

# DSA PRACTICE - 9

Name : Sidharth.N

Dept : AI & DS - 'C'

## *Two Pointers*

### 1. Valid Palindrome

```
class Solution {
    public boolean isPalindrome(String s) {
        if(s.isEmpty()){
            return true;
        }

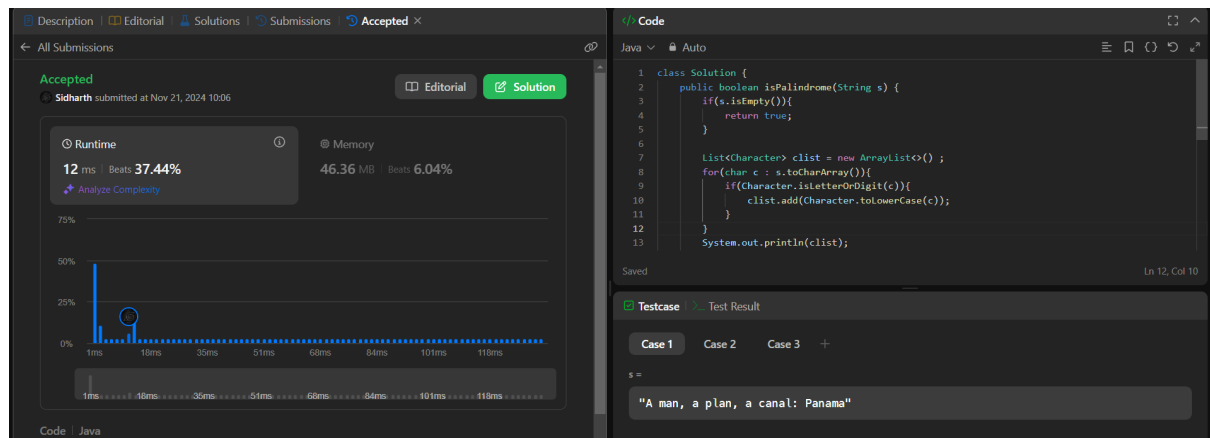
        List<Character> clist = new ArrayList<>() ;
        for(char c : s.toCharArray()){
            if(Character.isLetterOrDigit(c)){
                clist.add(Character.toLowerCase(c));
            }
        }
        System.out.println(s.get(0));

        int i = 0;
        int j = clist.size()-1;

        while (i<j){
            if(clist.get(i) != clist.get(j)){
                return false;
            }
            i++;
            j--;
        }
        return true;
    }
}
```

Time Complexity :  $O(n)$

Output:



## 2. Is Subsequence

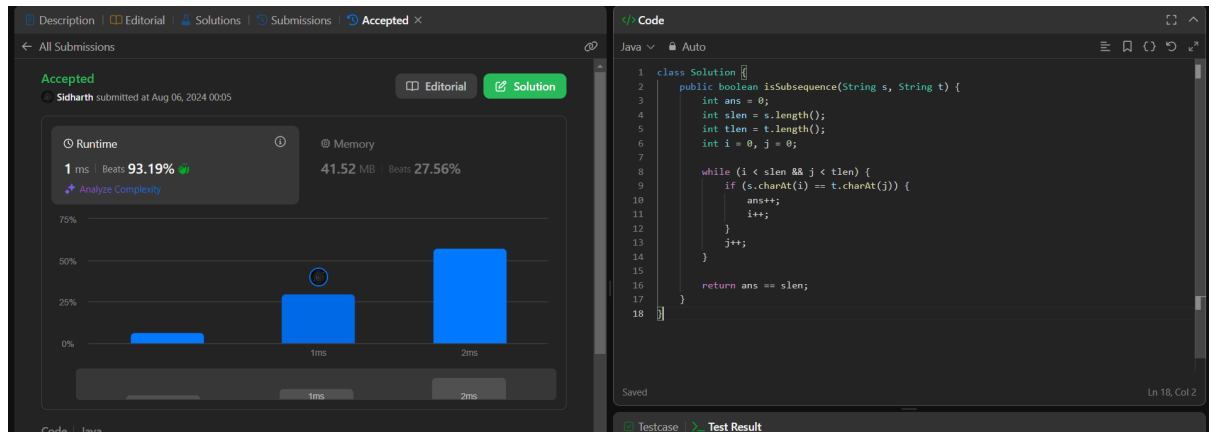
```
class Solution {
    public boolean isSubsequence(String s, String t) {
        int ans = 0;
        int slen = s.length();
        int tlen = t.length();
        int i = 0, j = 0;

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

        return ans == slen;
    }
}
```

Time Complexity :  $O(n)$

Output:



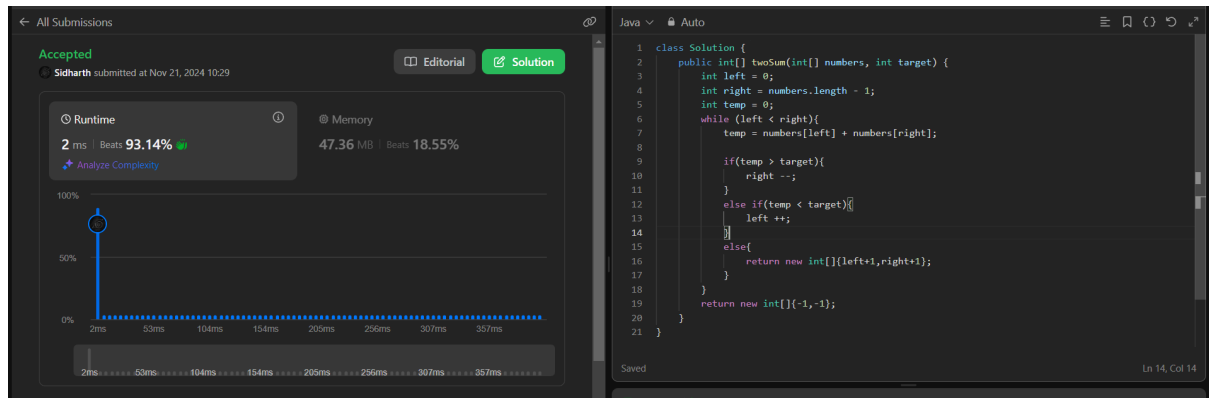
### 3. Two Sum - ||

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int left = 0;
        int right = numbers.length - 1;
        int temp = 0;
        while (left < right){
            temp = numbers[left] + numbers[right];

            if(temp > target){
                right --;
            }
            else if(temp < target){
                left ++;
            }
            else{
                return new int[]{left+1,right+1};
            }
        }
        return new int[]{-1,-1};
    }
}
```

Time Complexity :  $O(n)$

Output:



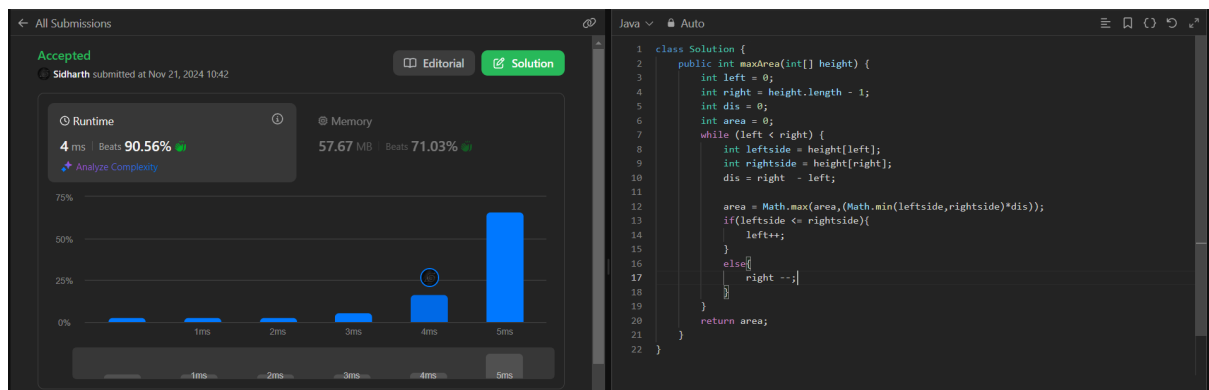
#### 4.Container with most water

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

            area = Math.max(area, (Math.min(leftside, rightside) * dis));
            if (leftside <= rightside) {
                left++;
            } else {
                right--;
            }
        }
        return area;
    }
}
```

Time Complexity :  $O(n)$

Output:

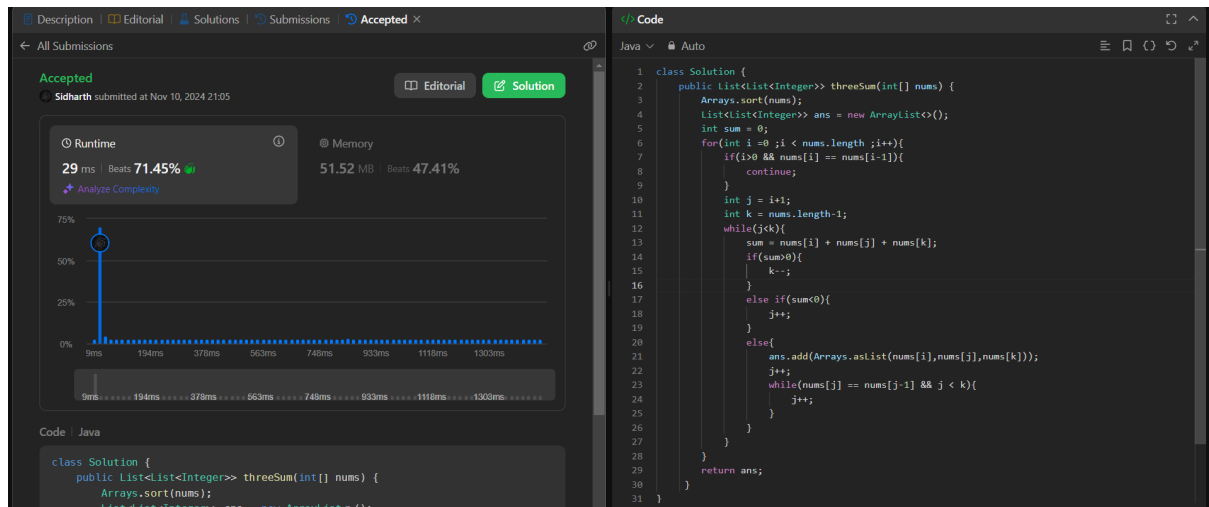


### 5. 3Sum

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

Time Complexity :  $O(n^2)$

Output:



## Sliding Window

### 1. Minimum size subarray

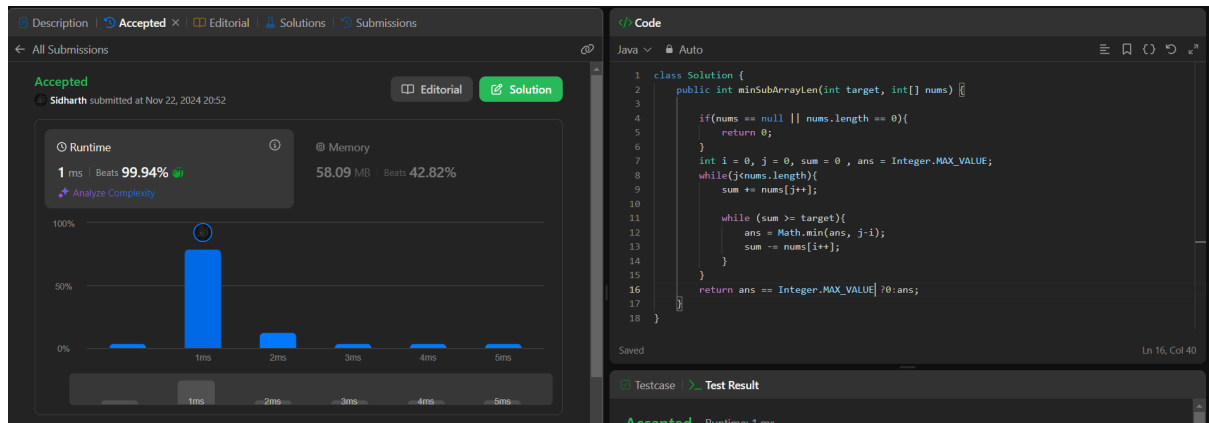
```
class Solution {
    public int minSubArrayLen(int target, int[] nums) {

        if(nums == null || nums.length == 0){
            return 0;
        }
        int i = 0, j = 0, sum = 0 , ans = Integer.MAX_VALUE;
        while(j < nums.length){
            sum += nums[j++];

            while (sum >= target){
                ans = Math.min(ans, j-i);
                sum -= nums[i++];
            }
        }
        return ans == Integer.MAX_VALUE ? 0 : ans;
    }
}
```

Time Complexity :  $O(n)$

Output:

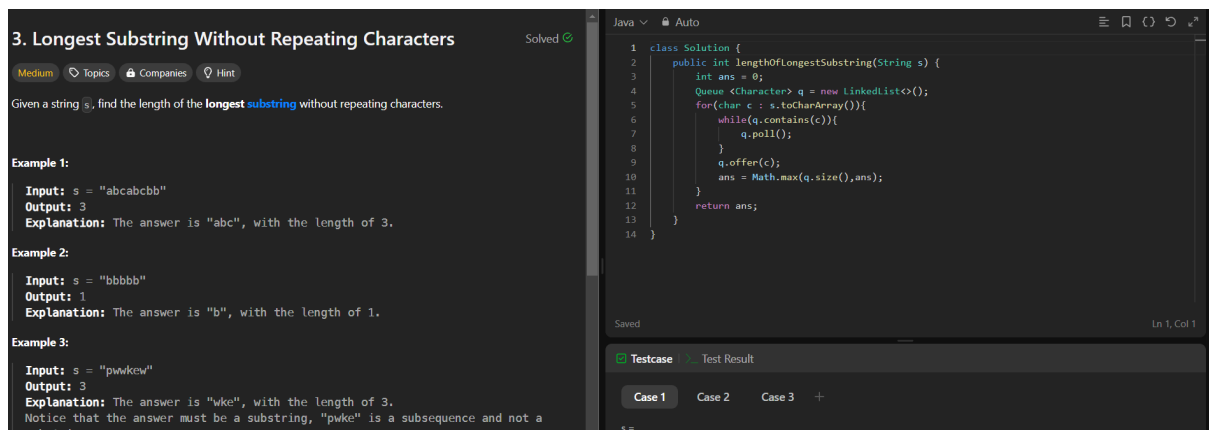


## 2. Longest Substring Without Repeating Characters

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int ans = 0;
        Queue<Character> q = new LinkedList<>();
        for(char c : s.toCharArray()){
            while(q.contains(c)){
                q.poll();
            }
            q.offer(c);
            ans = Math.max(q.size(), ans);
        }
        return ans;
    }
}
```

Time Complexity :  $O(n)$

Output:



3.

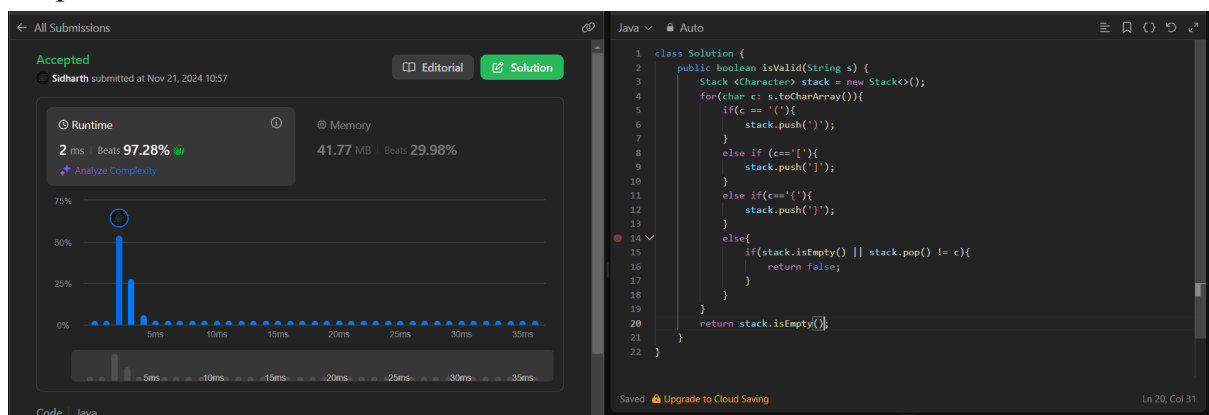
## Stack

### 1. 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:





## 2.Simplify Path

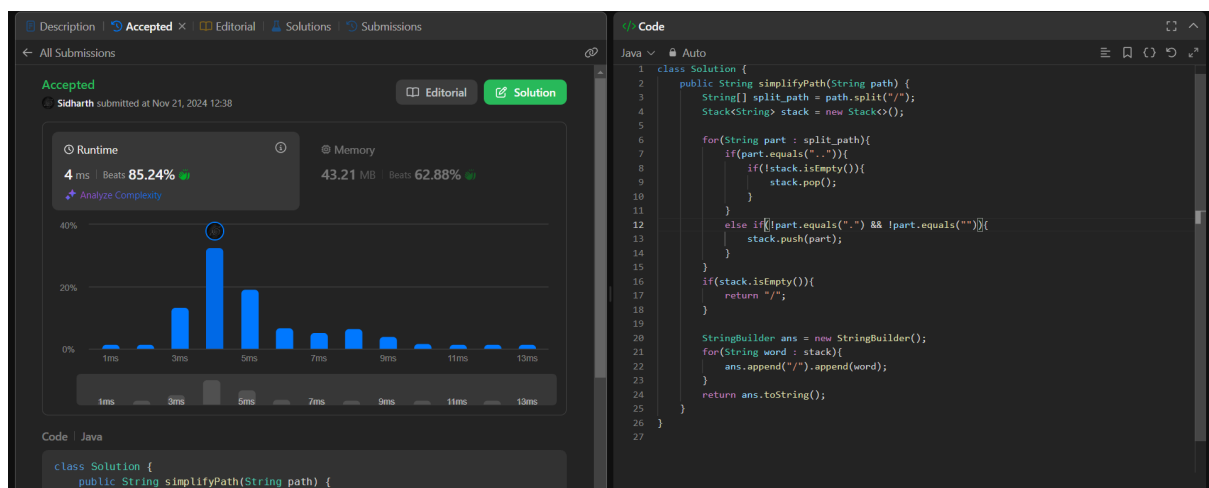
```
class Solution {
    public String simplifyPath(String path) {
        String[] split_path = path.split("/");
        Stack<String> stack = new Stack<>();

        for(String part : split_path){
            if(part.equals("..")){
                if(!stack.isEmpty()){
                    stack.pop();
                }
            }
            else if(!part.equals(".") && !part.equals("")){
                stack.push(part);
            }
        }

        if(stack.isEmpty()){
            return "/";
        }

        StringBuilder ans = new StringBuilder();
        for(String word : stack){
            ans.append("/").append(word);
        }
        return ans.toString();
    }
}
```

Time Complexity :  $O(n)$



### 3.Min Stack

```
class MinStack {
    public Node head;
    public MinStack() {

    }

    public void push(int val) {
        if(head == null){
            head = new Node(val,val,null);
        }
        else{
            head = new Node(val,Math.min(head.min,val),head);
        }
    }

    public void pop() {
        head = head.next;
    }

    public int top() {
        return head.val;
    }

    public int getMin() {
        return head.min;
    }
    private class Node{
        int val;
        int min;
        Node next;

        Node(int val , int min , Node next){
            this.val = val;
            this.min = min;
            this.next = next;
        }
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(val);
```

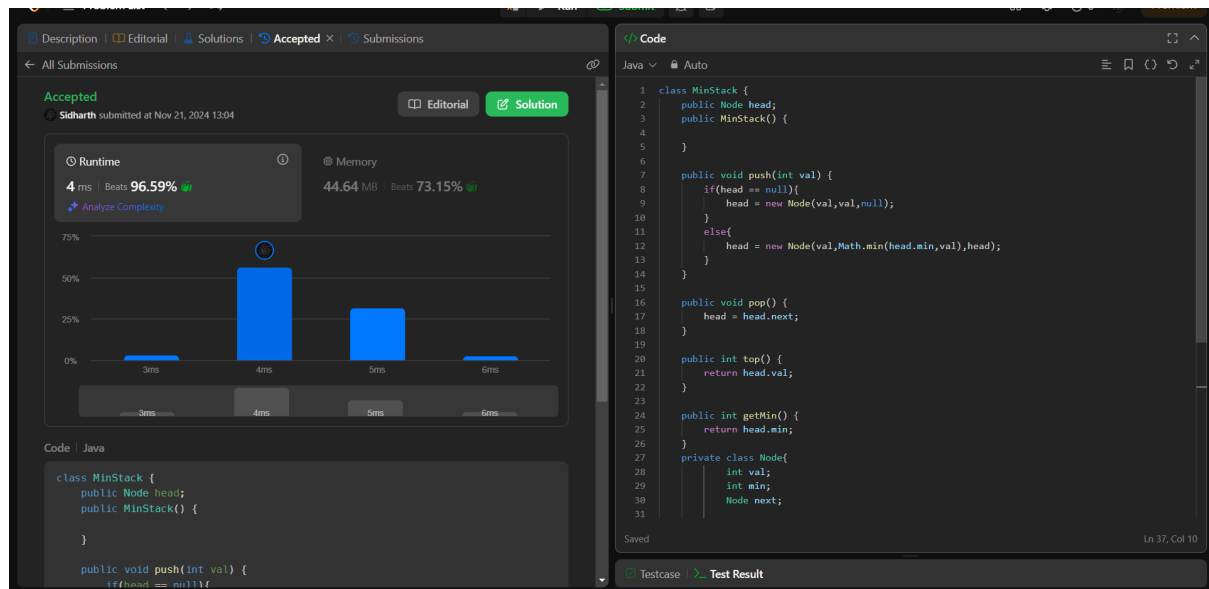
```

* obj.pop();
* int param_3 = obj.top();
* int param_4 = obj.getMin();
*/

```

Time Complexity :  $O(1)$

Output:



4. Evaluate reverse polish notation

```

class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();
        int a = 0;
        int b = 0;
        for(String element : tokens){
            if(element.equals("+")){
                stack.push(stack.pop() + stack.pop());
            }
            else if(element.equals("-")){
                a = stack.pop();
                b = stack.pop();
                stack.push(b-a);
            }
            else if(element.equals("*")){
                stack.push(stack.pop() * stack.pop());
            }
        }
    }
}

```

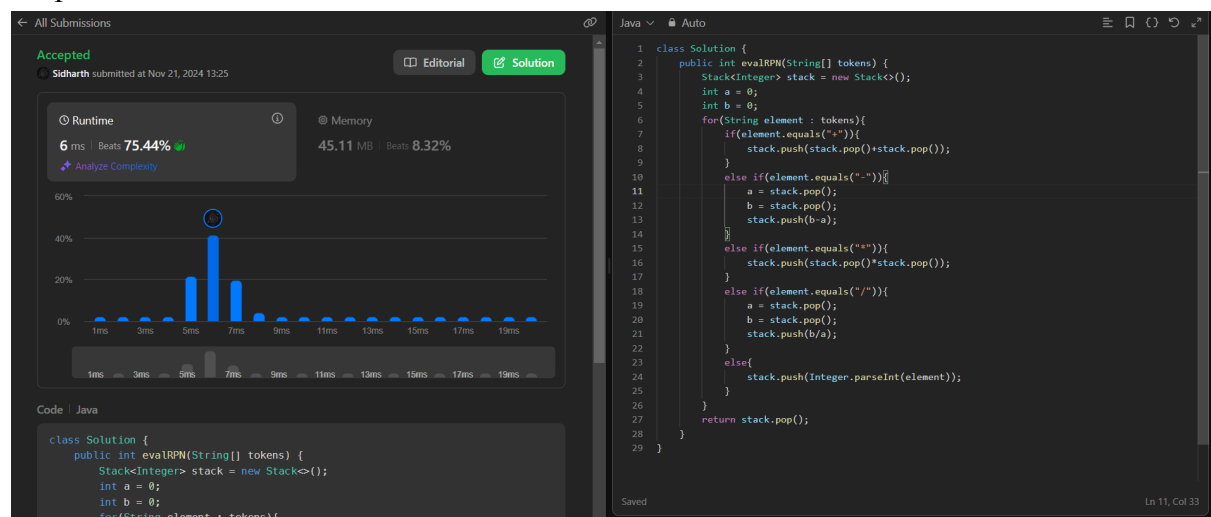
```

    }
    else if(element.equals("/")){
        a = stack.pop();
        b = stack.pop();
        stack.push(b/a);
    }
    else{
        stack.push(Integer.parseInt(element));
    }
}
return stack.pop();
}
}

```

Time Complexity :  $O(n)$

Output:



## 5. Basic Calculator

```

public int calculate(String s) {
    Stack<Integer> stack = new Stack<Integer>();
    int result = 0;
    int number = 0;
    int sign = 1;
    for(int i = 0; i < s.length(); i++){
        char c = s.charAt(i);
        if(Character.isDigit(c)){
            number = 10 * number + (int)(c - '0');
        }else if(c == '+'){
            result += sign * number;
            number = 0;
        }
    }
}

```

```

        sign = 1;
    }else if(c == '-'){
        result += sign * number;
        number = 0;
        sign = -1;
    }else if(c == '('){
        stack.push(result);
        stack.push(sign);
        sign = 1;
        result = 0;
    }else if(c == ')'){
        result += sign * number;
        number = 0;
        result *= stack.pop();
        result += stack.pop();

    }
}
if(number != 0) result += sign * number;
return result;
}

```

Time Complexity :  $O(n)$

## *Binary Search*

### 1. Search Insert Position

```

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

```

```

        low = mid + 1;
    }
}
return low;
}
}

```

Time Complexity :  $O(n)$

Output:

The screenshot shows a LeetCode submission interface. On the left, the 'Accepted' status is confirmed for a submission by 'Sidharth' at Nov 11, 2024 14:44. Performance metrics are displayed: Runtime is 0 ms (beats 100.00%) and Memory is 42.66 MB (beats 82.63%). On the right, the Java code is shown, which implements a binary search algorithm on each row of the matrix.

```

1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int low = 0;
4         int high = nums.length-1;
5         int mid = 0;
6         while(low <= high){
7             mid = (low+high)/2;
8             if(nums[mid] == target){
9                 return mid;
10            }
11            else if(target < nums[mid]){
12                high = mid-1;
13            }
14            else{
15                low = mid +1;
16            }
17        }
18        return low;
19    }
20 }

```

## 2. Search a 2D Matrix

```

class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        for(int [] row : matrix){
            int low = 0;
            int high = row.length - 1;
            int mid = 0;
            while (low<=high){
                mid = (high+low)/2;
                if(target > row[mid]){
                    low = mid + 1;
                }
                else if(target == row[mid]){
                    return true;
                }
            }
        }
    }
}

```

```

        else{
            high = mid - 1;
        }
    }
}

return false;
}
}

```

Time Complexity :  $O(m \log n)$

Output:

The screenshot shows a code submission interface. On the left, the submission status is 'Accepted' for user 'Sidharth' submitted at Nov 21, 2024 14:41. The runtime is 0 ms (Beats 100.00%) and memory is 41.65 MB (Beats 96.05%). The right panel displays the Java code for the searchMatrix method.

```

1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         for(int [] row : matrix){
4             int low = 0;
5             int high = row.length - 1;
6             int mid = 0;
7             while (low<high){
8                 mid = (high+low)/2;
9                 if(target > row[mid]){
10                     low = mid + 1;
11                 }
12                 else if(target == row[mid]){
13                     return true;
14                 }
15                 else{
16                     high = mid - 1;
17                 }
18             }
19         }
20         return false;
21     }
22 }
23

```

### 3.Find Peak Element

```

class Solution{
    public int findPeakElement(int[] nums) {
        int N = nums.length;
        if (N == 1) {
            return 0;
        }

        int left = 0, right = N - 1;
        while (right - left > 1) {
            int mid = left + (right - left) / 2;
            if (nums[mid] < nums[mid + 1]) {
                left = mid + 1;
            }
        }
        return left;
    }
}

```

```

        } else {
            right = mid;
        }
    }

    return (left == N - 1 || nums[left] > nums[left + 1]) ? left : right;
}
}

```

Time Complexity :  $O(\log n)$

Output:

The screenshot displays a code editor interface. On the left, the 'Accepted' status is confirmed, showing a runtime of 0 ms (beats 100.00%) and memory usage of 43.00 MB (beats 7.97%). The right pane contains the following Java code:

```

1 class Solution{
2     public int findPeakElement(int[] nums) {
3         int N = nums.length;
4         if (N == 1) {
5             return 0;
6         }
7
8         int left = 0, right = N - 1;
9         while (right - left > 1) {
10             int mid = left + (right - left) / 2;
11             if (nums[mid] < nums[mid + 1]) {
12                 left = mid + 1;
13             } else {
14                 right = mid;
15             }
16         }
17         return (left == N - 1 || nums[left] > nums[left + 1]) ? left : right;
18     }
19 }
20

```

#### 4. Search in a sorted array

```

class Solution {
    public int search(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;
        while (low <= high) {
            int mid = (low+high) / 2;
            if (nums[mid] == target) {
                return mid;
            }
            else if (nums[mid] >= nums[low]) {
                if (target >= nums[low] && target < nums[mid]) {
                    high = mid - 1;
                }
            }
            else {

```



```

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

```

Time Complexity :  $O(\log n)$

Output:

The screenshot displays a code editor interface. On the left, a submission status panel indicates 'Accepted' for a submission by 'Sidharth' at Nov 22, 2024 20:25. It shows a runtime of 0 ms (Beats 100.00%) and memory usage of 42.12 MB (Beats 31.82%). Below this, a message states 'Sorry, there are not enough accepted submissions to show data'. The main editor area shows a Java solution for the 'search' method, which uses binary search to find the first and last positions of a target in a sorted array. The code is as follows:

```

class Solution {
    public int search(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;
        while (low <= high){
            int mid = (low+high) / 2;
            if(nums[mid] == target){
                return mid;
            }
            else if(nums[mid] >= nums[low]){
                if(target >= nums[low] && target < nums[mid]){
                    high = mid -1;
                }
                else{
                    low = mid + 1;
                }
            }
            else{
                if(nums[high] >= target && nums[mid] < target){
                    low = mid + 1;
                }
                else{
                    high = mid -1;
                }
            }
        }
        return -1;
    }
}

```

## 5. Find First and Last Position of Element in Sorted Array

```

class Solution {
    public int[] searchRange(int[] nums, int target) {
        int start = binarysearch(nums, target, true);
        int end = binarysearch(nums, target, false);

        if (start != -1) {
            return new int[]{start, end};
        } else {
            return new int[]{-1, -1};
        }
    }
}

```

```

private int binarysearch(int[] nums, int target, boolean check) {
    int left = 0;
    int right = nums.length - 1;
    int idx = -1;

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

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

    return idx;
}

```

Time Complexity :  $O(\log n)$

Output:

**34. Find First and Last Position of Element in Sorted Array** Solved

Medium Topics Companies

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]`

```

1 class Solution {
2     public int[] searchRange(int[] nums, int target) {
3         int start = binarysearch(nums, target, true);
4         int end = binarysearch(nums, target, false);
5
6         if (start != -1) {
7             return new int[]{start, end};
8         } else {
9             return new int[]{-1, -1};
10        }
11    }
12
13    private int binarysearch(int[] nums, int target, boolean check) {
14        int left = 0;
15        int right = nums.length - 1;
16        int idx = -1;
17
18        while (left <= right) {
19            int mid = (left + right) / 2;
20
21            if (nums[mid] < target) {
22                left = mid + 1;
23            } else if (nums[mid] > target) {
24                right = mid - 1;
25            } else {
26                idx = mid;
27                if (check) {
28                    right = mid - 1;
29                } else {
30                    left = mid + 1;
31                }
32            }
33        }
34
35        return idx;
36    }
37 }

```

6. Find Minimum in Rotated Sorted Array

```

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

```

Time Complexity :  $O(\log n)$

Output:

