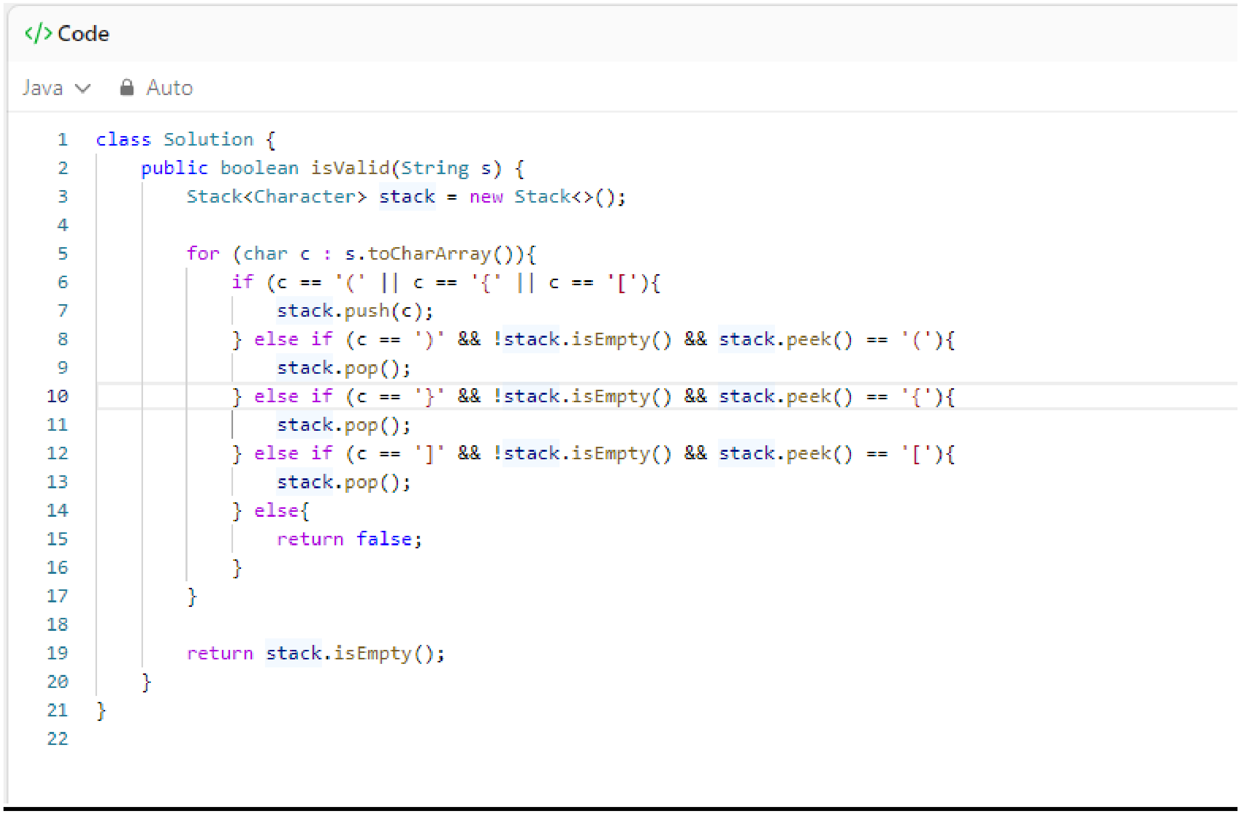
<u>TASK 01:</u> Valid Parentheses

<u>Coding:</u>

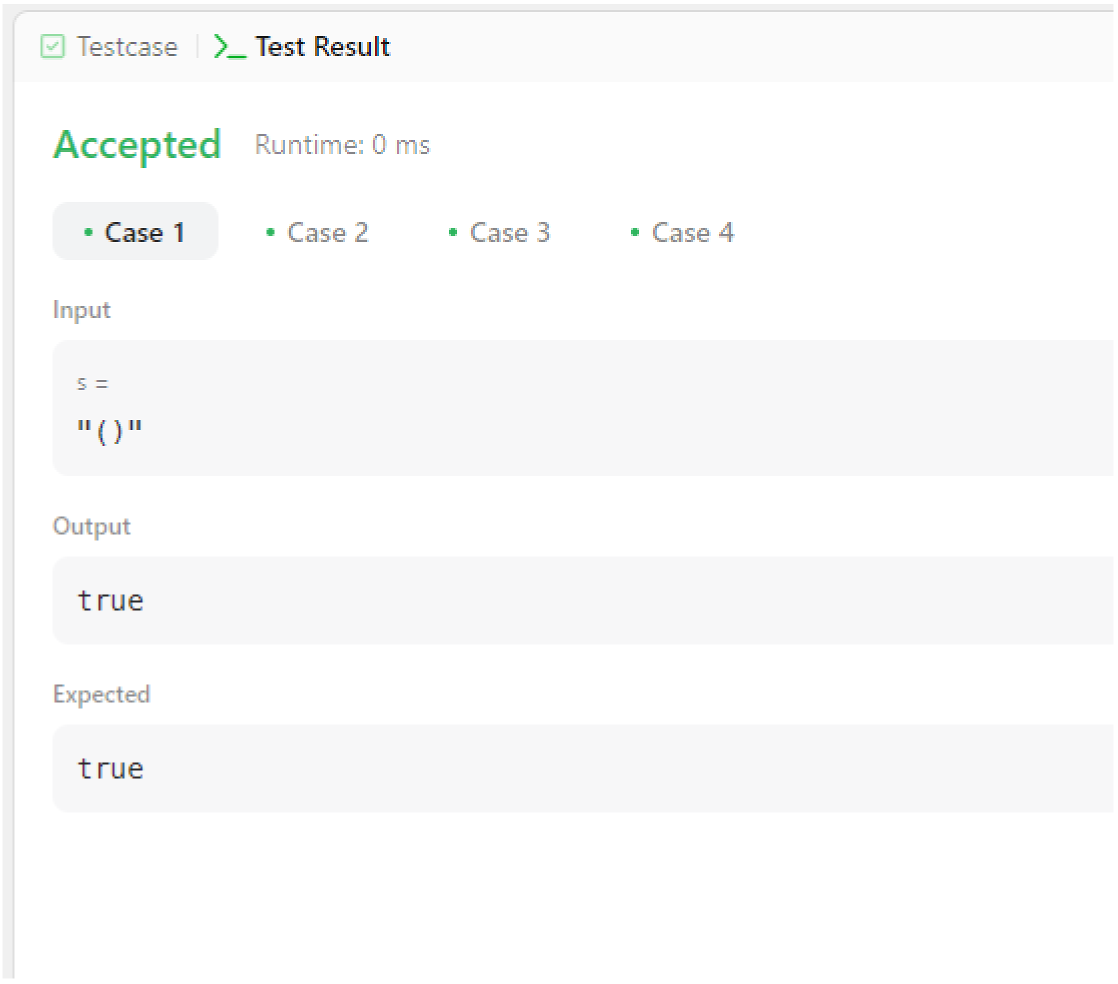


</> Code

Java Auto

```java
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()){
            if (c == '(' || c == '{' || c == '['){
                stack.push(c);
            } else if (c == ')' && !stack.isEmpty() && stack.peek() == '('){
                stack.pop();
            } else if (c == '}' && !stack.isEmpty() && stack.peek() == '{'){
                stack.pop();
            } else if (c == ']' && !stack.isEmpty() && stack.peek() == '['){
                stack.pop();
            } else{
                return false;
            }
        }

        return stack.isEmpty();
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

```
s =
"()"
```

Output

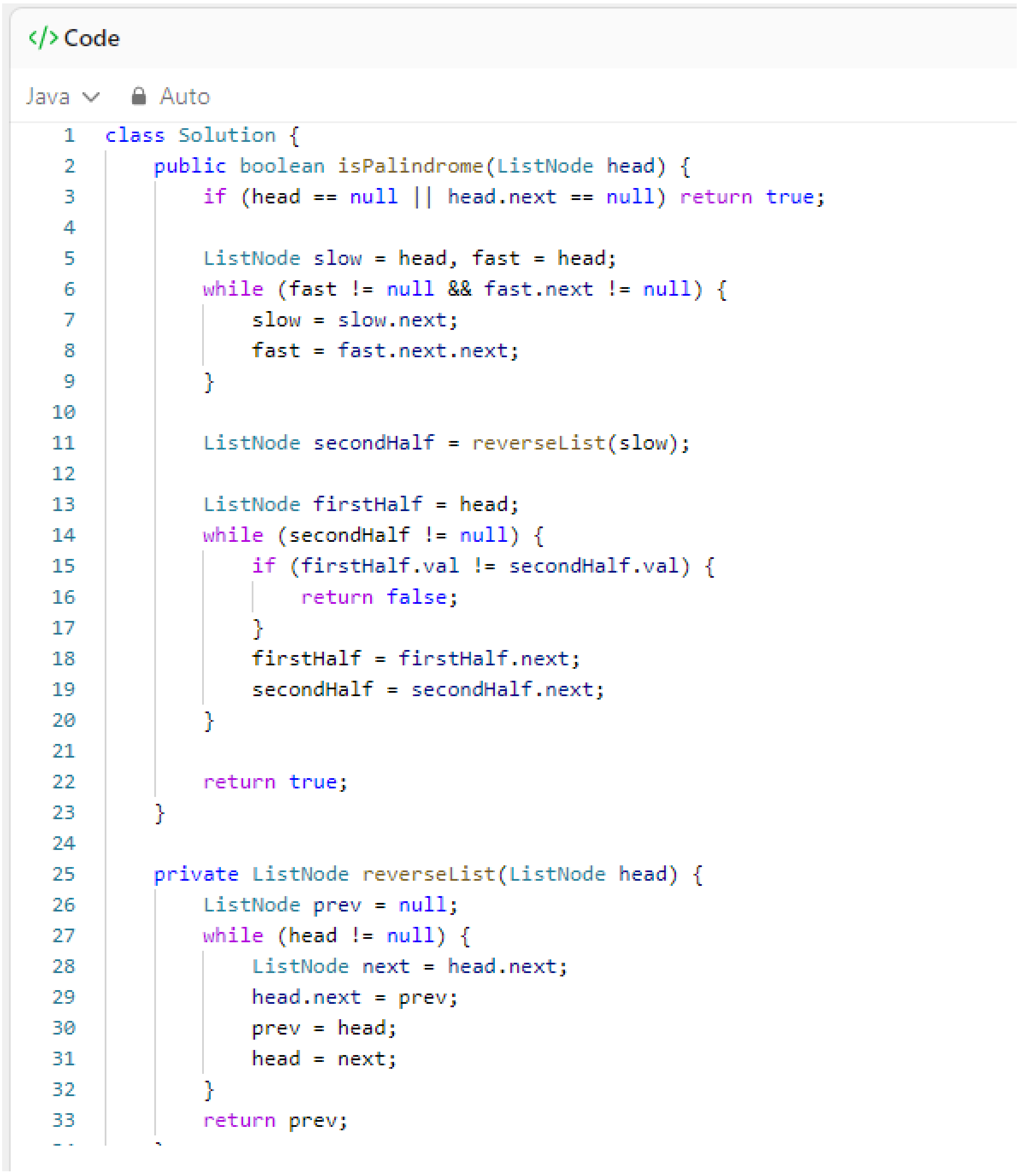
```
true
```

Expected

```
true
```

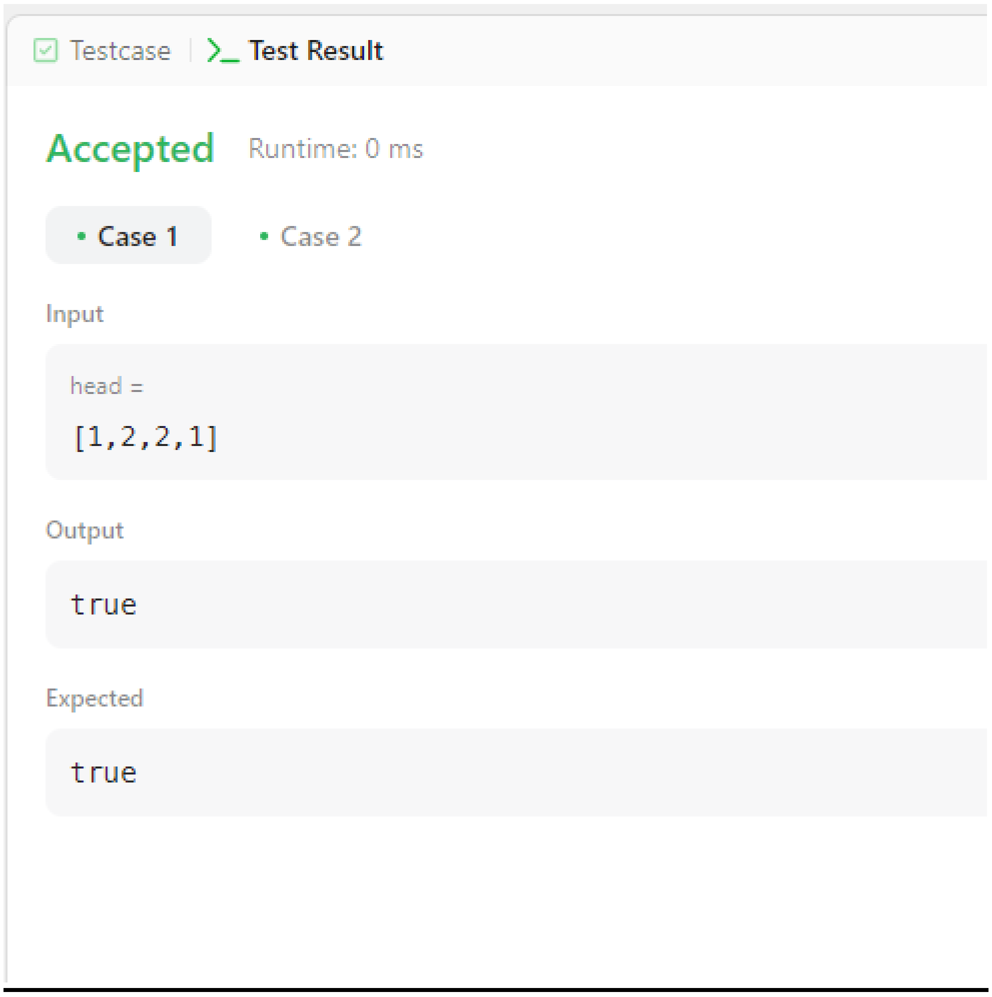
TASK 02: **Palindrome Linked List**

Coding:



```java
class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null || head.next == null) return true;

        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        ListNode secondHalf = reverseList(slow);

        ListNode firstHalf = head;
        while (secondHalf != null) {
            if (firstHalf.val != secondHalf.val) {
                return false;
            }
            firstHalf = firstHalf.next;
            secondHalf = secondHalf.next;
        }

        return true;
    }

    private ListNode reverseList(ListNode head) {
        ListNode prev = null;
        while (head != null) {
            ListNode next = head.next;
            head.next = prev;
            prev = head;
            head = next;
        }
        return prev;
```

OUTPUT:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

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

Output

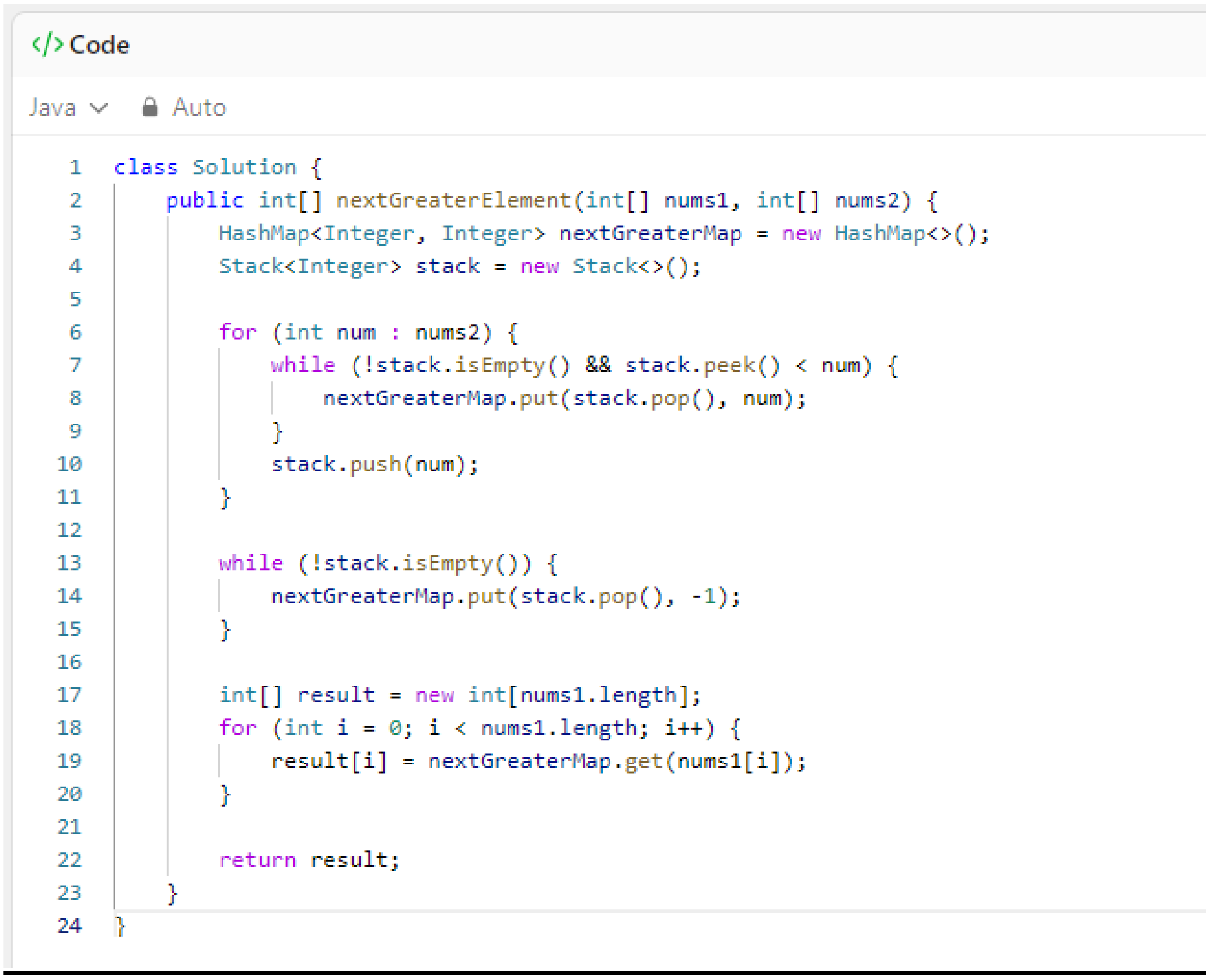
```
true
```

Expected

```
true
```

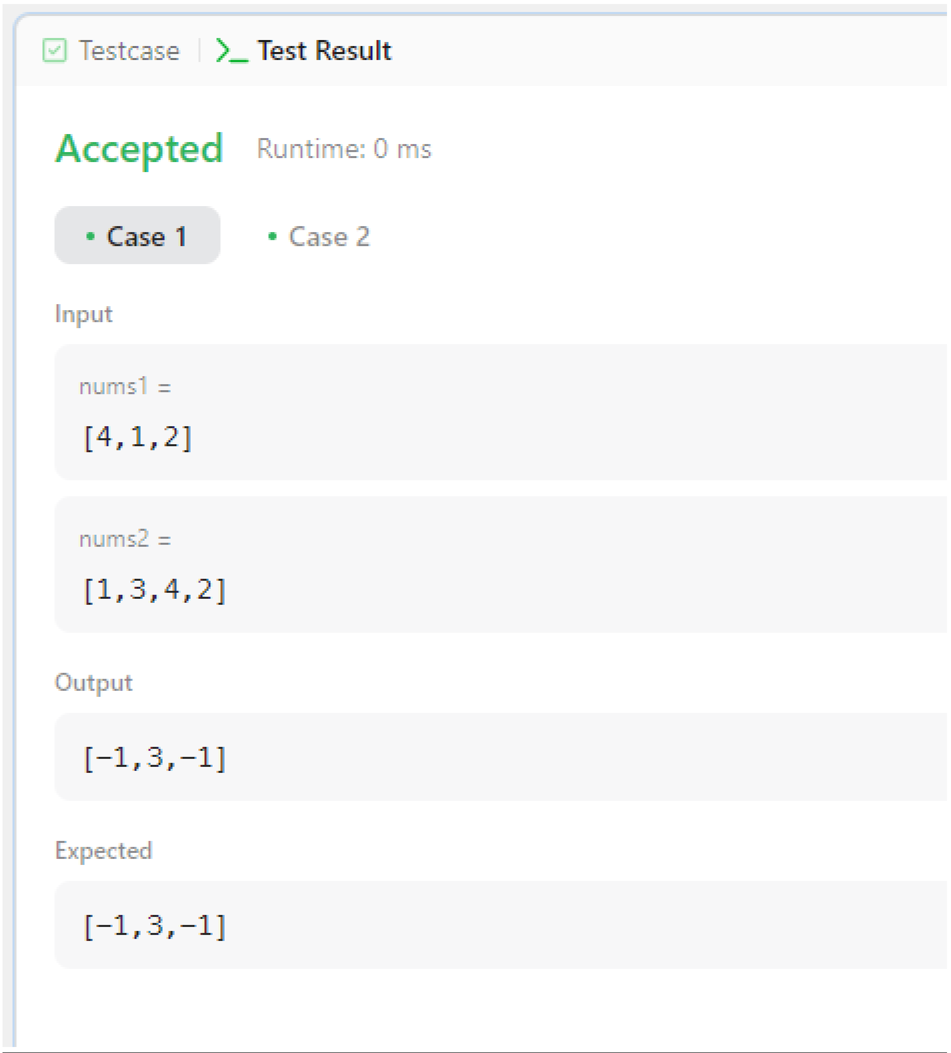
TASK 03: Next Greater Element I

Coding:



```java
class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        HashMap<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();

        for (int num : nums2) {
            while (!stack.isEmpty() && stack.peek() < num) {
                nextGreaterMap.put(stack.pop(), num);
            }
            stack.push(num);
        }

        while (!stack.isEmpty()) {
            nextGreaterMap.put(stack.pop(), -1);
        }

        int[] result = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            result[i] = nextGreaterMap.get(nums1[i]);
        }

        return result;
    }
}
```

OUTPUT:



Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums1 =

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

nums2 =

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

Output

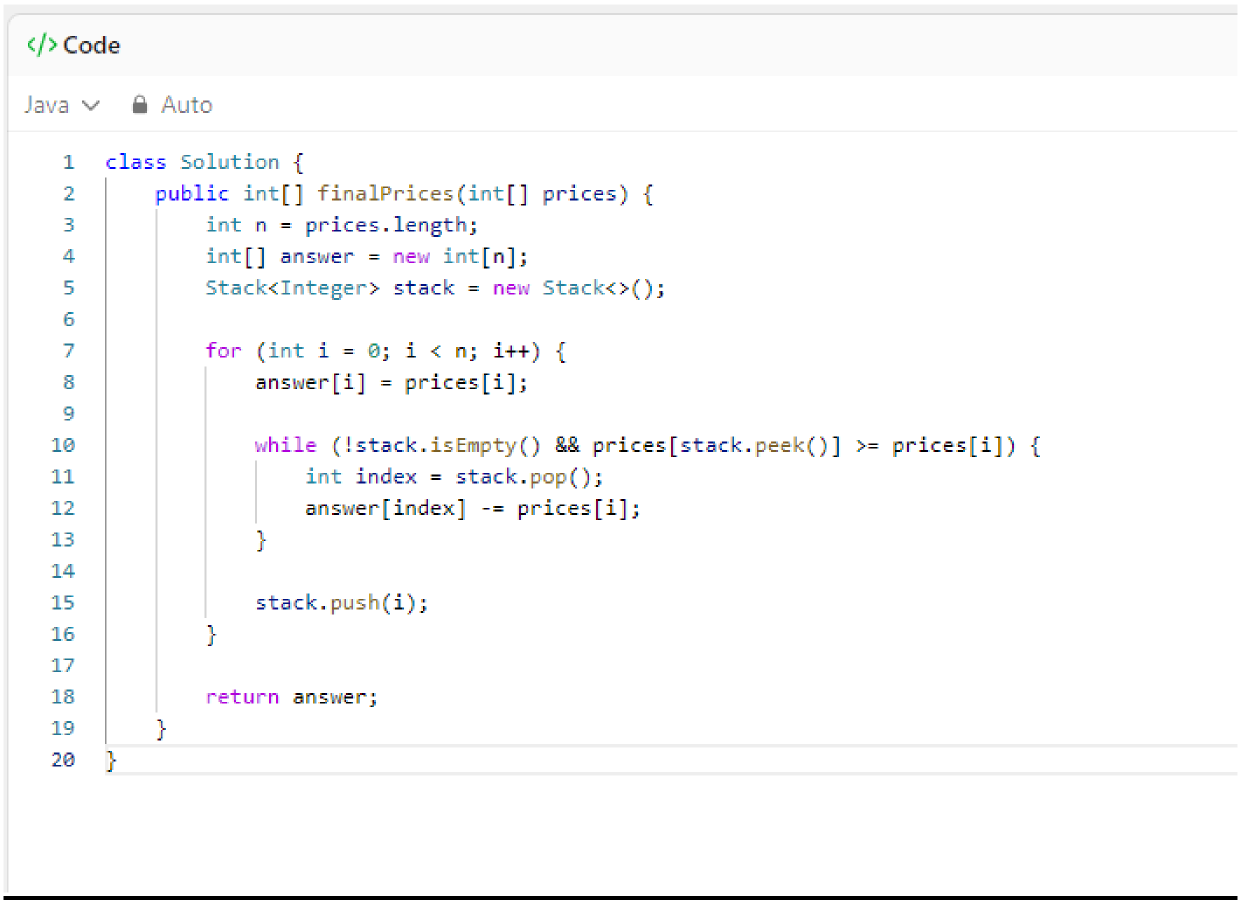
```
[-1,3,-1]
```

Expected

```
[-1,3,-1]
```

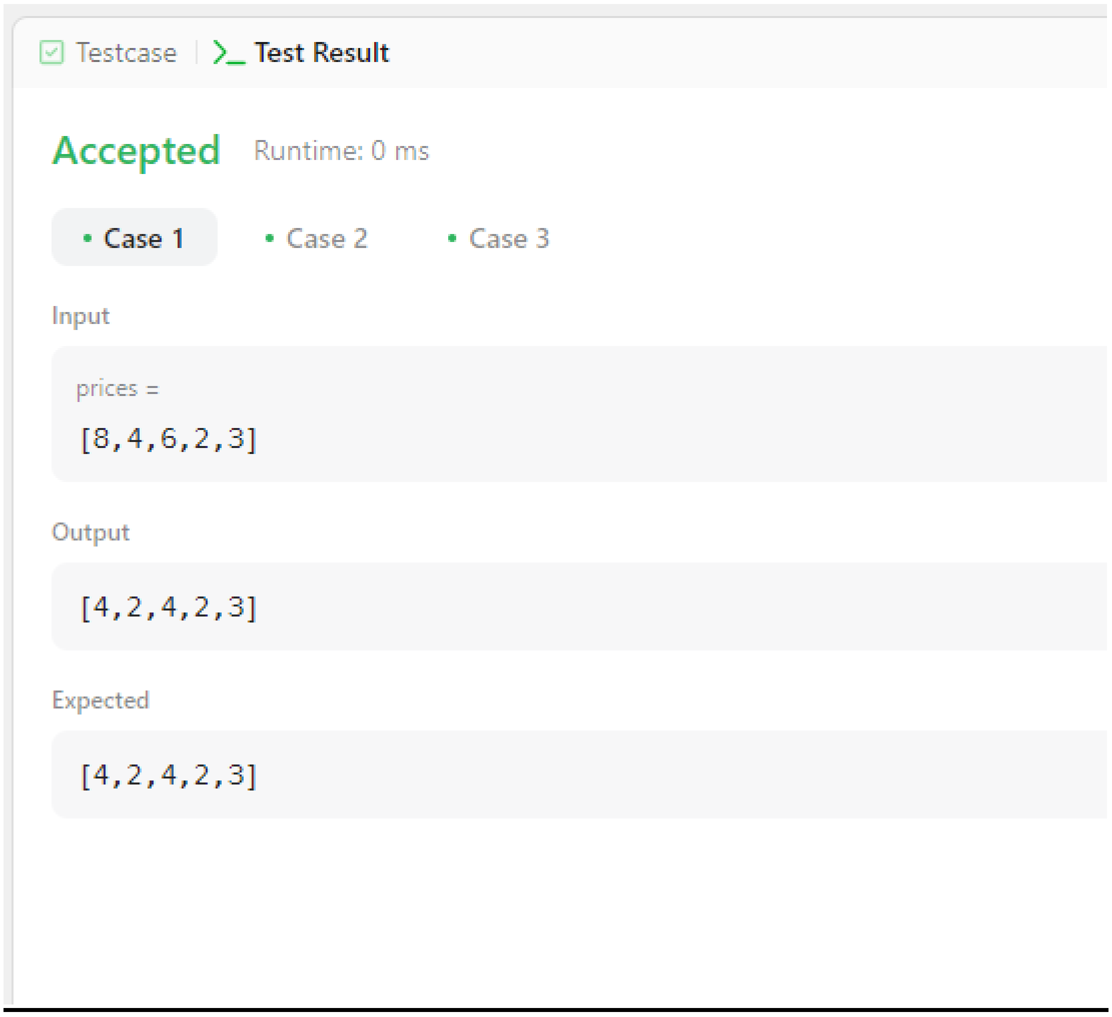
TASK 04: Final Prices With a Special Discount in a Shop

Coding:



```java
class Solution {
    public int[] finalPrices(int[] prices) {
        int n = prices.length;
        int[] answer = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < n; i++) {
            answer[i] = prices[i];

            while (!stack.isEmpty() && prices[stack.peek()] >= prices[i]) {
                int index = stack.pop();
                answer[index] -= prices[i];
            }

            stack.push(i);
        }

        return answer;
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

prices =

```
[8,4,6,2,3]
```

Output

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

Expected

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

THE END