DSA Practice Question Set - 9

1)Valid Palindrome :

A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a palindrome, or* false *otherwise*.

Example 1:

Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.

Example 3:

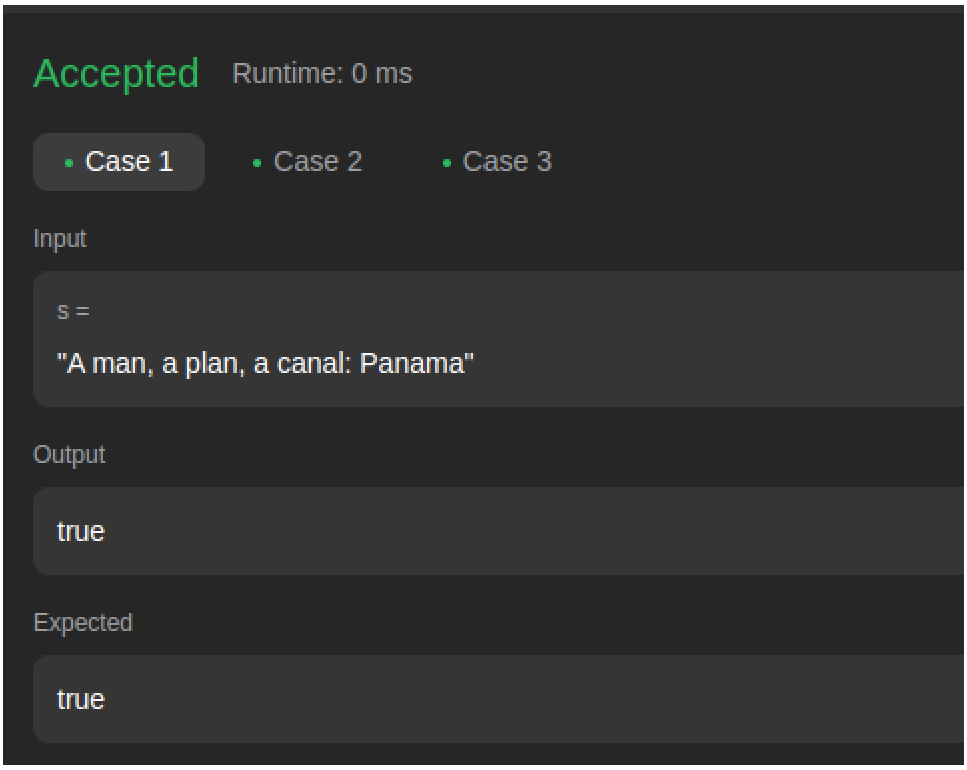
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.

Program :

```
class Solution {
  public boolean isPalindrome(String s) {
     List<Character> input  =  new ArrayList<>();
```

```
        for(char c : s.toLowerCase().toCharArray()){
            if (Character.isLetterOrDigit(c)){
                input.add(c);
            }
        }
        int left = 0;
        int right = input.size()-1;
        if(right<=0){
            return true;
        }

        while(left<right){
            if(input.get(left)!=input.get(right)){
                return false;
            }
            left++;
            right--;
        }
        return true;

    }
}
```

Output :



Accepted Runtime: 0 ms

Input

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

Output

```
true
```

Expected

```
true
```

i

Time Complexity : O(n)
Space Complexity : O(1)

2) Is Subsequence

Given two strings s and t, return true *if* s *is a subsequence of* t*, or* false *otherwise*.

A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:

Input: s = "abc", t = "ahbgdc"
Output: true

Example 2:

Input: s = "axc", t = "ahbgdc"
Output: false

Constraints:

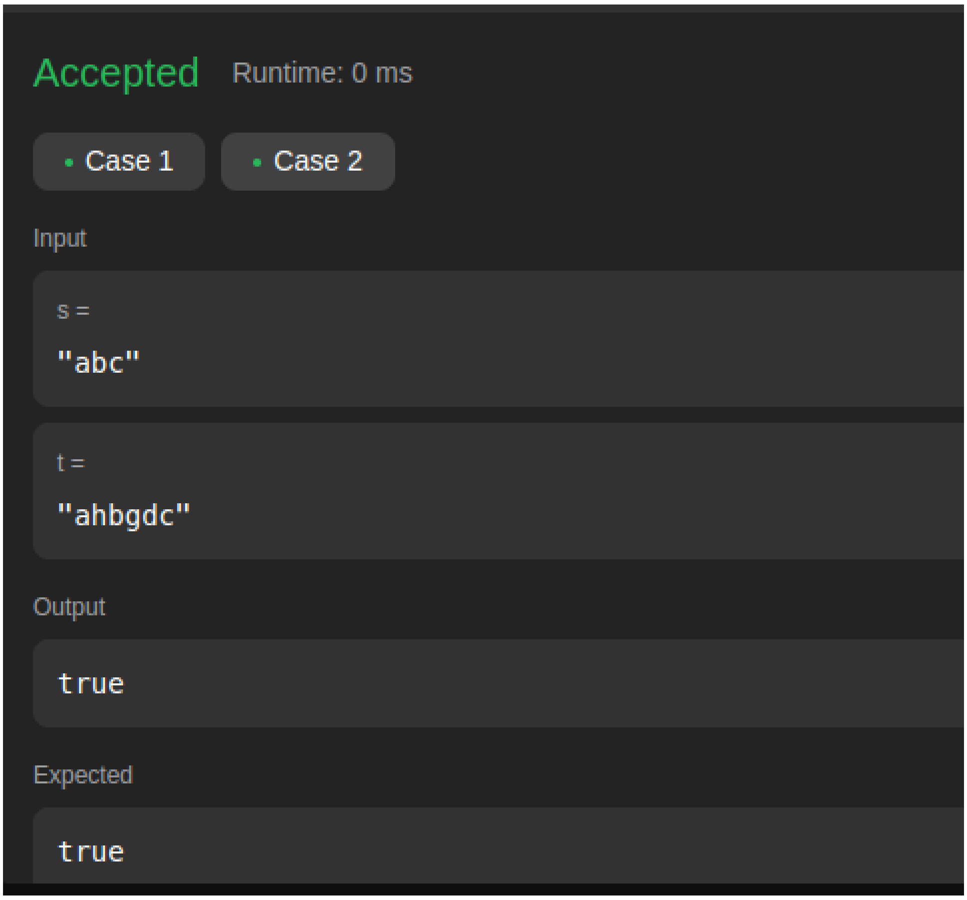
- **0 <= s.length <= 100**
- **0 <= t.length <= 104**
- **s and t consist only of lowercase English letters.**

Follow up: Suppose there are lots of incoming s, say s1, s2, ..., sk where k >= 109, and you want to check one by one to see if t has its subsequence. In this scenario, how would you change your code?

Program :

```
class Solution {
  public boolean isSubsequence(String s, String t) {
      int i = 0;
      int j = 0;

      while(i<s.length() && j<t.length()){
         if(t.charAt(j)== s.charAt(i) ){
            i++;
         }
         j++;
      }

        return i == s.length();
}
}
```

Output :



```
Input
s =
"abc"
t =
"ahbgdc"
Output
true
Expected
true
```

Time Complexity : O(n)
Space Complexity : O(1)

3)TwoSum II

Given a 1-indexed array of integers numbers that is already *sorted in non-decreasing order*, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.

Return *the indices of the two numbers,* index1 *and* index2*, added by one as an integer array* [index1, index2] *of length 2.*

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9
Output: [1,2]

Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

Example 2:

Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].

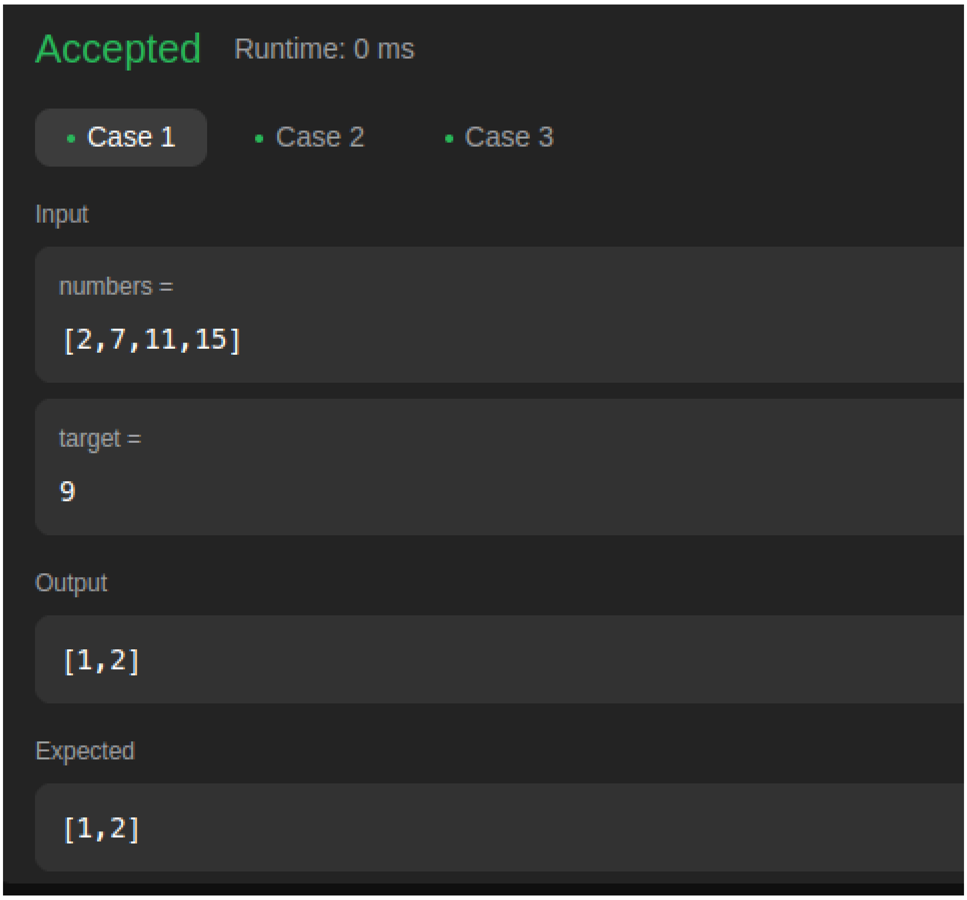
Example 3:

Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Program :

```
class Solution {
  public int[] twoSum(int[] numbers, int target) {
    HashMap<Integer,Integer> map = new HashMap<>();

    for(int i = 0;i<numbers.length;i++){
      int complement = target - numbers[i];

      if(map.containsKey(complement)){
        return new int[] {map.get(complement),i+1};
      }
      map.put(numbers[i],i+1);

    }
    return new int[numbers.length];

  }
}
```

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]
```

Time Complexity : O(n)
Space Complexity : O(1)

4)Container with Most Water :

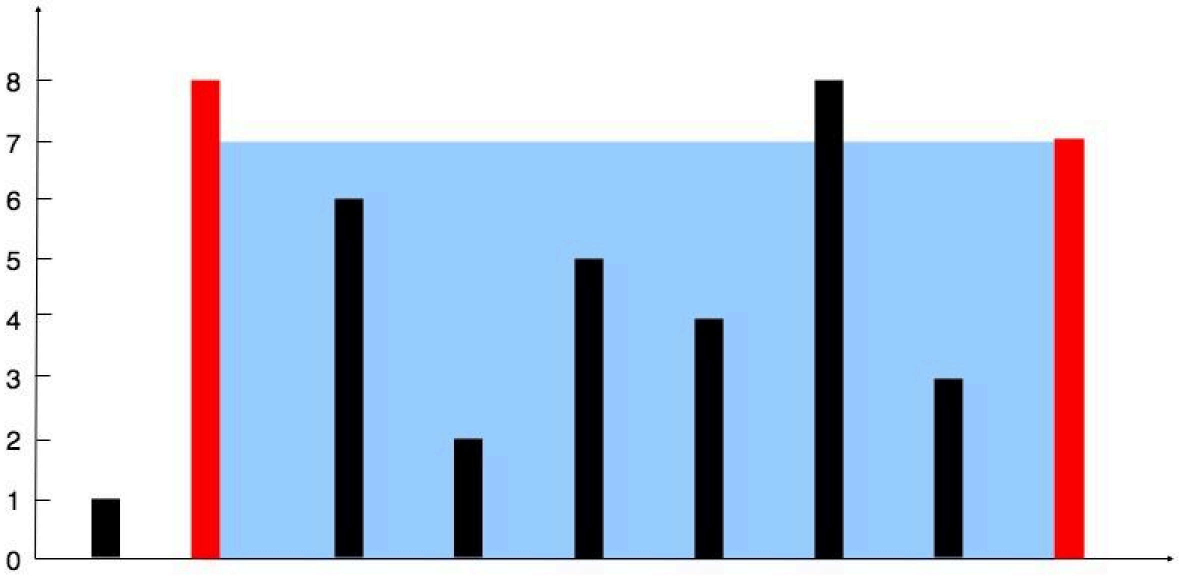
You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

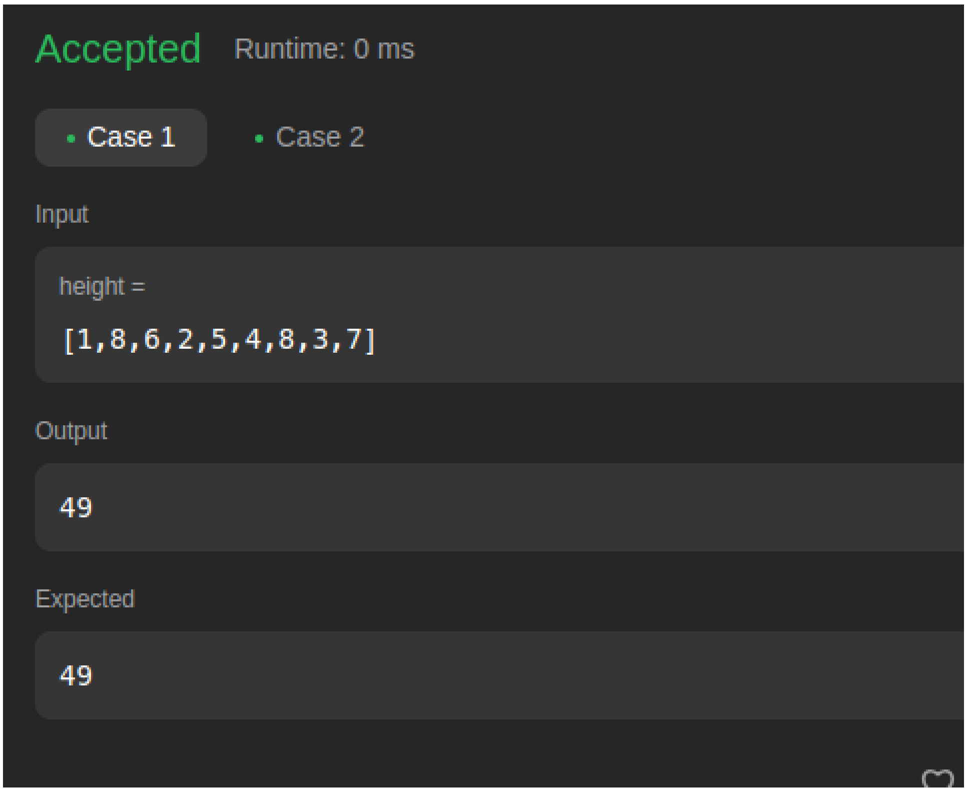
Input: height = [1,1]
Output: 1

Program :

```
class Solution {
  public int maxArea(int[] height) {
    int left = 0;
    int right = height.length-1;
    int area = 0;

    while (left<right){
      int aea  = Math.min(height[left],height[right])*(right-left);
      area = Math.max(area,aea);
      if(height[left]<height[right]){
        left+=1;
```

```
            }else{
                right-=1;
            }

        }
        return area;

    }
}
```

Ouput :



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

height =

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

Output

```
49
```

Expected

```
49
```

Time Complexity : O(n)
Space Complexity : O(1)

5)3Sum :

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.

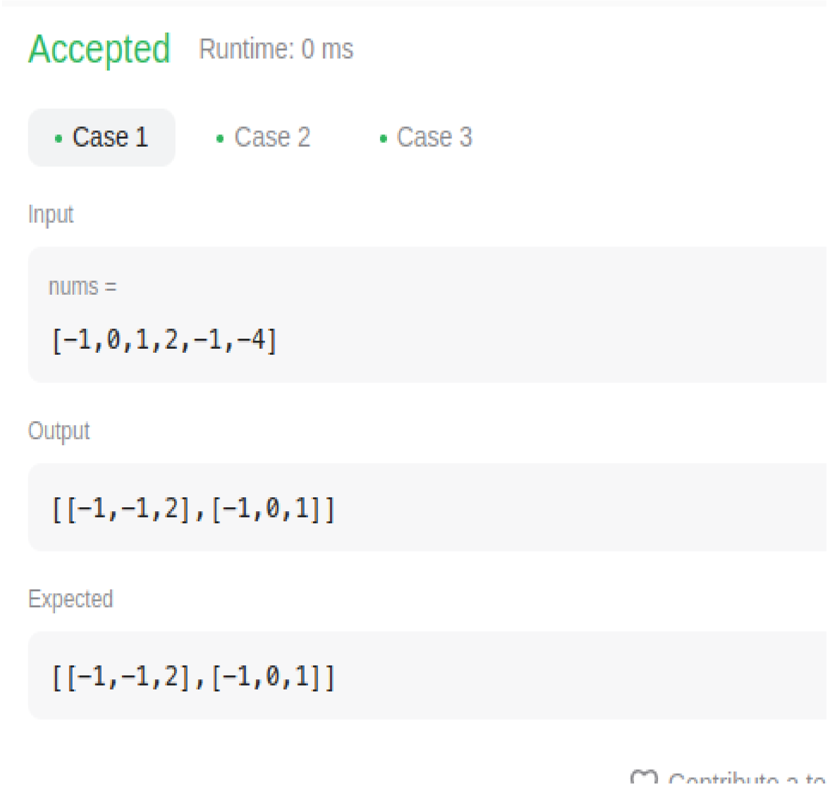
Example 3:

Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.

Program :

```
import java.util.*;

class Solution {
  public List<List<Integer>> threeSum(int[] nums) {
     List<List<Integer>> three = new ArrayList<>();
     Arrays.sort(nums);

     for (int i = 0; i < nums.length - 2; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }

            int j = i + 1;
            int k = nums.length - 1;

            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];

                if (sum == 0) {
                    three.add(Arrays.asList(nums[i], nums[j], nums[k]));

                    while (j < k && nums[j] == nums[j + 1]) {
                        j++;
                    }
                    while (j < k && nums[k] == nums[k - 1]) {
                        k--;
                    }

                    j++;
                    k--;
                } else if (sum < 0) {
                    j++;
                } else {
                    k--;
                }
            }
        }

        return three;
    }
}
```

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]]
```

6)Minimize Subarray sum length

Given an array of positive integers nums and a positive integer target, return *the minimal length of a*

subarray

***whose sum is greater than or equal to* target. If there is no such subarray, return 0 instead.**

Example 1:

Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.

Example 2:

Input: target = 4, nums = [1,4,4]
Output: 1

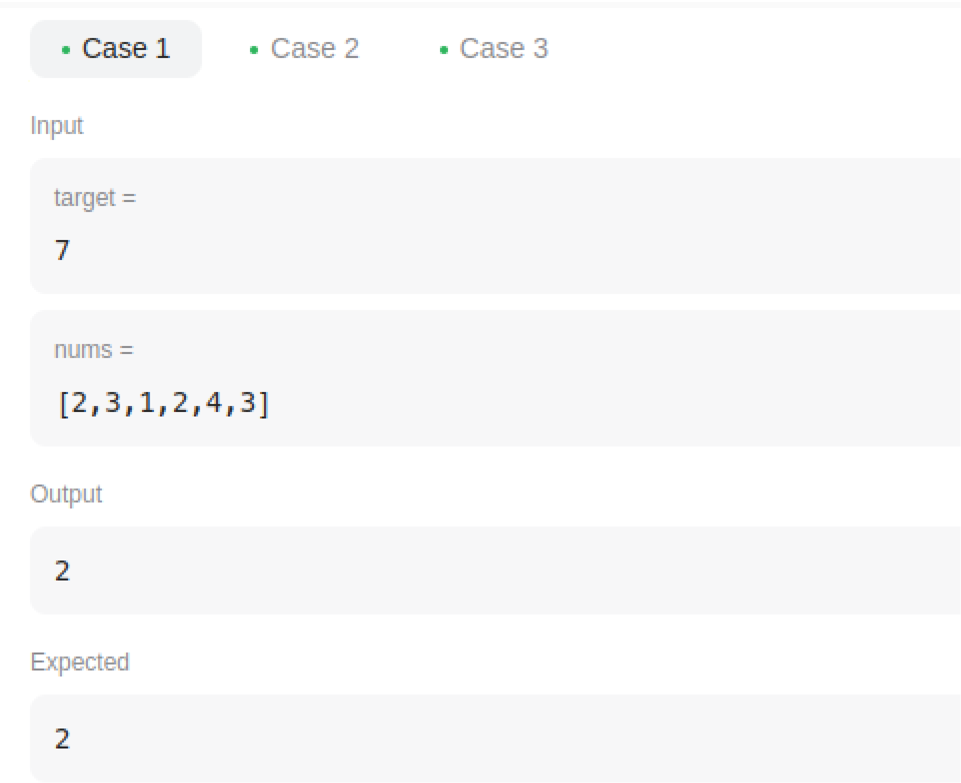
Example 3:

Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0

Program :

```
class Solution {
  public int minSubArrayLen(int target, int[] nums) {
     int left=0,right=0,sum =0;
     int ans = Integer.MAX_VALUE;

     for(right=0;right<nums.length;right++){
        sum +=nums[right];
        while(sum>=target){
           ans=Math.min(ans,right-left+1);
           sum -=nums[left++];
        }

     }
     return ans == Integer.MAX_VALUE ? 0:ans;
  }
}
```

Output :



Input

target =

```
7
```

nums =

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

Output

```
2
```

Expected

```
2
```

Time Complexity : O(n)
Space Complexity : O(1)

7)Longest Substring without repeating characters

Given a string s, find the length of the longest

substring
without repeating characters.

Example 1:

Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"

Output: 3

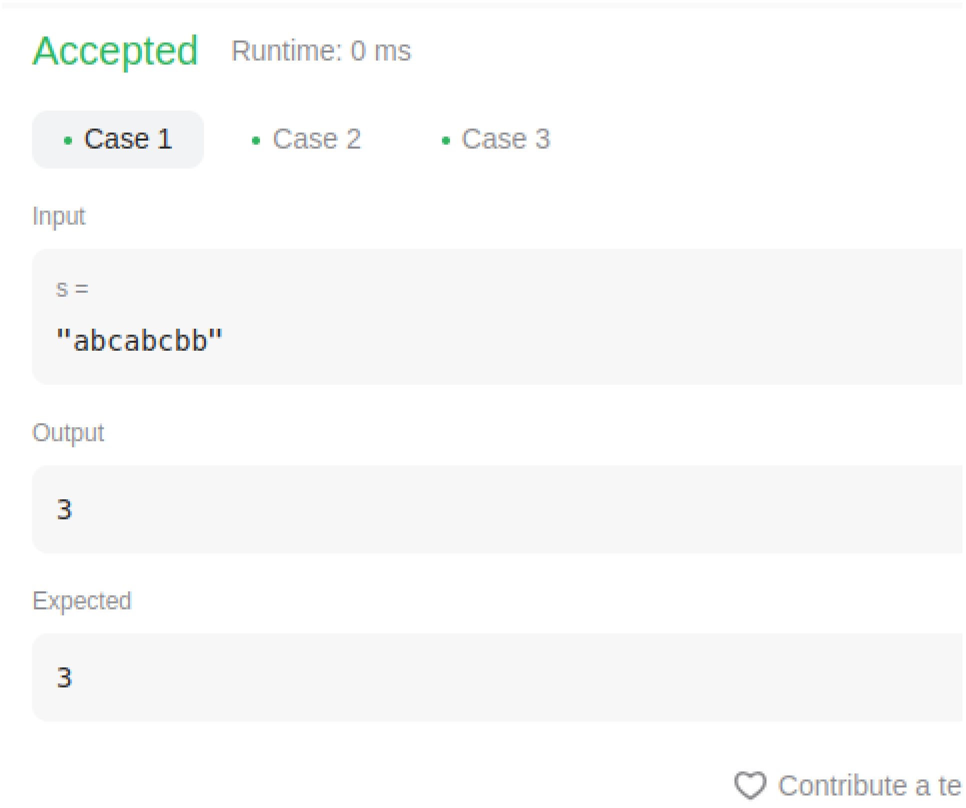
Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Program :

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int a = 0;
        int b = 0;
        int max = 0;

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

        while(b<s.length()){
            if(!hash_set.contains(s.charAt(b))){
                hash_set.add(s.charAt(b));
                b++;
                max = Math.max(hash_set.size(),max);
            }
            else{
                hash_set.remove(s.charAt(a));
                a++;
            }

        }
        return max;

    }
}
```

Output :



Accepted Runtime: 0 ms

Input

s =

```
"abcabcbb"
```

Output

```
3
```

Expected

```
3
```

Time Complexity : O(n)
Space Complexity : O(1)

8)Substring with Concatenation of All Words

You are given a string s and an array of strings words. All the strings of words are of the same length.

A concatenated string is a string that exactly contains all the strings of any permutation of words concatenated.

- **For example, if words = ["ab","cd","ef"], then "abcdef", "abefcd", "cdabef", "cdefab", "efabcd", and "efcdab" are all concatenated strings. "acdbef" is not a concatenated string because it is not the concatenation of any permutation of words.**

Return an array of *the starting indices* of all the concatenated substrings in s. You can return the answer in any order.

Example 1:

Input: s = "barfoothefoobarman", words = ["foo","bar"]

Output: [0,9]

Explanation:

The substring starting at 0 is "barfoo". It is the concatenation of ["bar","foo"] which is a permutation of words.
The substring starting at 9 is "foobar". It is the concatenation of ["foo","bar"] which is a permutation of words.

Example 2:

Input: s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]

Output: []

Explanation:

There is no concatenated substring.

Example 3:

Input: s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]

Output: [6,9,12]

Explanation:

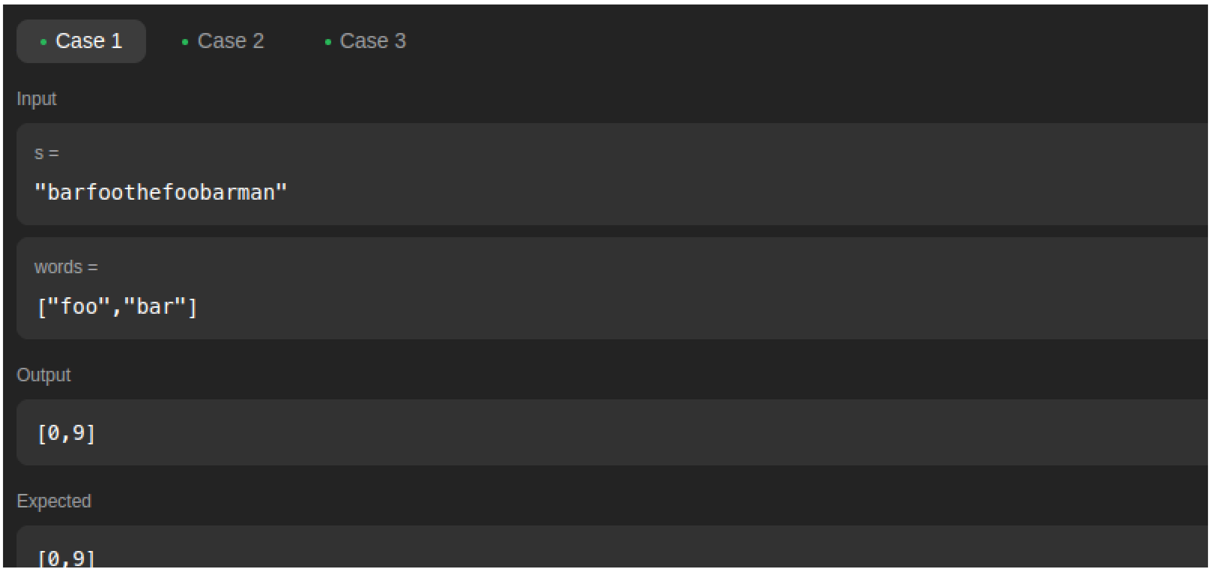
The substring starting at 6 is "foobarthe". It is the concatenation of ["foo","bar","the"].
The substring starting at 9 is "barthefoo". It is the concatenation of ["bar","the","foo"].
The substring starting at 12 is "thefoobar". It is the concatenation of ["the","foo","bar"].

Program :

```
class Solution {
  public List<Integer> findSubstring(String s, String[] words) {
    Map<String, Integer> dictionary = new HashMap<>();
    Arrays.stream(words).forEach(word -> dictionary.merge(word, 1, Integer::sum));
    int simpleWordLength = words[0].length();
    int concatenatedWordLength = simpleWordLength * words.length;
    if (s.length() < concatenatedWordLength) return new ArrayList<>();
    StringBuilder currentWord = new StringBuilder(s.substring(0, concatenatedWordLength));
    List<Integer> result = new ArrayList<>();
    for (int i = concatenatedWordLength; i < s.length(); i++) {
      if (containsFullDictionary(currentWord, new HashMap<>(dictionary), simpleWordLength))
        result.add(i - concatenatedWordLength);
      currentWord.deleteCharAt(0);
      currentWord.append(s.charAt(i));
    }
    if (containsFullDictionary(currentWord, new HashMap<>(dictionary), simpleWordLength))
      result.add(s.length() - concatenatedWordLength);
```

```
        return result;
    }


    private boolean containsFullDictionary(StringBuilder currentWord, Map<String, Integer> dictionary, int simpleWordLength) {
        int start = 0;
        int end = simpleWordLength;
        while (start < currentWord.length()) {
            String word = currentWord.substring(start, end);
            if (dictionary.containsKey(word) && dictionary.get(word) > 0) dictionary.merge(word, -1, Integer::sum);
            else return false;
            start += simpleWordLength;
            end += simpleWordLength;
        }
        return true;
    }
}
```

Output:



Input

s =

"barfoothefoobarman"

words =

["foo","bar"]

Output

[0,9]

Expected

[0,9]

Time Complexity : O(m*n*k)

Space Complexity : O(m*k)

9)Minimum Window Substring :

Given two strings s and t of lengths m and n respectively, return *the minimum window*

substring

***of* s *such that every character in* t *(including duplicates) is included in the window*. If there is no such substring, return *the empty string* "".**

The testcases will be generated such that the answer is unique.

Example 1:

Input: s = "ADOBECODEBANC", t = "ABC"

Output: "BANC"

Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input: s = "a", t = "a"

Output: "a"

Explanation: The entire string s is the minimum window.

Example 3:

Input: s = "a", t = "aa"

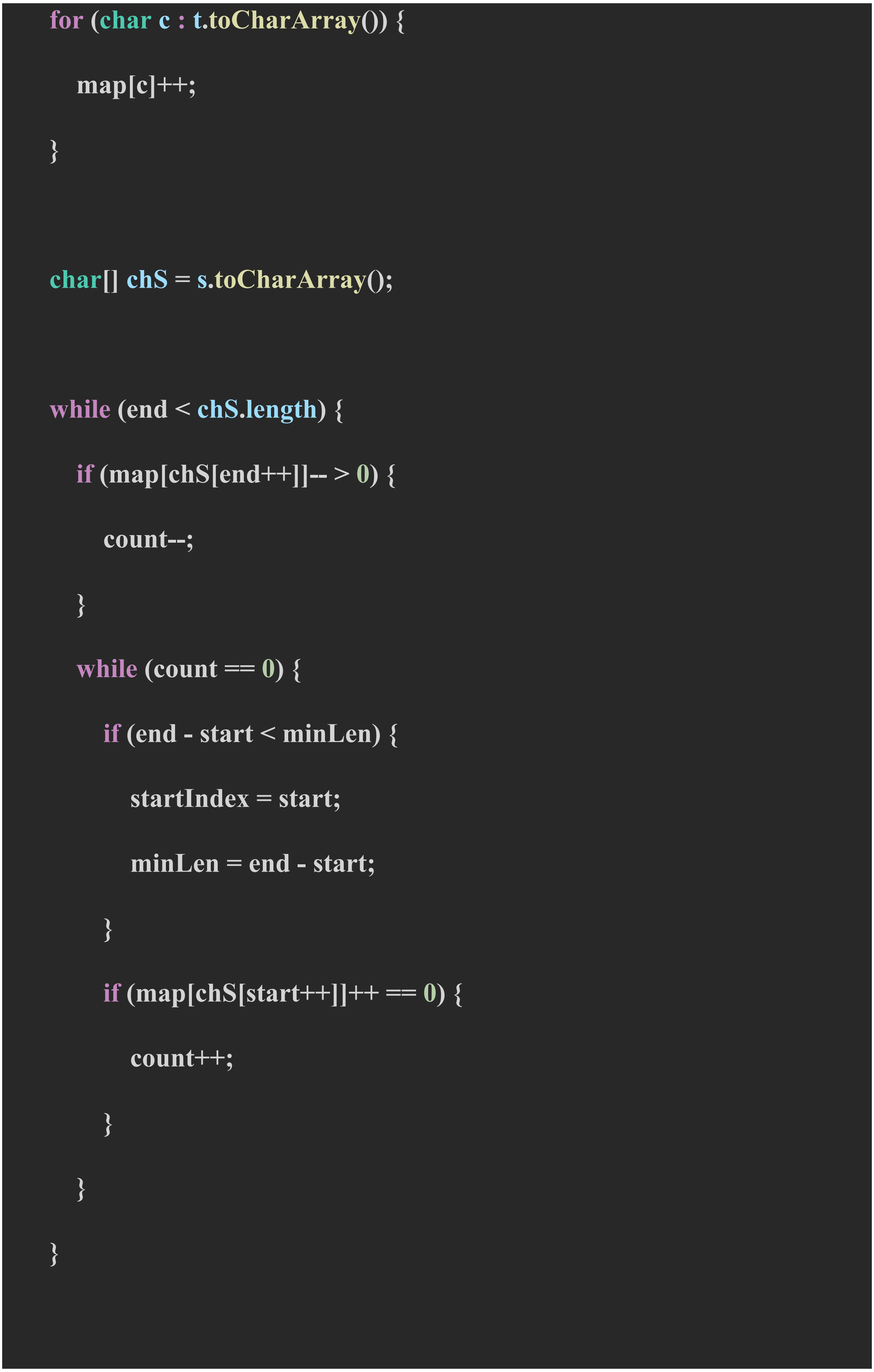
Output: ""

Explanation: Both 'a's from t must be included in the window.

Since the largest window of s only has one 'a', return empty string.

Program :

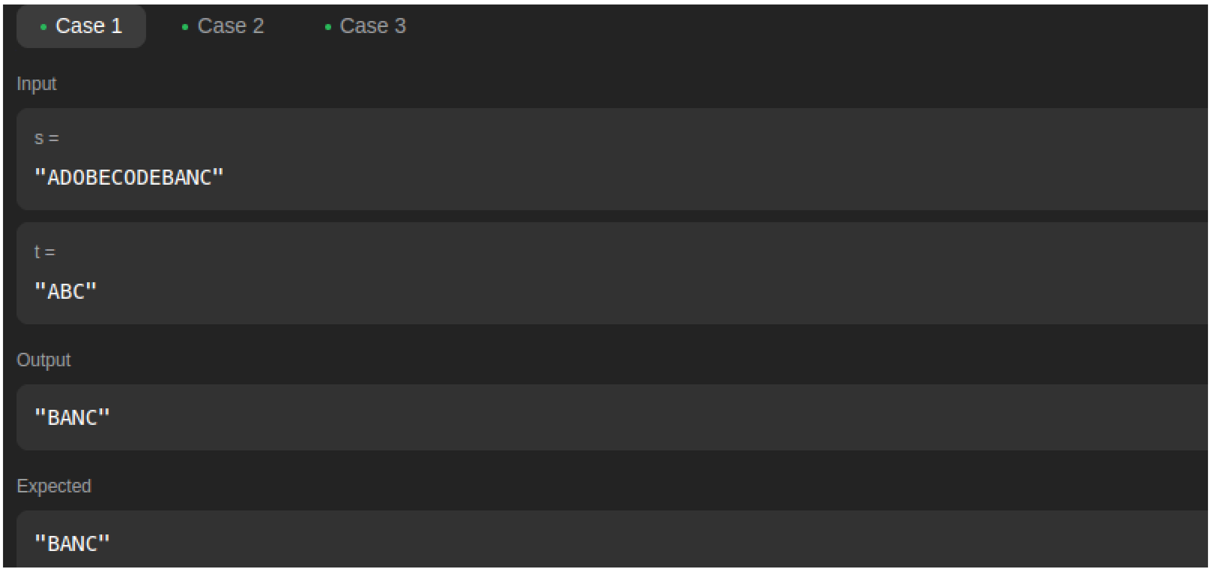
```
class Solution {

  public String minWindow(String s, String t) {

    if (s == null || t == null || s.length() == 0 || t.length() == 0 ||

          s.length() < t.length()) {

      return new String();

    }

    int[] map = new int[128];

    int count = t.length();

    int start = 0, end = 0, minLen = Integer.MAX_VALUE, startIndex = 0;

    /// UPVOTE !
```



```
for (char c : t.toCharArray()) {
    map[c]++;
}


char[] chS = s.toCharArray();


while (end < chS.length) {
    if (map[chS[end++]]-- > 0) {
        count--;
    }
    while (count == 0) {
        if (end - start < minLen) {
            startIndex = start;
            minLen = end - start;
        }
        if (map[chS[start++]]++ == 0) {
            count++;
        }
    }
}
```

```
    return minLen == Integer.MAX_VALUE ? new String() :
        new String(chS, startIndex, minLen);
  }
}
```

Output :



Input

s =

"ADOBECODEBANC"

t =

"ABC"

Output

"BANC"

Expected

"BANC"

Time Complexity : O(n)

Space Complexity : O(1)

10)Valid Parenthesis :

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. **Open brackets must be closed by the same type of brackets.**
2. **Open brackets must be closed in the correct order.**

3. **Every close bracket has a corresponding open bracket of the same type.**

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

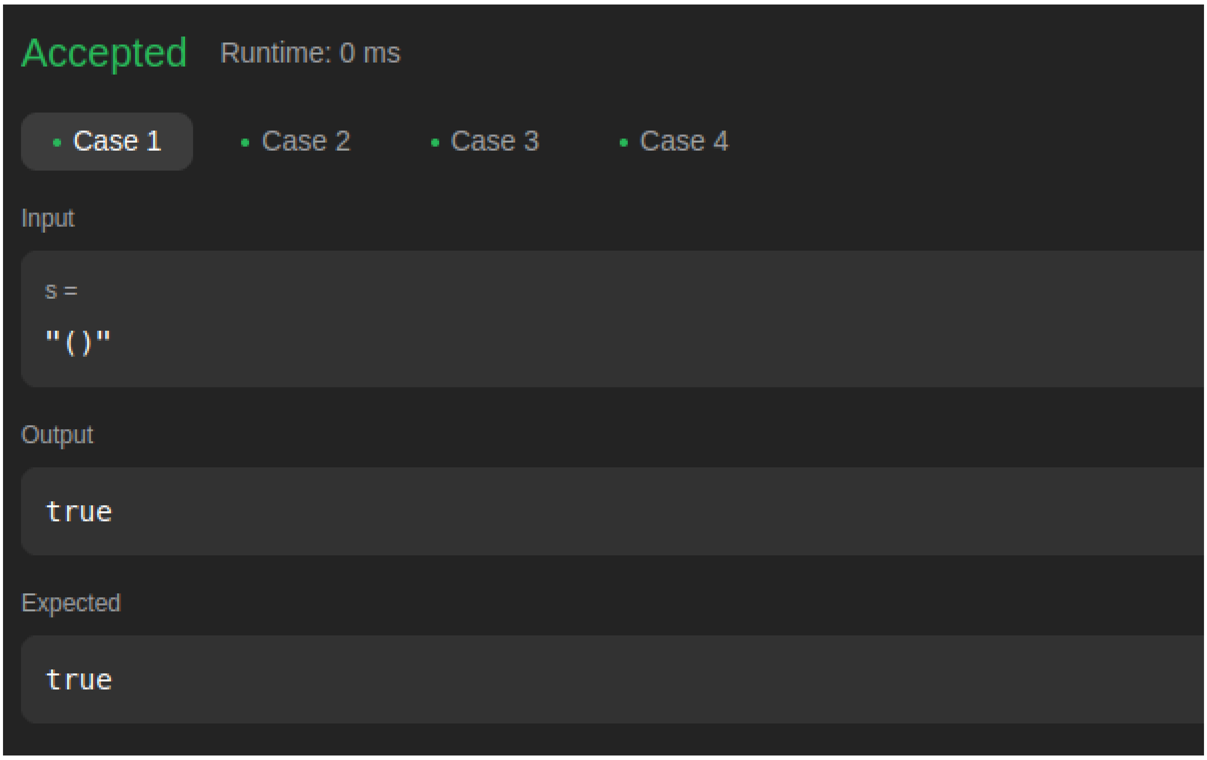
Output: true

Program :

```
class Solution {

  public boolean isValid(String s) {



  Stack<Character> st = new Stack<>();

  String open = "({[";

  String close = ")}]";
```

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

        char current = s.charAt(i);

        if(open.contains(String.valueOf(current))){

            st.push(current);

        }else{

            if(st.empty() || open.indexOf(st.pop())!=close.indexOf(s.charAt(i)) ){

                return false;

            }

        }


    }

    return st.empty();


    }

}
```

Output :



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3   • Case 4

Input
s =
"()"

Output
true

Expected
true
```

Time Complexity : O(n)

Space Complexity : O(n)

11) Simplify Path :

You are given an *absolute* path for a Unix-style file system, which always begins with a slash '/'. Your task is to transform this absolute path into its simplified canonical path.

The *rules* of a Unix-style file system are as follows:

- **A single period '.' represents the current directory.**
- **A double period '..' represents the previous/parent directory.**
- **Multiple consecutive slashes such as '//' and '///' are treated as a single slash '/'.**
- **Any sequence of periods that does not match the rules above should be treated as a valid directory or file name. For example, '...' and '....' are valid directory or file names.**

The simplified canonical path should follow these *rules*:

- **The path must start with a single slash '/'.**
- **Directories within the path must be separated by exactly one slash '/'.**

- **The path must not end with a slash '/', unless it is the root directory.**
- **The path must not have any single or double periods ('.' and '..') used to denote current or parent directories.**

Return the simplified canonical path.

Example 1:

Input: path = "/home/"

Output: "/home"

Explanation:

The trailing slash should be removed.

Example 2:

Input: path = "/home//foo/"

Output: "/home/foo"

Explanation:

Multiple consecutive slashes are replaced by a single one.

Example 3:

Input: path = "/home/user/Documents/../Pictures"

Output: "/home/user/Pictures"

Explanation:

A double period ".." refers to the directory up a level (the parent directory).

Example 4:

Input: path = "/../"

Output: "/"

Explanation:

Going one level up from the root directory is not possible.

Example 5:

Input: path = "/.../a/../b/c/../d/./"

Output: "/.../b/d"

Explanation:

"..." is a valid name for a directory in this problem.

Program :

Output :

```
//Algo Used: Stack

// TC: O N , SC: O N

public class Solution {

    public static String simplifyPath(String path) {

        Stack<String> stack = new Stack<>();



        // Split the input path by "/"

        String[] components = path.split("/");



        // Traverse each component

        for (String component : components) {

            // Skip empty components and "." (current directory)

            if (component.equals("") || component.equals(".")) {
                continue;
            }
            // If "..", pop the stack if it's not empty (going back to the parent directory)
            if (component.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop();
                }
            } else {
                // Push valid directory names onto the stack
                stack.push(component);
            }
        }

        // If stack is empty, return "/"
        if (stack.isEmpty()) {
            return "/";
        }

        // Construct the simplified path
        StringBuilder result = new StringBuilder();
```

```
    for (String dir : stack) {

      result.append("/").append(dir);

    }


    return result.toString();

  }

}
```

Time Complexity :

O(n)

Space Complexity :

O(n)

12)Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- **MinStack() initializes the stack object.**
- **void push(int val) pushes the element val onto the stack.**
- **void pop() removes the element on the top of the stack.**
- **int top() gets the top element of the stack.**
- **int getMin() retrieves the minimum element in the stack.**

You must implement a solution with O(1) time complexity for each function.

Example 1:

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

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

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top();   // return 0

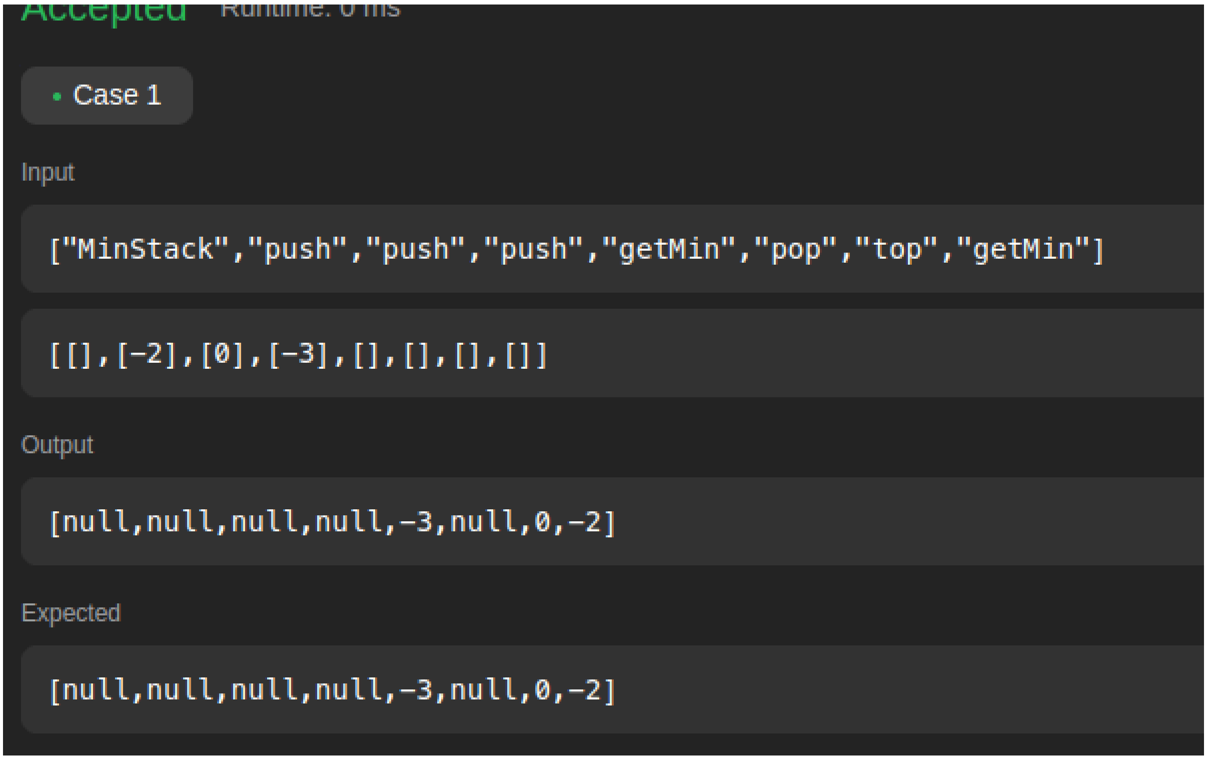
minStack.getMin(); // return -2

Program :

```
class MinStack {

  int min = Integer.MAX_VALUE;

  Stack<Integer> stack = new Stack<Integer>();

  public void push(int x) {
        // only push the old minimum value when the current
        // minimum value changes after pushing the new value x
        if(x <= min){
            stack.push(min);
            min=x;
        }
        stack.push(x);
    }

    public void pop() {
        // if pop operation could result in the changing of the current minimum value,
        // pop twice and change the current minimum value to the last minimum value.
        if(stack.pop() == min) min=stack.pop();
    }

    public int top() {
        return stack.peek();
    }
```

```
  public int getMin() {

     return min;

  }

}
```

Output :



Accepted Runtime: 0 ms

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]
```

Time Complexity : O(1)

Space Complexity : O(n)

13)Evaluate reverse polished notation :

You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression*.

Note that:

- **The valid operators are '+', '-', '*', and '/'.**
- **Each operand may be an integer or another expression.**
- **The division between two integers always truncates toward zero.**
- **There will not be any division by zero.**
- **The input represents a valid arithmetic expression in a reverse polish notation.**
- **The answer and all the intermediate calculations can be represented in a 32-bit integer.**

Example 1:

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

Output: 9

Explanation: ((2 + 1) * 3) = 9

Example 2:

Input: tokens = ["4","13","5","/","+"]

Output: 6

Explanation: (4 + (13 / 5)) = 6

Example 3:

Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]

Output: 22

Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5

= ((10 * (6 / (12 * -11))) + 17) + 5

= ((10 * (6 / -132)) + 17) + 5

= ((10 * 0) + 17) + 5

= (0 + 17) + 5

= 17 + 5

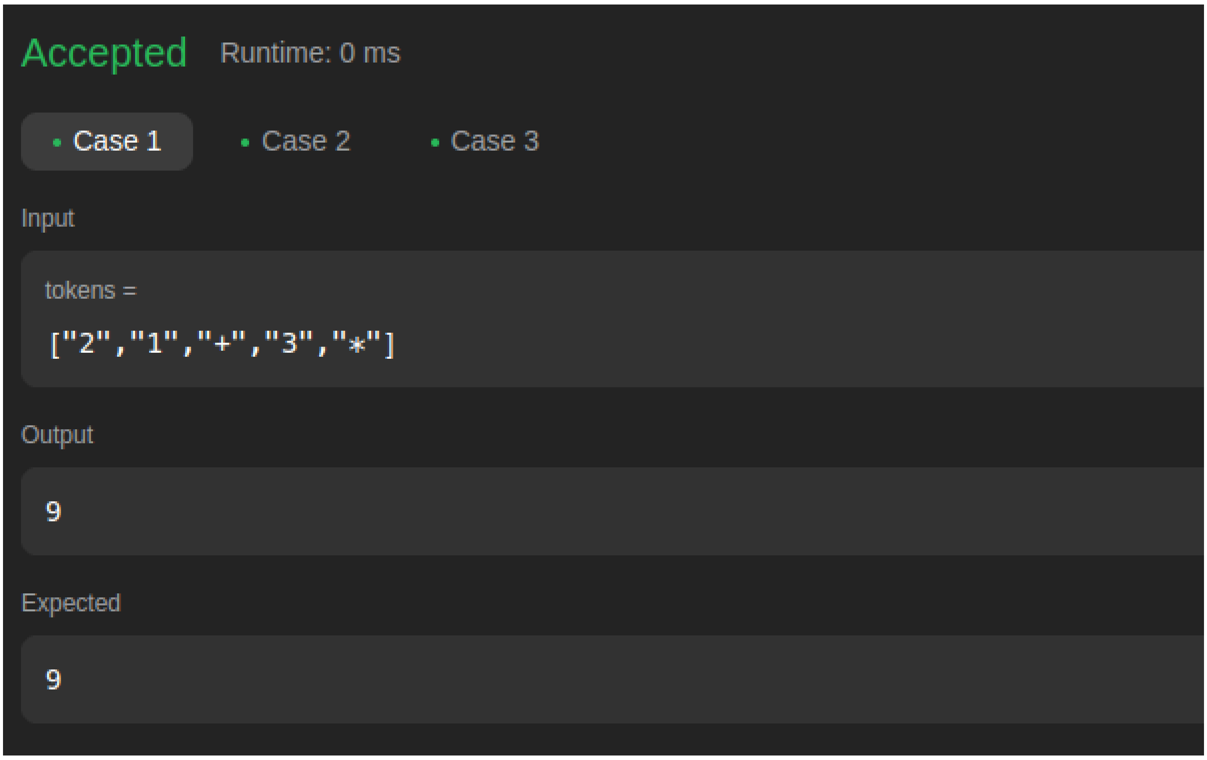
= 22

Program :

```
class Solution {

  long resolves(long a, long b, char Operator) {

     if (Operator == '+') return a + b;

     else if (Operator == '-') return a - b;

     else if (Operator == '*') return a * b;

     return a / b;

  }


  public int evalRPN(String[] tokens) {

     Stack<Long> stack = new Stack<>();

     int n = tokens.length;

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

        if (tokens[i].length() == 1 && tokens[i].charAt(0) < 48) {

           long integer2 = stack.pop();
```

```
                long integer1 = stack.pop();

                char operator = tokens[i].charAt(0);

                long resolvedAns = resolves(integer1, integer2, operator);

                stack.push(resolvedAns);

            } else {

                stack.push(Long.parseLong(tokens[i]));

            }

        }

        return stack.pop().intValue();

    }

}
```

Output :



Accepted Runtime: 0 ms

Input

tokens =

```
["2","1","+","3","*"]
```

Output

```
9
```

Expected

```
9
```

Time Complexity :O(n)

Space Complexity:O(n)

14)Basic Calculator :

Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return *the result of the evaluation*.

Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

Example 1:

Input: s = "1 + 1"

Output: 2

Example 2:

Input: s = " 2-1 + 2 "

Output: 3

Example 3:

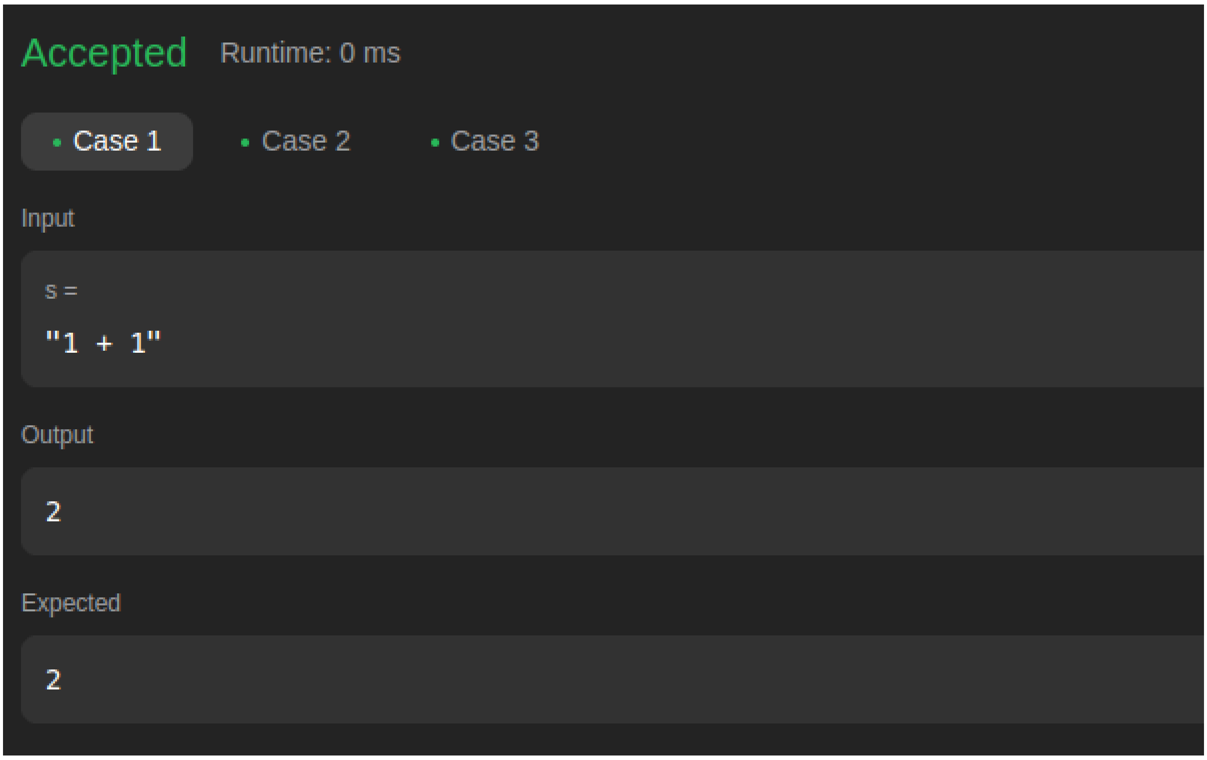
Input: s = "(1+(4+5+2)-3)+(6+8)"

Output: 23

Program :

```
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;
  }
}
```

Output :


Accepted Runtime: 0 ms

Input

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

Output

```
2
```

Expected

```
2
```

Time Complexity : O(n)

Space Complexity : O(n)

15)Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5

Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2

Output: 1

Example 3:

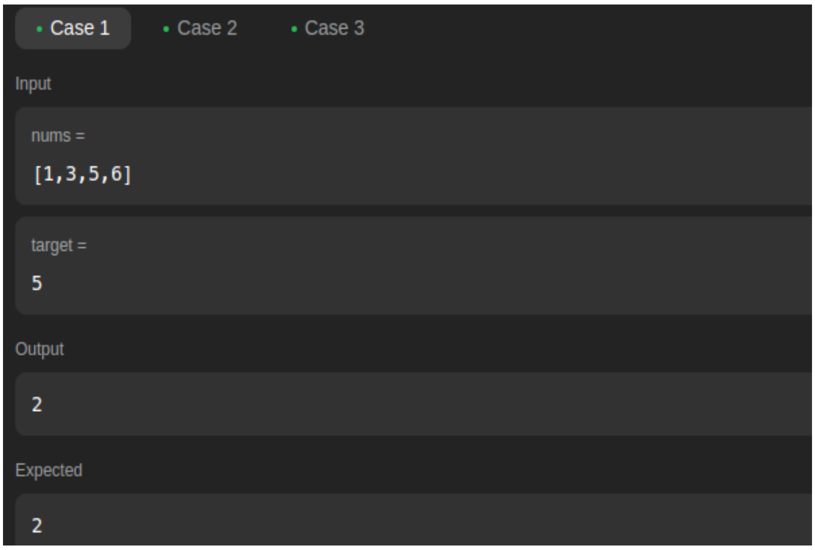
Input: nums = [1,3,5,6], target = 7

Output: 4

Program :

```
class Solution {

  public int searchInsert(int[] nums, int target) {

     return binarySearch(nums, target);

  }


  private int binarySearch(int[] nums, int target) {

     int low = 0;

     int high = nums.length - 1;


     while (low <= high) {

        int mid = (low + high) / 2;


        if (nums[mid] > target) {

           high = mid - 1;

        } else if (nums[mid] < target) {
                low = mid + 1;

            } else {

                return mid;

            }

        }


        return low;

    }

}
```

Output :



• Case 1 • Case 2 • Case 3

Input

nums =

```
[1,3,5,6]
```

target =

```
5
```

Output

```
2
```

Expected

```
2
```

TIme Complexity:O(logn)

Space Complexity : O(1)

16)Search a 2D Matrix :

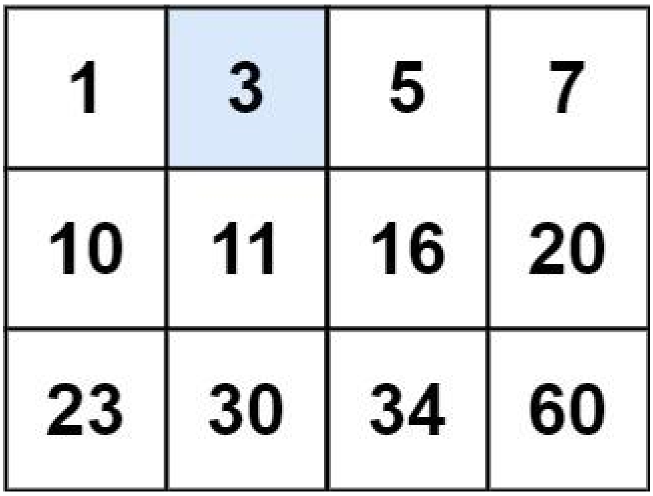
You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise.*

You must write a solution in O(log(m * n)) time complexity.

Example 1:

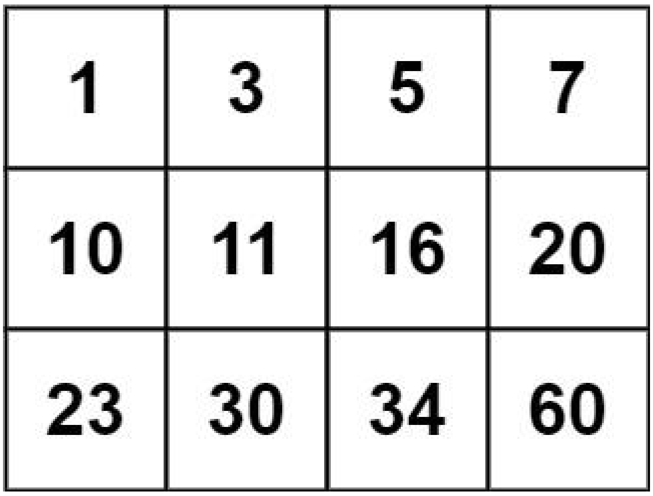


1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

Output: true

Example 2:



1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

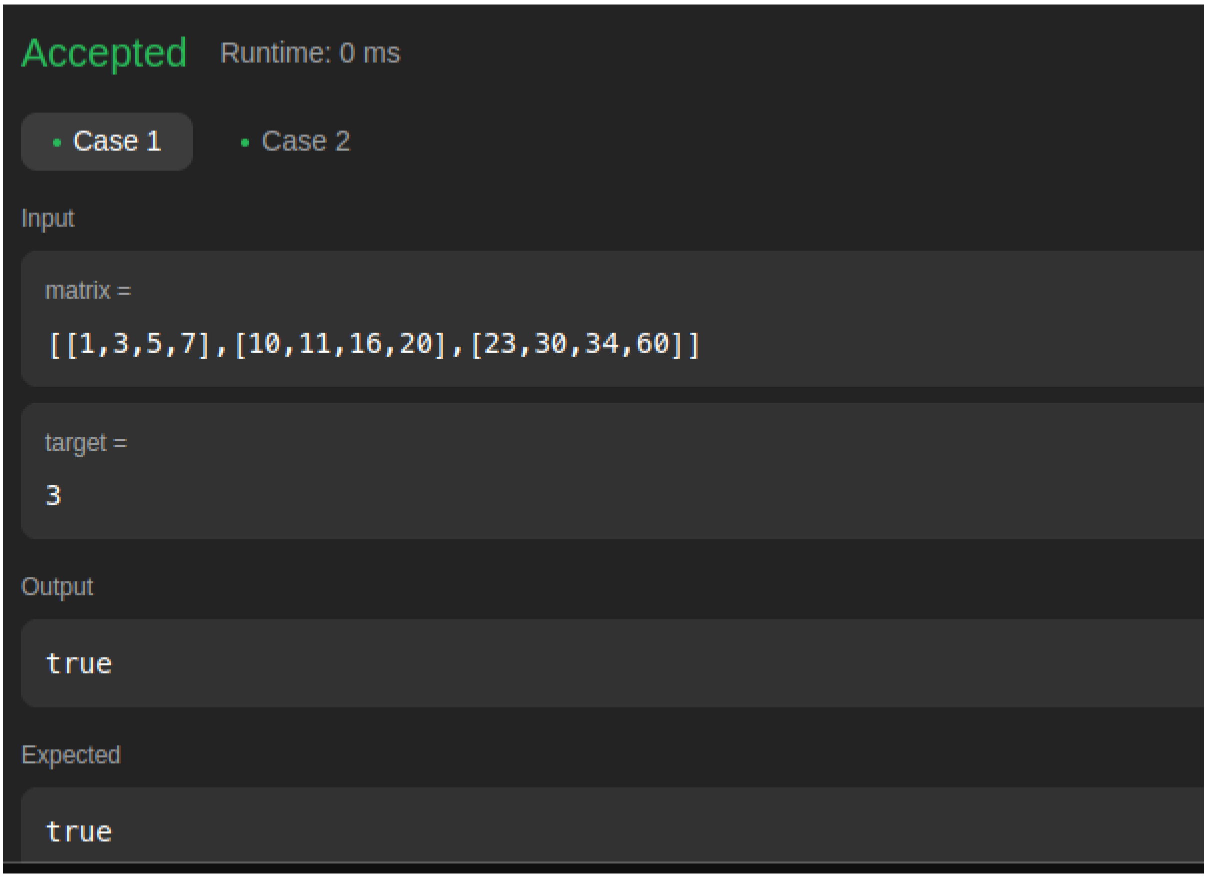
Output: false

Program :

```
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;
    }
}
```

Output :



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

matrix =

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

target =

```
3
```

Output

```
true
```

Expected

```
true
```

Time Complexity : O(m+n)

Space Complexity : O(1)

17)Find a Peak Element :

A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

Example 1:

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

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

Input: nums = [1,2,1,3,5,6,4]

Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

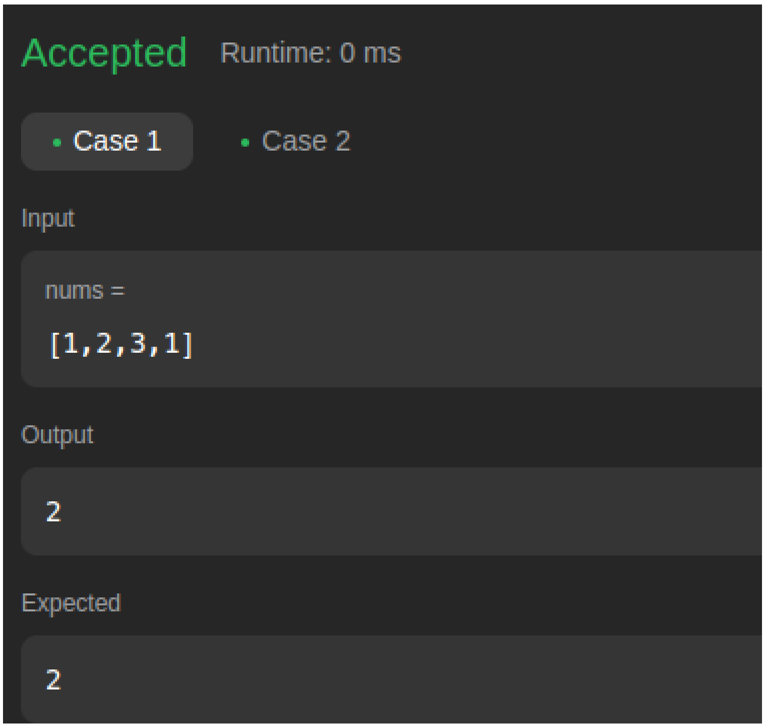
Program :

```
class Solution {

  public int findPeakElement(int[] nums) {

     if(nums.length <= 1){

        return 0;
```

```
    }

    return helper(nums , 0 , nums.length - 1);

}

public int helper(int[] nums , int si , int ei) {

    if(ei-si <= 0)

    return -1;

    if(ei - si == 1) {

        if(nums[si] > nums[ei])

        return si;

        else

        return ei;

    }

    int mid = si + (ei - si)/2;

    if(nums[mid] > nums[mid+1] && nums[mid] > nums[mid-1]){

        return mid;

    }else if(nums[si] > nums[si+1])

    return si;

    else if(nums[ei] > nums[ei-1])

    return ei;

    else {

        int i = helper(nums , si , mid - 1);
```

```
      int j = helper(nums,mid,ei);

      if(i != -1 && j != -1) {

        if(nums[i] > nums[j])

        return i;

        else

        return j;

      }else {

        if(i != -1)

        return i;

        else

        return j;

      }

    }

  }

}
```

Output :


Accepted Runtime: 0 ms

Input

nums =

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

Output

```
2
```

Expected

```
2
```

Time Complexity : O(n)

Space Complexity : O(n)

18)Search in Rotated Sorted Array :

There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0

Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Example 3:

Input: nums = [1], target = 0

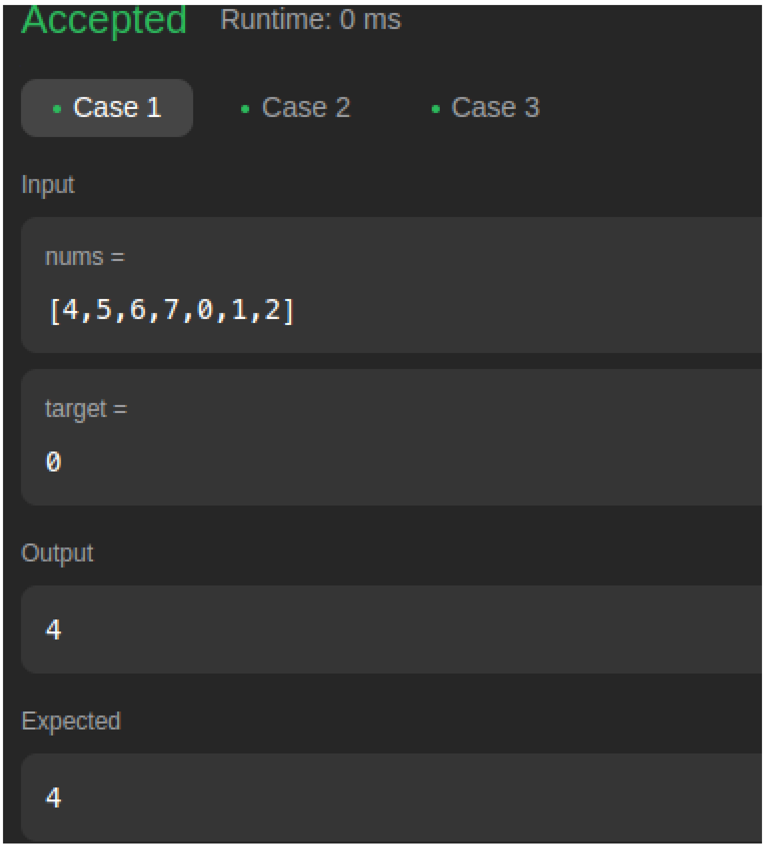
Output: -1

Program :

```
class Solution {

  public int search(int[] arr, int target) {

     int n= arr.length;

     int lo=0;

     int hi=n-1;

     while(lo<=hi){

        int mid= lo+ (hi-lo)/2;

        if(arr[mid]==target) return mid;

        else if(arr[mid] <=arr[hi]){ // i am in right sorted array,mid to hi everything is sorted

           if(target >arr[mid] &&  target <=arr[hi]) lo=mid+1;

              else hi=mid-1;



        }
```

```
            else { // i am in left sorted array lo to mid everything sorted

                if(target >=arr[lo] &&  target <arr[mid]){

                    hi=mid-1;

                }

                else lo =mid+1;

            }


        }

        return -1;

    }


}
```

Output :



Accepted Runtime: 0 ms

```
Input
nums =
[4,5,6,7,0,1,2]
target =
0
Output
4
Expected
4
```

Time Complexity : O(logn)

Space Complexity : O(1)

19) Find First and Last position of an element in an Sorted Array :

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Example 3:

Input: nums = [], target = 0

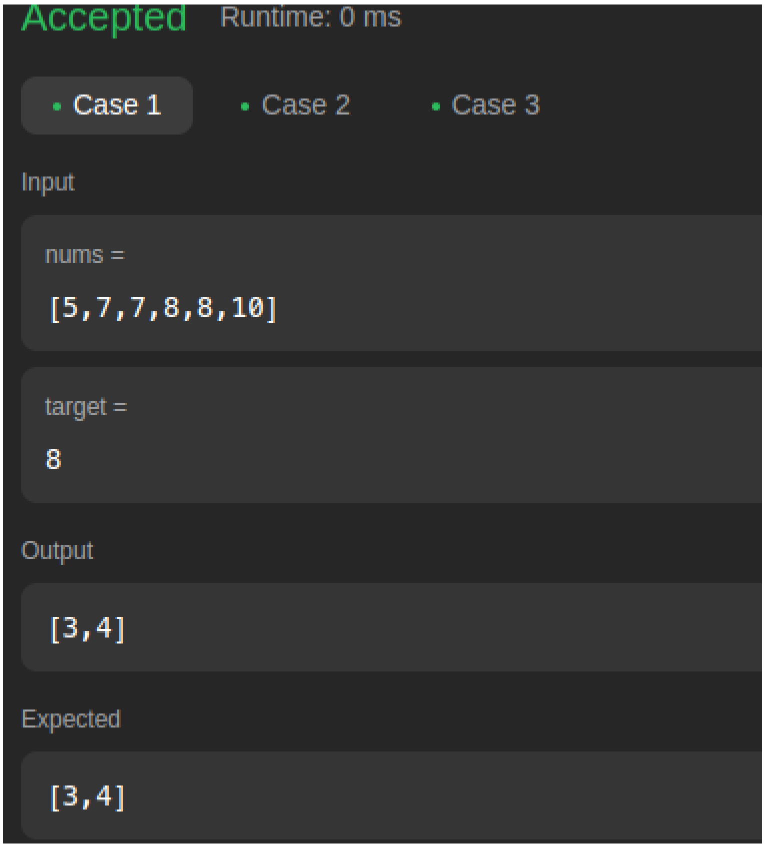
Output: [-1,-1]

Program :

```
class Solution {

  public int[] searchRange(int[] arr, int target) {

     int n= arr.length;

     int[] ans={-1,-1};



     // fist postion

     int lo=0;

     int hi=n-1;

     int fp=-1;

     while(lo<=hi){

        int mid=lo +(hi-lo)/2;

        if(arr[mid] == target){

           if( mid>0 && arr[mid] == arr[mid-1]) hi=mid-1;
        else{
            fp=mid;
            break;
        }
    }
    else if(arr[mid]<target) lo=mid+1;
    else hi=mid-1;
}
// last position
 lo=0;
 hi=n-1;
int lp=-1;
while(lo<=hi){
    int mid=lo +(hi-lo)/2;
    if(arr[mid] == target){
        if( mid+1 <n && arr[mid] == arr[mid+1]) lo=mid+1;
        else{
            lp=mid;
            break;
        }
    }
```

```
            else if(arr[mid]<target) lo=mid+1;

            else hi=mid-1;

        }

        ans[0]=fp;

        ans[1]=lp;

        return ans;

    }

}
```

Output :



Input

nums =

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

target =

```
8
```

Output

```
[3,4]
```

Expected

```
[3,4]
```

Time Complexity : O(logn)

Space Complexity : O(1)

20)Find Minimum in an rotated Sorted Array

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

- **[4,5,6,7,0,1,2] if it was rotated 4 times.**
- **[0,1,2,4,5,6,7] if it was rotated 7 times.**

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

Example 1:

Input: nums = [3,4,5,1,2]

Output: 1

Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Example 2:

Input: nums = [4,5,6,7,0,1,2]

Output: 0

Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Example 3:

Input: nums = [11,13,15,17]

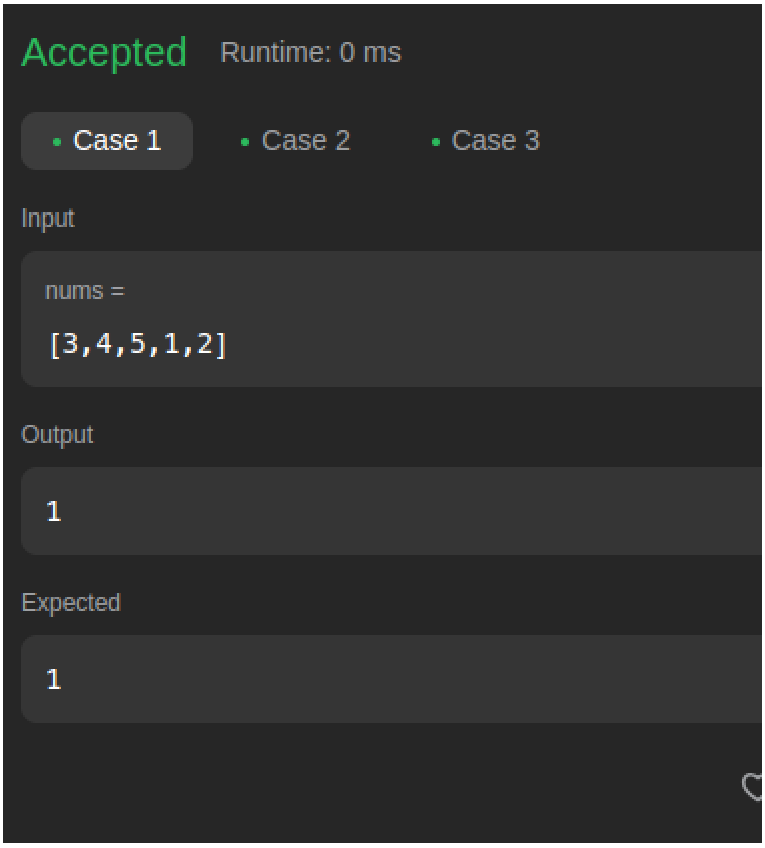
Output: 11

Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

Program :

```
class Solution {

  public int findMin(int[] nums) {

     int start=0;

     int end=nums.length-1;

     if(nums[end] > nums[start]){

        return nums[start];

     }

     return minElement(nums, start, end);

  }


  public static int minElement(int[] nums, int start, int end) {

     while (start < end) {

        int mid = start + (end - start) / 2;

        if (nums[mid] > nums[mid + 1]) {

           return nums[mid+1];

        }

        if (nums[mid] >= nums[start]) {
                start = mid + 1;
            } else {
                end = mid;
            }
        }
        return nums[start];
    }
}
```

Output:



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input

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

Output

1

Expected

1
```

Time Complexity : O(logn)

Space Complexity : O(1)