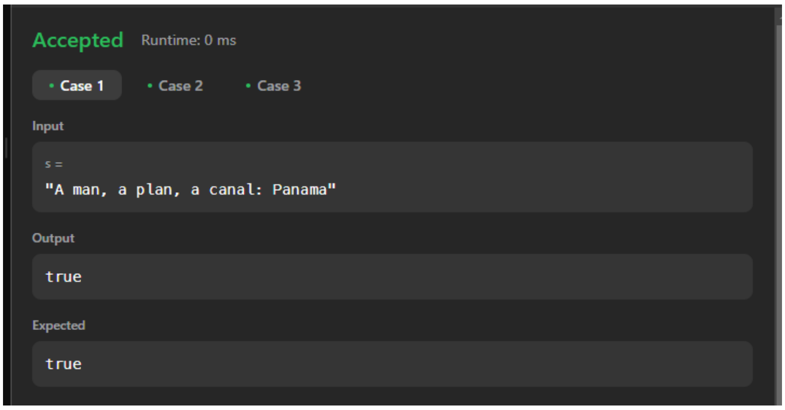
Coding practice Problems(21/11/2024)

Q 1) Valid Palindrome

```
class Solution {
   public boolean isPalindrome(String s) {
      if (s.isEmpty()) {
         return true;
      }
      int start = 0;
      int last = s.length() - 1;
      while(start <= last) {
         char currFirst = s.charAt(start);
         char currLast = s.charAt(last);
         if (!Character.isLetterOrDigit(currFirst )) {
                start++;
         } else if(!Character.isLetterOrDigit(currLast)) {
                last--;
         } else {
                if (Character.toLowerCase(currFirst) != Character.toLowerCase(currLast)) {
                       return false;
                }
                start++;
                last--;
         }
      }
      return true;
   }
}
```

Time Complexity: O(n)

Output:



Input

```
s =
"A man, a plan, a canal: Panama"
```

Output

```
true
```

Expected

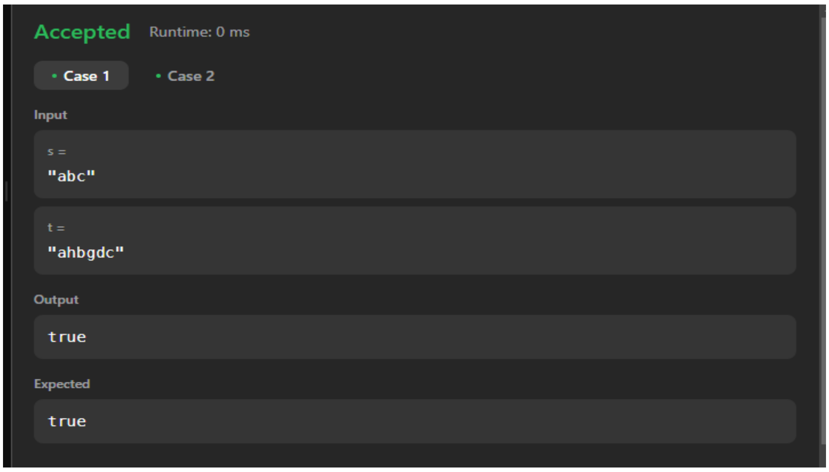
```
true
```

Q 2) Is Subsequence

```
class Solution {
 public boolean isSubsequence(String s, String t) {
  if (s.isEmpty())
   return true;
  int i = 0;
  for (final char c : t.toCharArray())
   if (s.charAt(i) == c && ++i == s.length())
    return true;

  return false;
 }
}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

s =
"abc"

t =
"ahbgdc"

Output

true

Expected

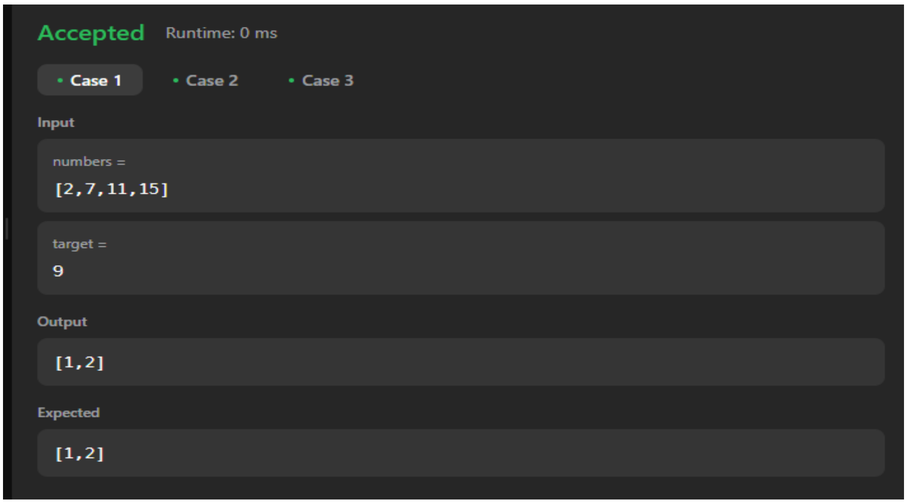
true

Q 3) Two Sum II – Input Array Is Sorted

```
class Solution {

    public int[] twoSum(int[] numbers, int target) {

        int left = 0;

        int right = numbers.length - 1;

        while (left < right) {

            int total = numbers[left] + numbers[right];

            if (total == target) {

                return new int[]{left + 1, right + 1};

            } else if (total > target) {

                right--;

            } else {

                left++;

            }

        }

        return new int[]{-1, -1};

    }

}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

numbers =

```
[2,7,11,15]
```

target =

```
9
```

Output

```
[1,2]
```

Expected

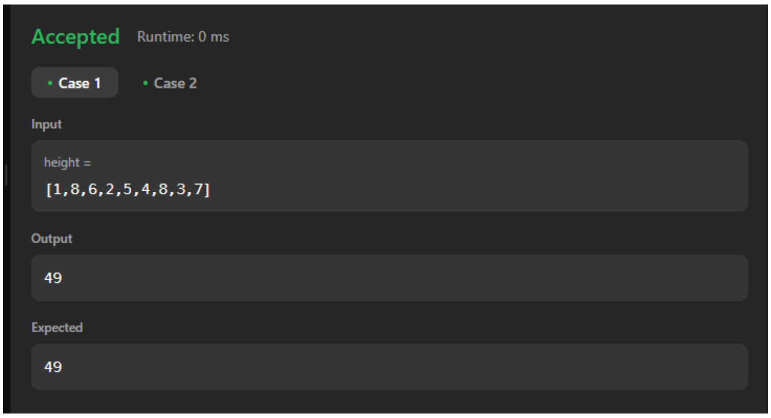
```
[1,2]
```

Q 4) Container With Most Water

```
class Solution {
    public int maxArea(int[] height) {
        int maxArea = 0;
        int left = 0;
        int right = height.length - 1;
        while (left < right) {
            maxArea = Math.max(maxArea, (right - left) * Math.min(height[left], height[right]));
            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }
        return maxArea;
    }
}
```

Time Complexity: O(n)

Output:


Accepted Runtime: 0 ms

Input

```
height =
[1,8,6,2,5,4,8,3,7]
```

Output

```
49
```

Expected

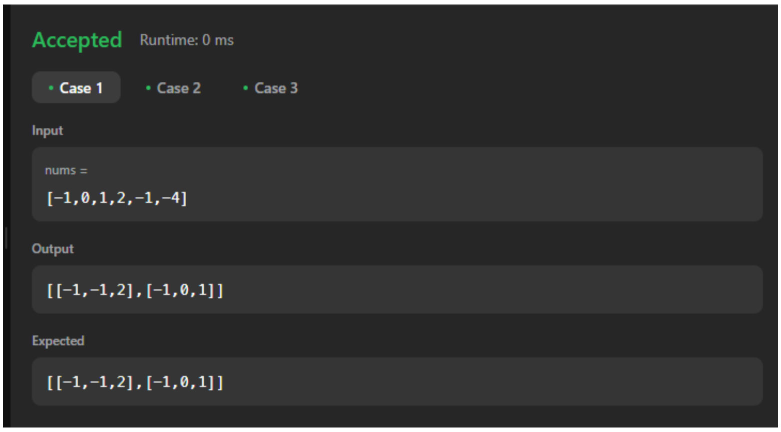
```
49
```

Q 5) 3Sum

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        for (int i = 0; i < nums.length; i++) {
            if (i > 0 && nums[i] == nums[i-1]) {
                continue;
            }
            int j = i + 1;
            int k = nums.length - 1;
            while (j < k) {
                int total = nums[i] + nums[j] + nums[k];
                if (total > 0) {
                    k--;
                } else if (total < 0) {
                    j++;
                } else {
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    j++;
                    while (nums[j] == nums[j-1] && j < k) {
                        j++;
                    }
                }
            }
        }
        return res;
    }
}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

nums =

```
[-1,0,1,2,-1,-4]
```

Output

```
[[-1,-1,2],[-1,0,1]]
```

Expected

```
[[-1,-1,2],[-1,0,1]]
```

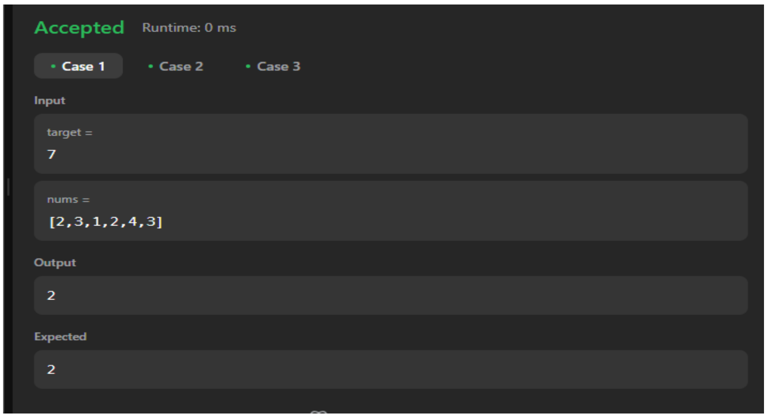
Q 6) Minimum Size Subarray Sum

```
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int minLen = Integer.MAX_VALUE;
        int left = 0;
        int curSum = 0;
        for (int right = 0; right < nums.length; right++) {
            curSum += nums[right];
            while (curSum >= target) {
                if (right - left + 1 < minLen) {
```

```
                minLen = right - left + 1;
            }
            curSum -= nums[left];
            left++;
        }
    }
    return minLen != Integer.MAX_VALUE ? minLen : 0;
  }
}
```

Time Complexity : O(n)

Output :



Accepted Runtime: 0 ms

```
Input
target =
7
nums =
[2,3,1,2,4,3]
Output
2
Expected
2
```

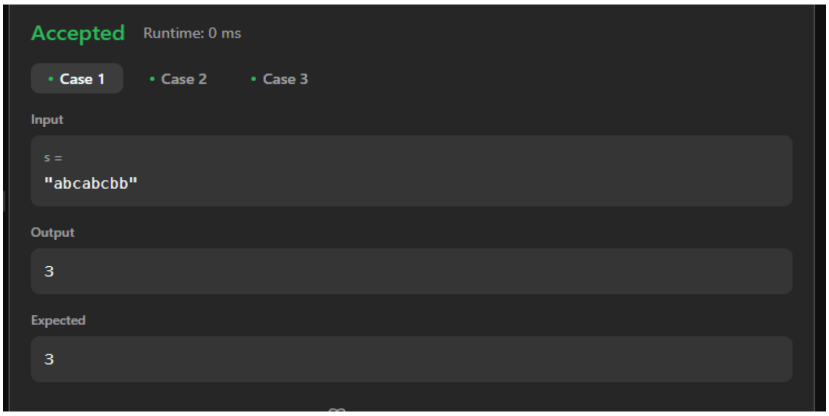
Q 7) Longest Substring Without Repeating Characters

```
class Solution {
  public int lengthOfLongestSubstring(String s) {
    int n = s.length();
    int maxLength = 0;
    int[] charIndex = new int[128];
    Arrays.fill(charIndex, -1);
    int left = 0;
    for (int right = 0; right < n; right++) {
      if (charIndex[s.charAt(right)] >= left) {
```

```
                left = charIndex[s.charAt(right)] + 1;
            }
            charIndex[s.charAt(right)] = right;
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```

Time Complexity: O(n)

Output:



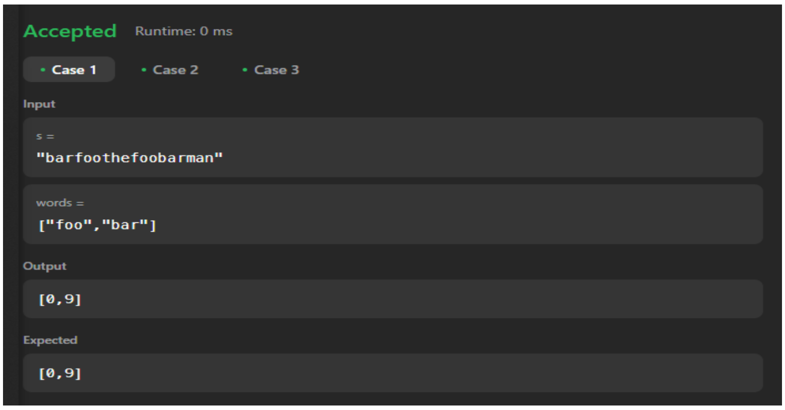
```
Accepted   Runtime: 0 ms
Case 1   Case 2   Case 3
Input
s =
"abcabcbb"
Output
3
Expected
3
```

Q 8) Substring with Concatenation of All Words

```
class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> ans = new ArrayList<>();
        int n = s.length();
        int m = words.length;
        int w = words[0].length();
        HashMap<String,Integer> map = new HashMap<>();
        for(String x : words)
        map.put(x, map.getOrDefault(x,0)+1);
        for(int i=0; i<w; i++){
            HashMap<String,Integer> temp = new HashMap<>();
            int count = 0;
            for(int j=i,k=i; j+w <= n; j=j+w){
                String word = s.substring(j,j+w);
                temp.put(word,temp.getOrDefault(word,0)+1);
                count++;
                if(count==m){
                    if(map.equals(temp)){
                        ans.add(k);
                    }
                    String remove = s.substring(k,k+w);
                    temp.computeIfPresent(remove, (a, b) -> (b > 1) ? b - 1 : null);
                    count--;
                    k=k+w;
                }
            }
        }
        return ans;
    }
}
```

Time Complexity: O(n.m)

Output:



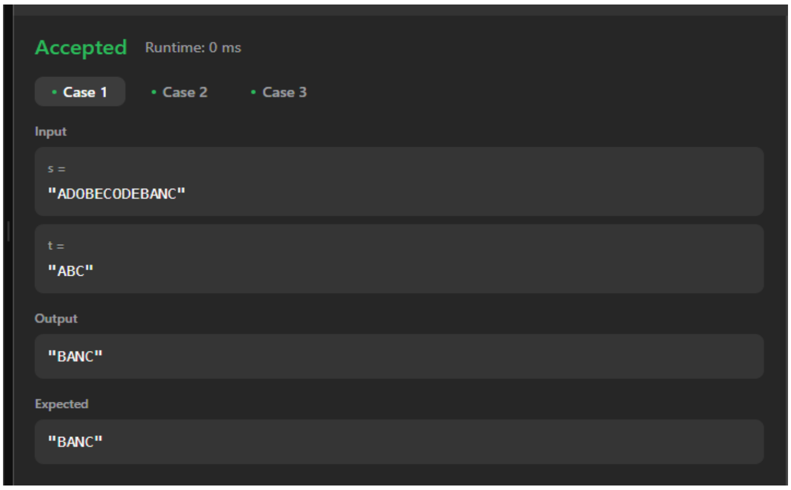
```
Input
s =
"barfoothefoobarman"
words =
["foo","bar"]
Output
[0,9]
Expected
[0,9]
```

Q 9) Minimum Window Substring

```
class Solution {
    public String minWindow(String s, String t) {
        if (s.length() < t.length()) {
            return "";
        }
        Map<Character, Integer> charCount = new HashMap<>();
        for (char ch : t.toCharArray()) {
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
        }
        int targetCharsRemaining = t.length();
        int[] minWindow = {0, Integer.MAX_VALUE};
        int startIndex = 0;
        for (int endIndex = 0; endIndex < s.length(); endIndex++) {
            char ch = s.charAt(endIndex);
            if (charCount.containsKey(ch) && charCount.get(ch) > 0) {
                targetCharsRemaining--;
            }
            charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);
            if (targetCharsRemaining == 0) {
                while (true) {
                    char charAtStart = s.charAt(startIndex);
                    if (charCount.containsKey(charAtStart) && charCount.get(charAtStart) == 0) {
                        break;
                    }
                    charCount.put(charAtStart, charCount.getOrDefault(charAtStart, 0) + 1);
                    startIndex++;
                }
                if (endIndex - startIndex < minWindow[1] - minWindow[0]) {
                    minWindow[0] = startIndex;
                    minWindow[1] = endIndex;
                }
                charCount.put(s.charAt(startIndex), charCount.getOrDefault(s.charAt(startIndex), 0) + 1);
                targetCharsRemaining++;
                startIndex++;
            }
        }
        return minWindow[1] >= s.length() ? "" : s.substring(minWindow[0], minWindow[1] + 1);
    }
}
```

Time Complexity: O(s+t)

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =

"ADOBECODEBANC"

t =

"ABC"

Output

"BANC"

Expected

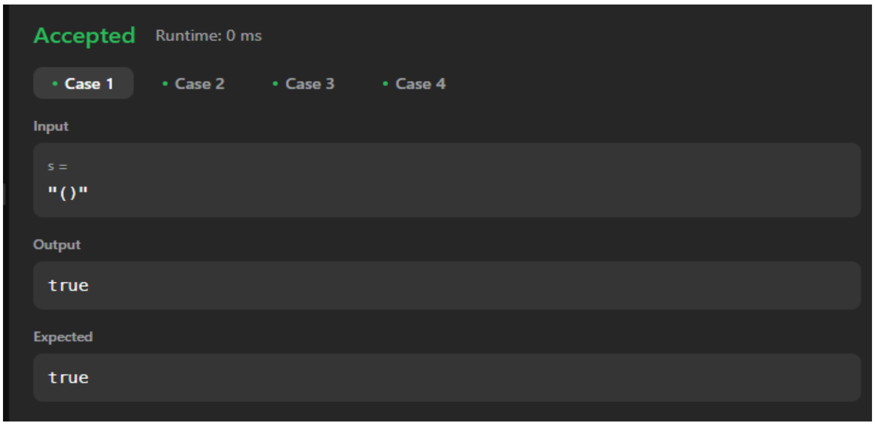
"BANC"

Q 10) Valid Parentheses

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(')
                stack.push(')');
            else if (c == '{')
                stack.push('}');
            else if (c == '[')
                stack.push(']');
            else if (stack.isEmpty() || stack.pop() != c)
                return false;
        }
        return stack.isEmpty();
    }
}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

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

Output

```
true
```

Expected

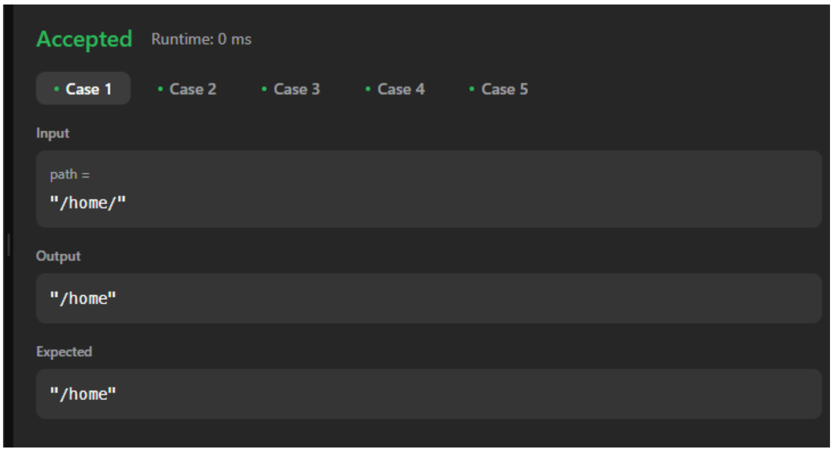
```
true
```

Q 11) Simplify Path

```
class Solution {
    public String simplifyPath(String path) {
        Stack<String> stack = new Stack<>();
        String[] directories = path.split("/");
        for (String dir : directories) {
            if (dir.equals(".") || dir.isEmpty()) {
                continue;
            } else if (dir.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop();
                }
            } else {
                stack.push(dir);
            }
        }
        return "/" + String.join("/", stack);
    }
}
```

Time Complexity: O(n)

Output:


Accepted Runtime: 0 ms

Input

path =
"/home/"

Output

"/home"

Expected

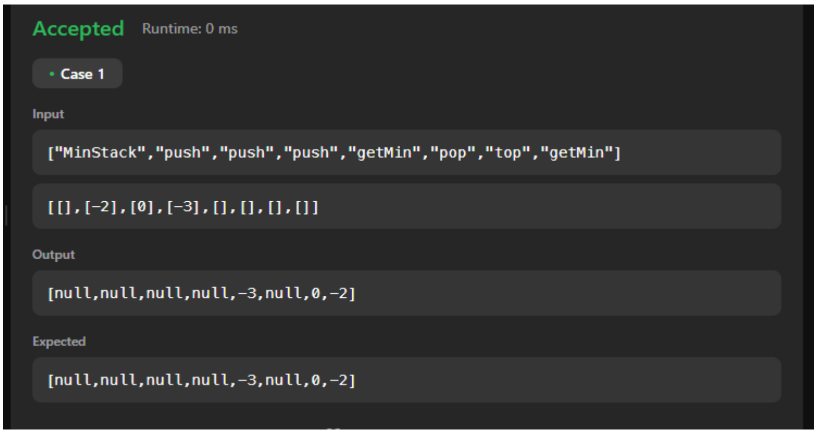
"/home"

Q 12) Min Stack

```
class MinStack {

    private List<int[]> st;

    public MinStack() {

        st = new ArrayList<>();

    }

    public void push(int val) {

        int[] top = st.isEmpty() ? new int[]{val, val} : st.get(st.size() - 1);

        int min_val = top[1];

        if (min_val > val) {

            min_val = val;

        }

        st.add(new int[]{val, min_val});

    }

    public void pop() {

        st.remove(st.size() - 1);

    }

    public int top() {

        return st.isEmpty() ? -1 : st.get(st.size() - 1)[0];

    }
    public int getMin() {
        return st.isEmpty() ? -1 : st.get(st.size() - 1)[1];
    }
}
```

Time Complexity: O(n)

Output:



Input

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

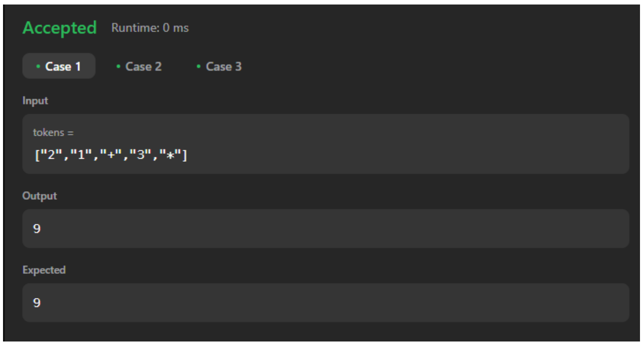
```
[null,null,null,null,-3,null,0,-2]
```

Q 13) Evaluate Reverse Polish Notation

```
class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();
        for (String c : tokens) {
            if (c.equals("+")) {
                stack.push(stack.pop() + stack.pop());
            } else if (c.equals("-")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first - second);
            } else if (c.equals("*")) {
                stack.push(stack.pop() * stack.pop());
            } else if (c.equals("/")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first / second);
            } else {
                stack.push(Integer.parseInt(c));
            }
        }
        return stack.peek();
    }
}
```

Time Complexity: O(n)

Output:



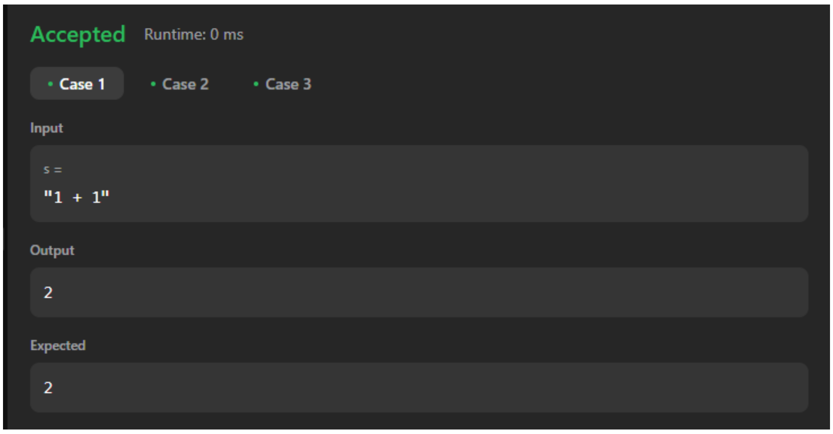
```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
tokens =
["2","1","+","3","*"]

Output
9

Expected
9
```

Q 14) Basic Calculator

```
class Solution {
    public int calculate(String s) {
        int number = 0;
        int signValue = 1;
        int result = 0;
        Stack<Integer> operationsStack = new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (Character.isDigit(c)) {
                number = number * 10 + (c - '0');
            } else if (c == '+' || c == '-') {
                result += number * signValue;
                signValue = (c == '-') ? -1 : 1;
                number = 0;
            } else if (c == '(') {
                operationsStack.push(result);
                operationsStack.push(signValue);
                result = 0;
                signValue = 1;
            } else if (c == ')') {
                result += signValue * number;
                result *= operationsStack.pop();
                result += operationsStack.pop();
                number = 0;
            }
        }
        return result + number * signValue;
    }
}
```

Time Complexity: O(n)

Output:



Input

```
s =
"1 + 1"
```

Output

```
2
```

Expected

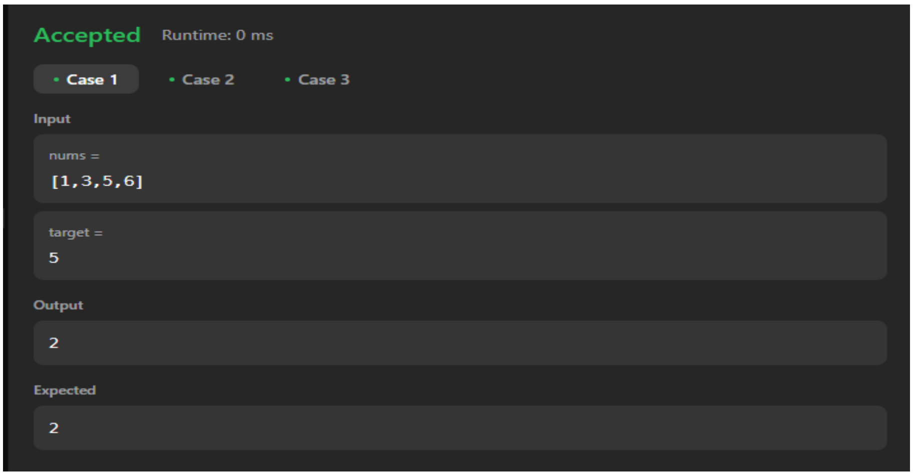
```
2
```

Q 15) Search Insert Position

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] > target) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

nums =
[1,3,5,6]

target =
5

Output

2

Expected

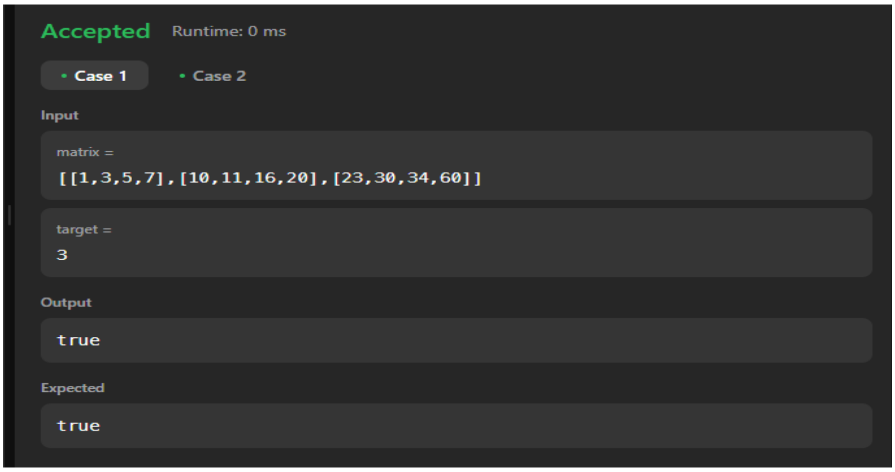
2

Q 16) Search a 2D Matrix

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int m = matrix.length;
        int n = matrix[0].length;
        int i=0;
        int j=n-1;
        while(i<m && j>=0){
            if(matrix[i][j]==target) return true;
            if(matrix[i][j]>target){
                j--;
            }
            else{
                i++;
            }
        }
        return false;
    }
}
```

Time Complexity: O(n)

Output:



Input

matrix =

```
[[1,3,5,7],[10,11,16,20],[23,30,34,60]]
```

target =

```
3
```

Output

```
true
```

Expected

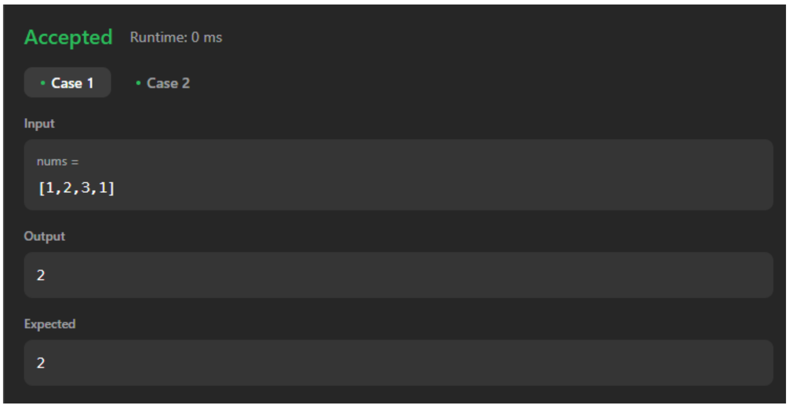
```
true
```

Q 17) Find Peak Element

```
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0;
        int right = nums.length - 1;

        while (left < right) {
            int mid = (left + right) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
}
```

Time Complexity: O(n)

Output:



Input

```
nums =
[1,2,3,1]
```

Output

```
2
```

Expected

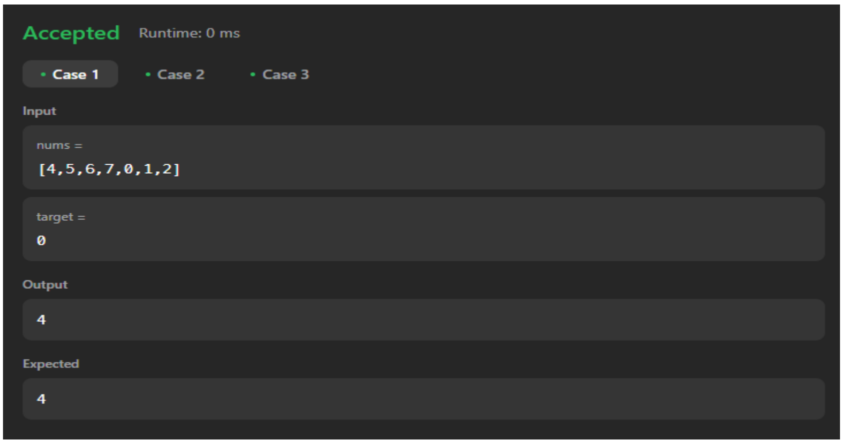
```
2
```

Q 18) Search in Rotated Sorted

```
class Solution {
    public int search(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

        while (left <= right) {
            int mid = (left + right) / 2;

            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] >= nums[left]) {
                if (nums[left] <= target && target <= nums[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (nums[mid] <= target && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }
        return -1;
    }
}
```

Time Complexity:O(logn)

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =

```
[4,5,6,7,0,1,2]
```

target =

```
0
```

Output

```
4
```

Expected

```
4
```

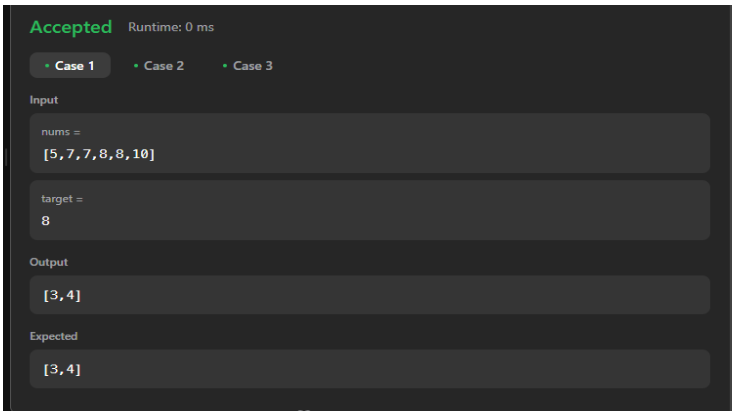
Q 19) Find First and Last Position of Element in Sorted Array

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};
        int left = binarySearch(nums, target, true);
        int right = binarySearch(nums, target, false);
        result[0] = left;
        result[1] = right;
        return result;
    }
    private int binarySearch(int[] nums, int target, boolean isSearchingLeft) {
```

```
    int left = 0;
    int right = nums.length - 1;
    int idx = -1;
    while (left <= right) {
      int mid = left + (right - left) / 2;
      if (nums[mid] > target) {
        right = mid - 1;
      } else if (nums[mid] < target) {
        left = mid + 1;
      } else {
        idx = mid;
        if (isSearchingLeft) {
          right = mid - 1;
        } else {
          left = mid + 1;
        }
      }
    }
    return idx;
  }
}
```

Time Complexity: O(log n)

Output:



Accepted Runtime: 0 ms

Input

nums =

[5,7,7,8,8,10]

target =

8

Output

[3,4]

Expected

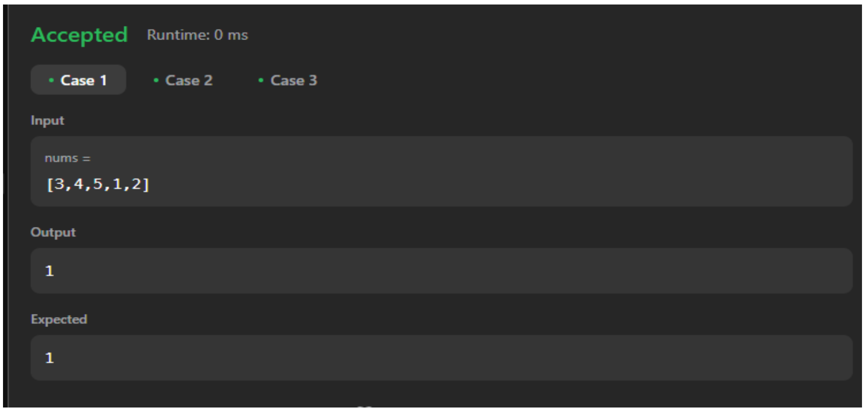
[3,4]

Q 20) Find Minimum in Rotated Sorted Array

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0;
        int right = nums.length-1;
        while(nums[left] > nums[right]){
            int mid = left + (right - left)/2;
            if(nums[mid+1] < nums[mid]){
                return nums[mid+1];
            }
            if(nums[mid-1] > nums[mid]){
                return nums[mid];
            }
            if(nums[mid] < nums[right]){
                right = mid - 1;
            }
            if(nums[mid] > nums[left]){
                left = mid + 1;
            }
        }
        return nums[left];
    }
}
```

Time Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

```
nums =
[3,4,5,1,2]
```

Output

```
1
```

Expected

```
1
```