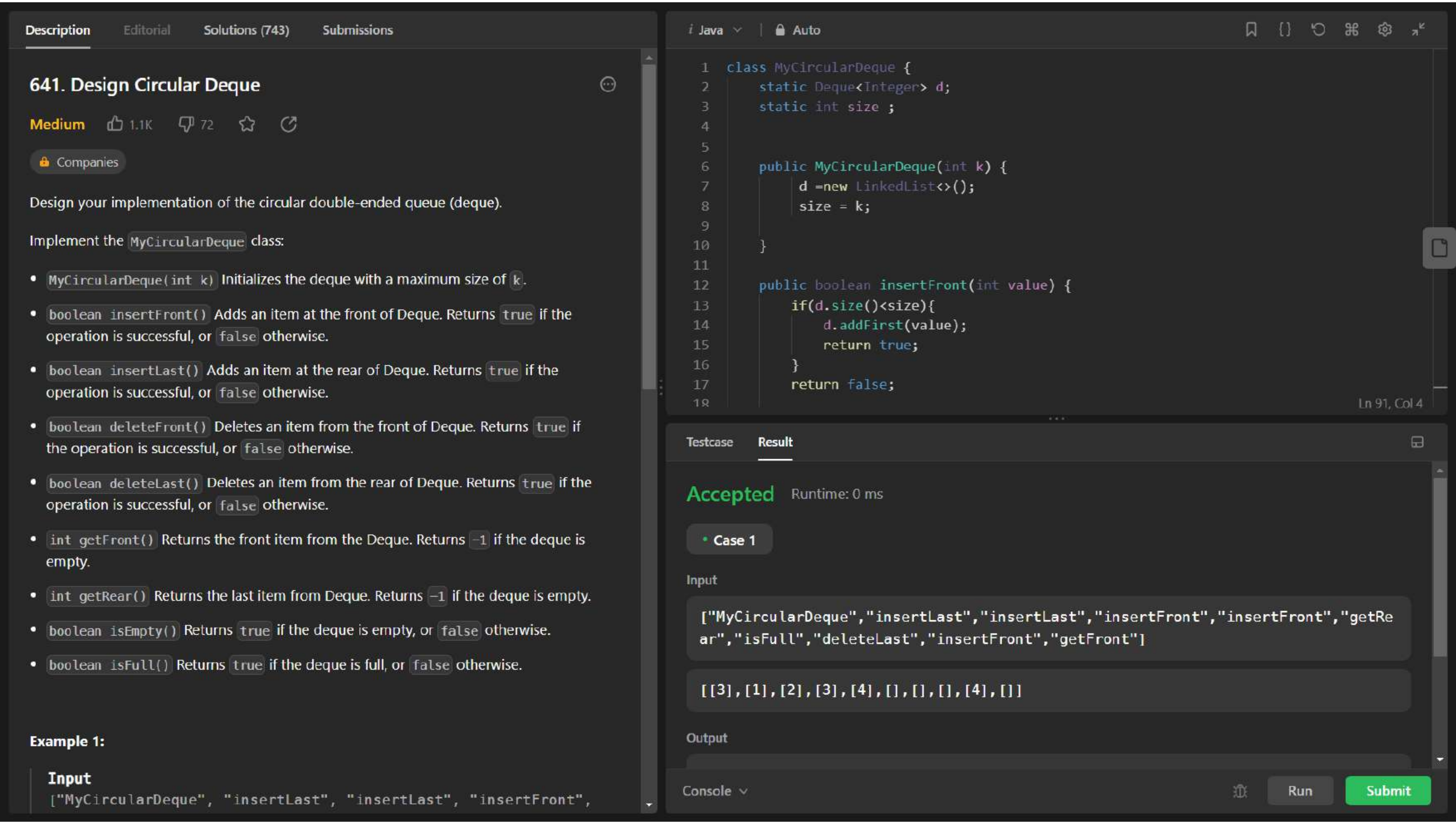


641. Design Circular Deque

Design your implementation of the circular double-ended queue (deque).

Implement the `MyCircularDeque` class:

- `MyCircularDeque(int k)` Initializes the deque with a maximum size of `k`.
- `boolean insertFront()` Adds an item at the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean insertLast()` Adds an item at the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteFront()` Deletes an item from the front of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `boolean deleteLast()` Deletes an item from the rear of Deque. Returns `true` if the operation is successful, or `false` otherwise.
- `int getFront()` Returns the front item from the Deque. Returns `-1` if the deque is empty.
- `int getRear()` Returns the last item from Deque. Returns `-1` if the deque is empty.
- `boolean isEmpty()` Returns `true` if the deque is empty, or `false` otherwise.
- `boolean isFull()` Returns `true` if the deque is full, or `false` otherwise.

Example 1:

Input
```
["MyCircularDeque", "insertLast", "insertLast", "insertFront",
```

```java
class MyCircularDeque {
    static Deque<Integer> d;
    static int size ;


    public MyCircularDeque(int k) {
        d =new LinkedList<>();
        size = k;

    }

    public boolean insertFront(int value) {
        if(d.size()<size){
            d.addFirst(value);
            return true;
        }
        return false;
```

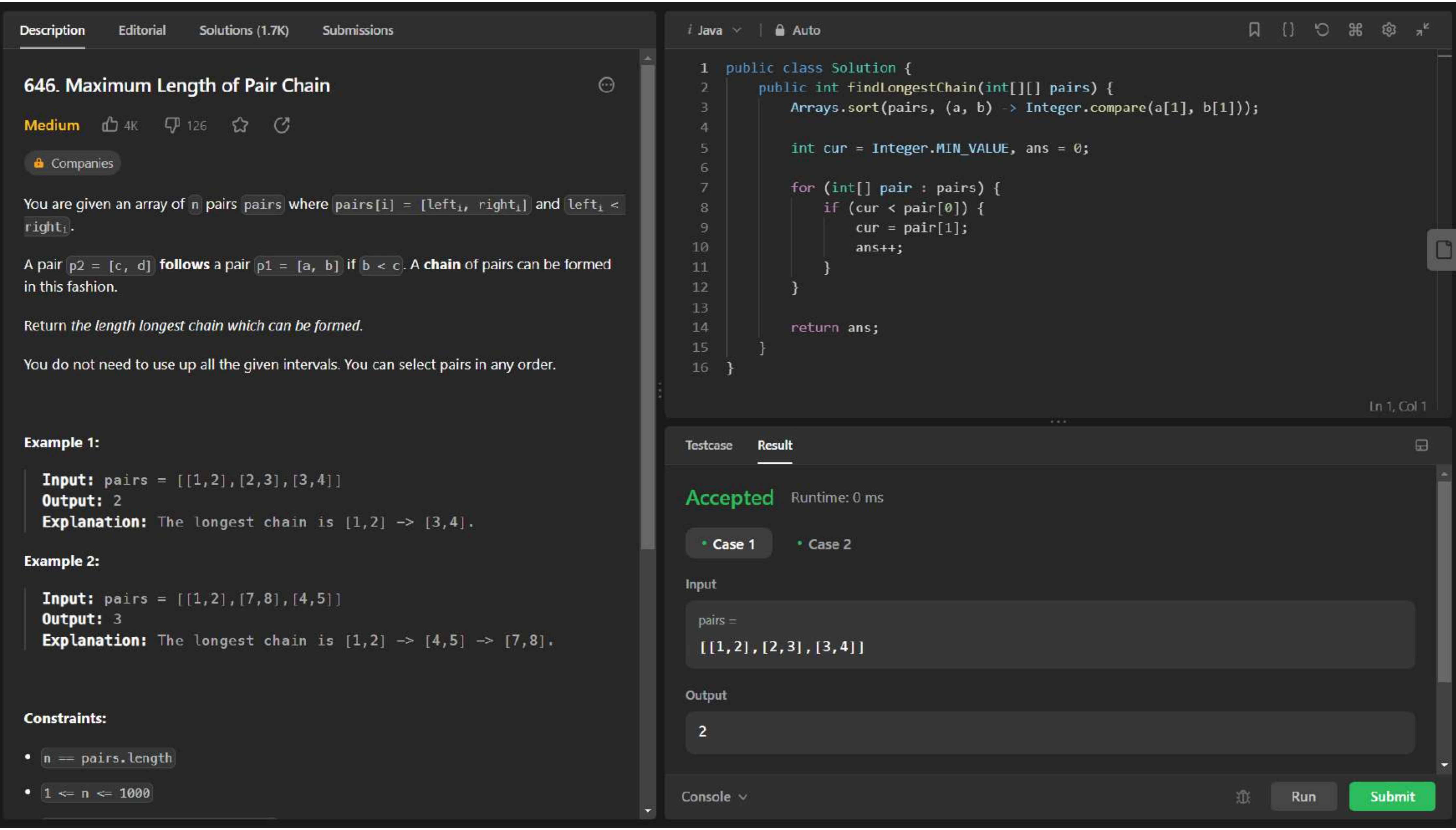
Accepted Runtime: 0 ms

Input

```
["MyCircularDeque","insertLast","insertLast","insertFront","insertFront","getRear","isFull","deleteLast","insertFront","getFront"]
```

```
[[3],[1],[2],[3],[4],[],[],[],[4],[]]
```

Output



646. Maximum Length of Pair Chain

You are given an array of `n` pairs `pairs` where `pairs[i] = [left_i, right_i]` and `left_i < right_i`.

A pair `p2 = [c, d]` **follows** a pair `p1 = [a, b]` if `b < c`. A **chain** of pairs can be formed in this fashion.

Return *the length longest chain which can be formed.*

You do not need to use up all the given intervals. You can select pairs in any order.

Example 1:

```
Input: pairs = [[1,2],[2,3],[3,4]]
Output: 2
Explanation: The longest chain is [1,2] -> [3,4].
```

Example 2:

```
Input: pairs = [[1,2],[7,8],[4,5]]
Output: 3
Explanation: The longest chain is [1,2] -> [4,5] -> [7,8].
```

Constraints:

- `n == pairs.length`
- `1 <= n <= 1000`

```java
public class Solution {
    public int findLongestChain(int[][] pairs) {
        Arrays.sort(pairs, (a, b) -> Integer.compare(a[1], b[1]));

        int cur = Integer.MIN_VALUE, ans = 0;

        for (int[] pair : pairs) {
            if (cur < pair[0]) {
                cur = pair[1];
                ans++;
            }
        }

        return ans;
    }
}
```

Ln 1, Col 1

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

pairs =

```
[[1,2],[2,3],[3,4]]
```

Output

```
2
```