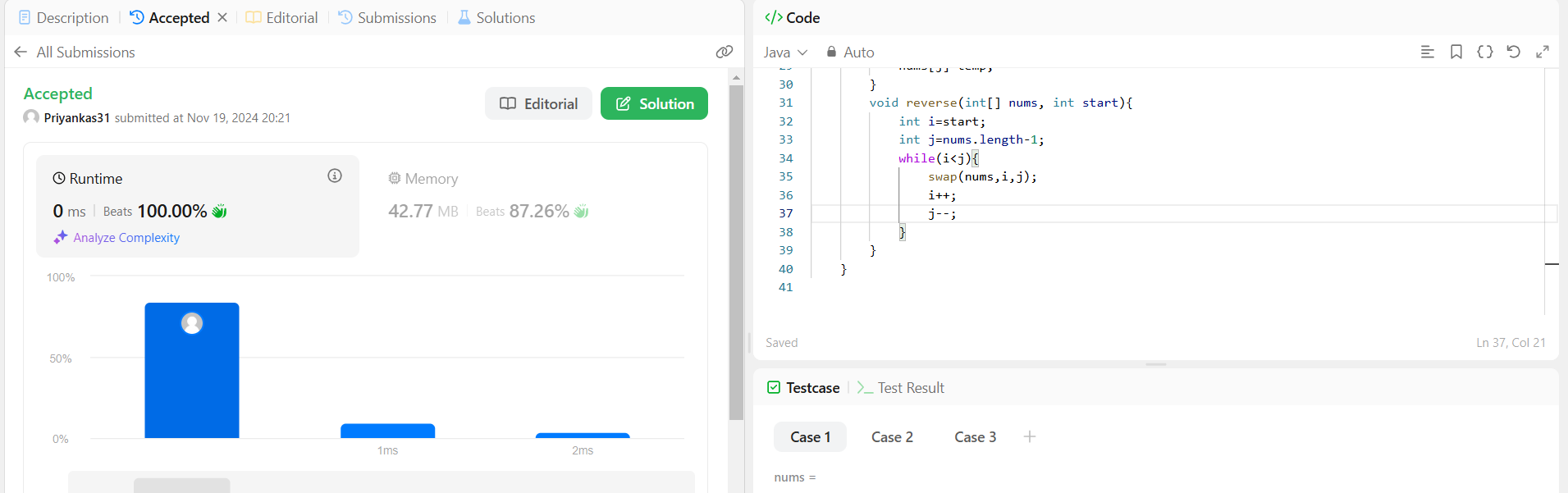
**DSA PRACTICE – 7 – 19/11/2024**

1. NEXT PERMUTATION
2. class Solution {
3. public void nextPermutation(int[] nums) {
4. int ind1=-1;
5. int ind2=-1;
6. for(int i=nums.length-2;i>=0;i--){
7. if(nums[i]<nums[i+1]){
8. ind1=i;
9. break;
10. }
11. }
12. if(ind1==-1){
13. reverse(nums,0);
14. }
15. else{
16. for(int i=nums.length-1;i>=0;i--){
17. if(nums[i]>nums[ind1]){
18. ind2=i;
19. break;
20. }
21. }
22. swap(nums,ind1,ind2);
23. reverse(nums,ind1+1);
24. }
25. }
27. void swap(int[] nums, int i,int j){
28. int temp=nums[i];
29. nums[i]=nums[j];
30. nums[j]=temp;
31. }
32. void reverse(int[] nums, int start){
33. int i=start;
34. int j=nums.length-1;
35. while(i<j){
36. swap(nums,i,j);
37. i++;
38. j--;
39. }
40. }
41. }



2.SPIRAL MATRIX  
class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        int rows = matrix.length;

        int cols = matrix[0].length;

        int x = 0;

        int y = 0;

        int dx = 1;

        int dy = 0;

        List<Integer> res = new ArrayList<>();

        for (int i = 0; i < rows \* cols; i++) {

            res.add(matrix[y][x]);

            matrix[y][x] = -101; // the range of numbers in matrix is from -100 to 100

            if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) || matrix[y+dy][x+dx] == -101) {

                int temp = dx;

                dx = -dy;

                dy = temp;

            }

            x += dx;

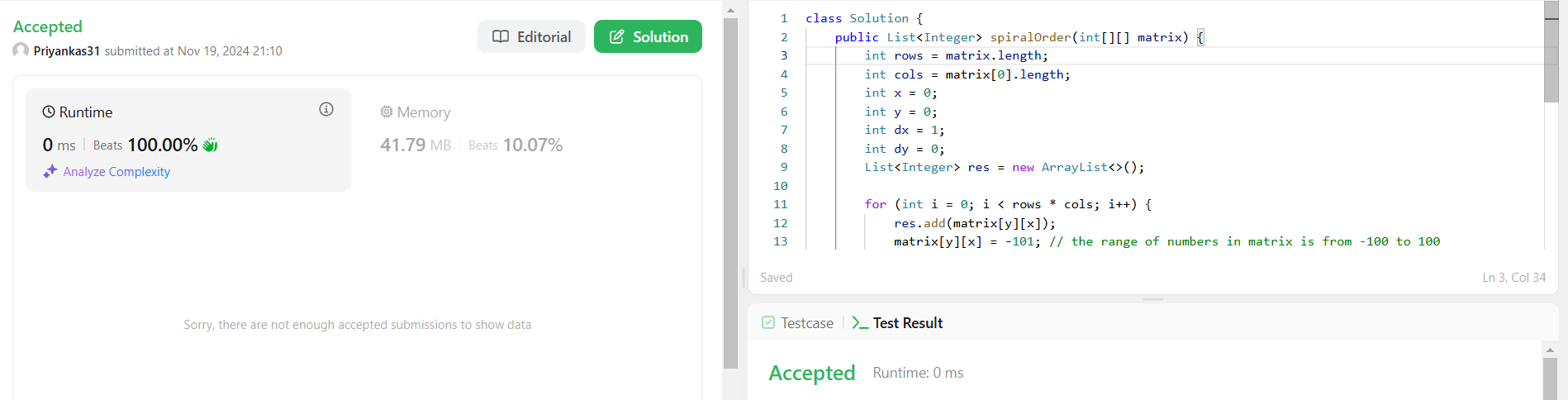
            y += dy;

        }

        return res;

    }

}



**3. LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int left = 0;

        int maxLength = 0;

        HashSet<Character> charSet = new HashSet<>();

        for (int right = 0; right < s.length(); right++) {

            while (charSet.contains(s.charAt(right))) {

                charSet.remove(s.charAt(left));

                left++;

            }

            charSet.add(s.charAt(right));

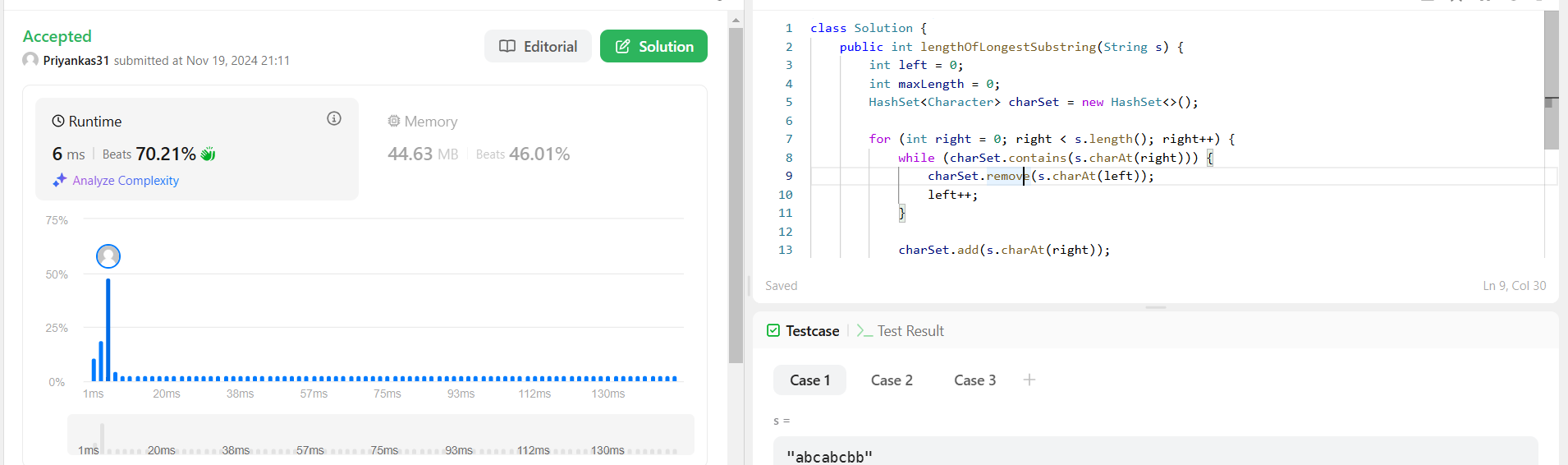
            maxLength = Math.max(maxLength, right - left + 1);

        }

        return maxLength;

    }

}



**4. REMOVE LINKEDLIST ELEMENTS**

class Solution {

    public ListNode removeElements(ListNode head, int val) {

        ListNode temp = new ListNode(0);

        ListNode dummy = temp;

        temp.next=head;

        while(dummy.next!=null){

            if(dummy.next.val==val){

                dummy.next=dummy.next.next;

            }

            else{

            dummy=dummy.next;

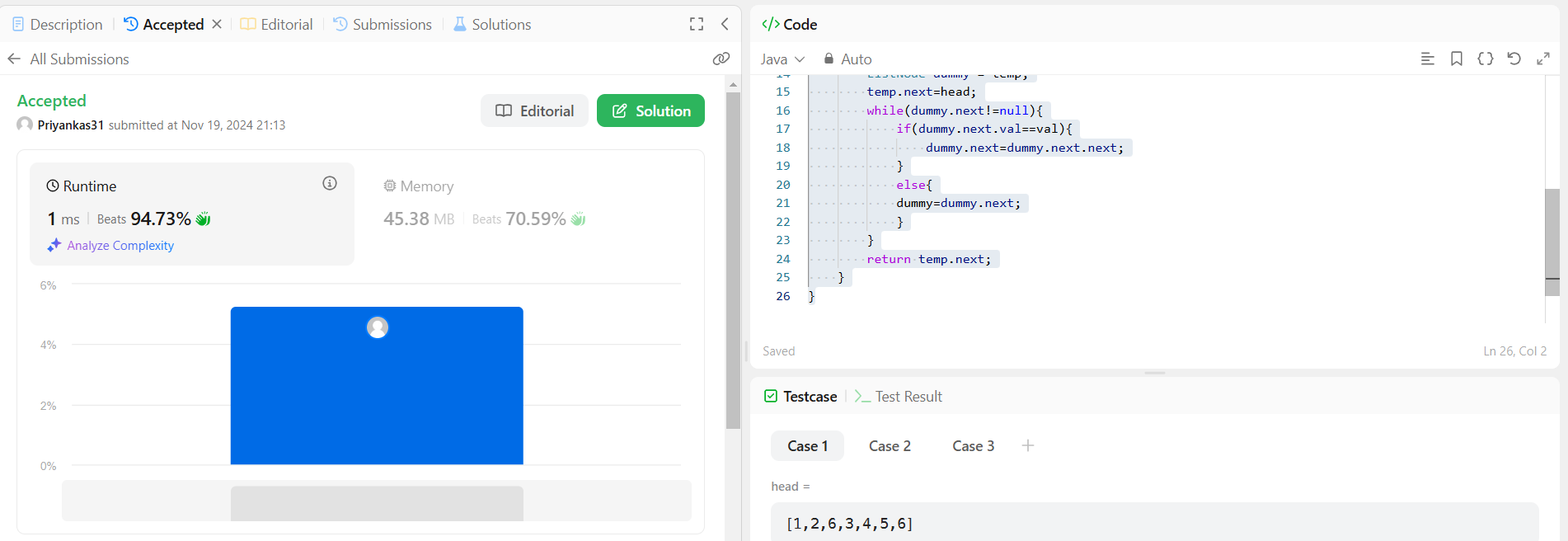
            }

        }

        return temp.next;

    }

}



**5. PALINDROME LINKED LIST**

class Solution {

    public boolean isPalindrome(ListNode head) {

        List<Integer> list = new ArrayList();

        while(head != null) {

            list.add(head.val);

            head = head.next;

        }

        int left = 0;

        int right = list.size()-1;

        while(left < right && list.get(left) == list.get(right)) {

            left++;

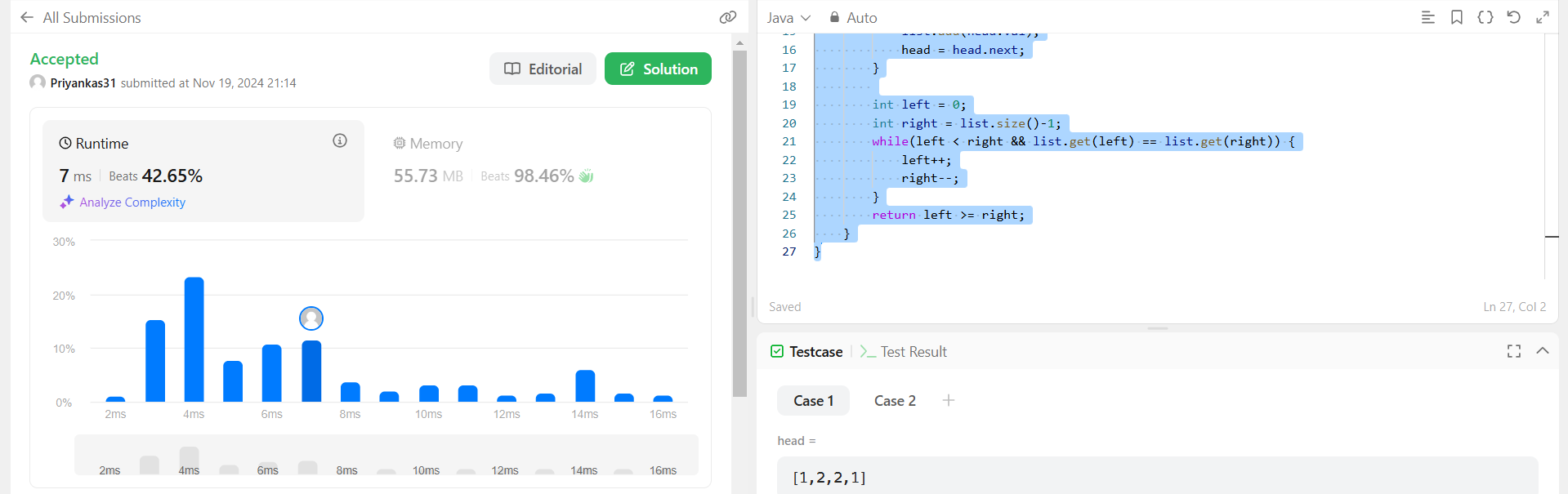
            right--;

        }

        return left >= right;

    }

}



6. Minimum Path Sum  
class Solution {

    public int minPathSum(int[][] grid) {

        int m = grid.length;

        int n = grid[0].length;

        for (int i = 0; i < m; i++) {

            for (int j = 0; j < n; j++) {

                if (i == 0 && j == 0) {

                    continue;

                } else if (i == 0) {

                    grid[i][j] += grid[i][j - 1];

                } else if (j == 0) {

                    grid[i][j] += grid[i - 1][j];

                } else {

                    grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);

                }

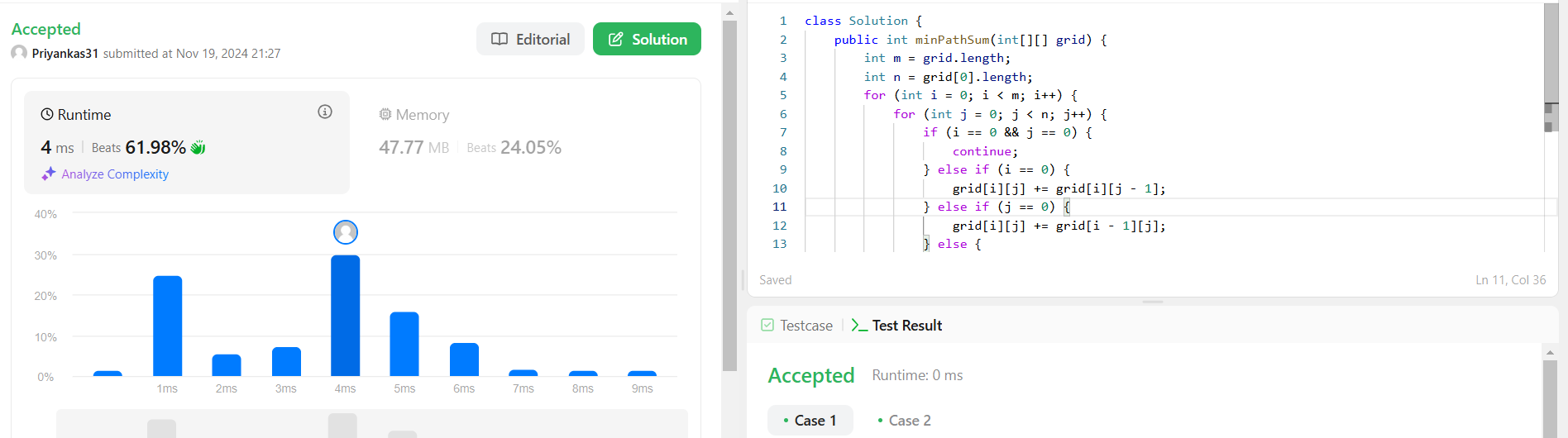
            }

        }

        return grid[m - 1][n - 1];

    }

}



7. Validate Binary Search Tree  
class Solution {

    public boolean isValidBST(TreeNode root) {

        List<Integer> inOrderList = new ArrayList<>();

        inOrderTraversal(root, inOrderList);

        for (int i = 1; i < inOrderList.size(); i++) {

            if (inOrderList.get(i) <= inOrderList.get(i - 1)) {

                return false;

            }

        }

        return true;

    }

    private void inOrderTraversal(TreeNode node, List<Integer> inOrderList) {

        if (node == null) {

            return;

        }

        inOrderTraversal(node.left, inOrderList);

        inOrderList.add(node.val);

        inOrderTraversal(node.right, inOrderList);

    }

}

