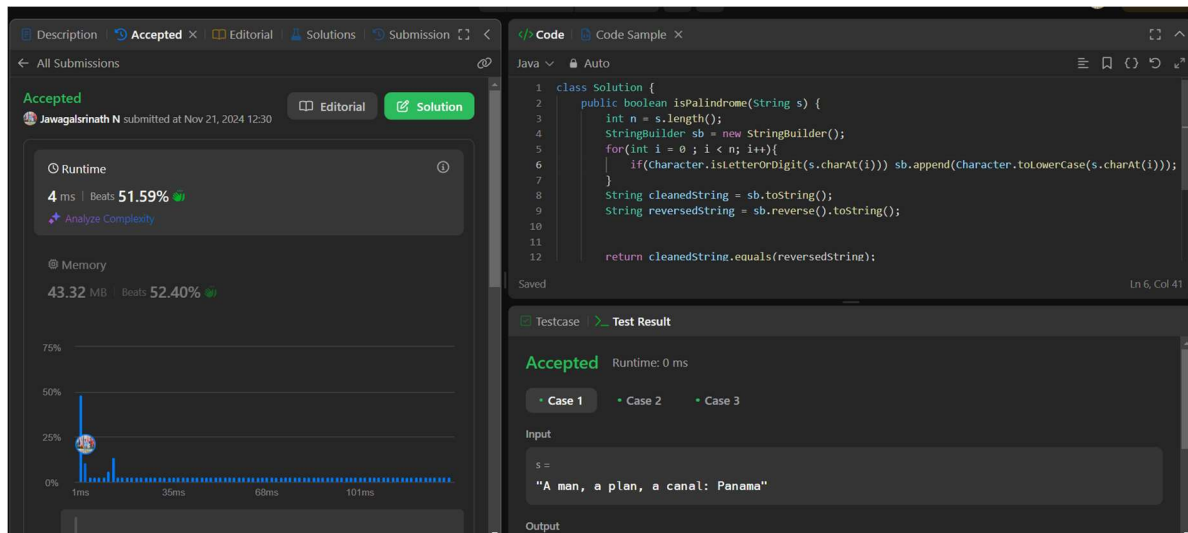


DSA PRACTICE

November 21 , 2024

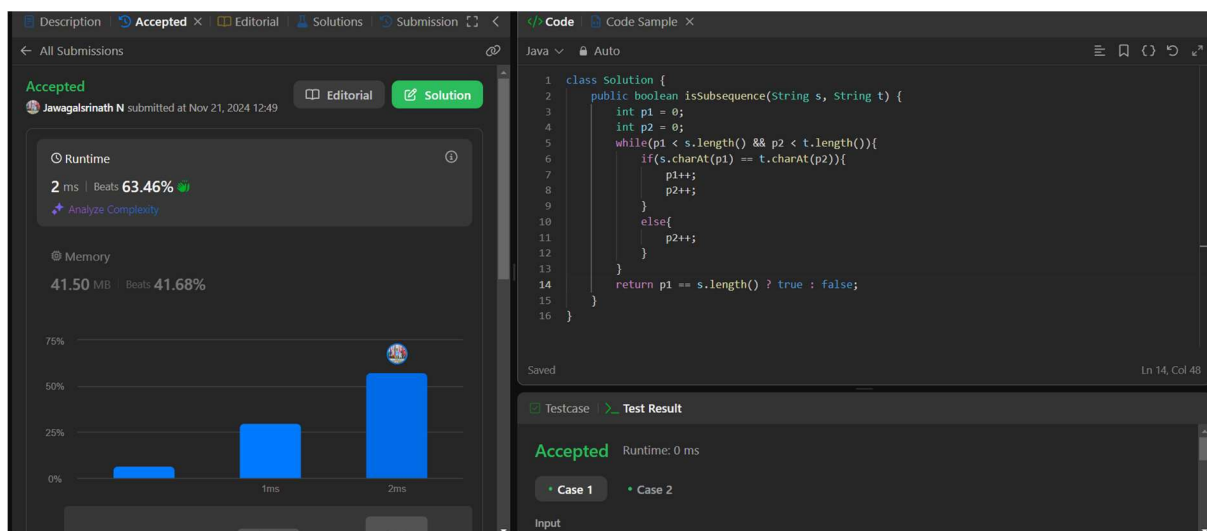
1. Valid Palindrome:



Time Complexity : $O(N)$

Space Complexity : $O(1)$

2. In Subsequence :



Time Complexity : $O(N)$

Space Complexity : $O(1)$

3. Two Sum II

The screenshot displays a LeetCode submission for the 'Two Sum II' problem. On the left, the 'Runtime' section shows a time of 2 ms, beating 93.14% of solutions, and a memory usage of 46.94 MB, beating 67.41%. A bar chart below these metrics shows the submission's performance relative to others. The 'Code' section on the right contains the following Java code:

```
1 class Solution {
2     public int[] twoSum(int[] numbers, int target) {
3         int p1 = 0, p2 = numbers.length-1;
4         int[] ans = new int[2];
5         while(p1 < p2){
6             if((numbers[p1] + numbers[p2]) == target){
7                 ans[0] = p1+1;
8                 ans[1] = p2+1;
9                 break;
10            }
11            else if((numbers[p1] + numbers[p2]) < target) p1++;
12            else p2--;
13        }
14        return ans;
15    }
16 }
```

The 'Test Result' section shows the submission is 'Accepted' with a runtime of 0 ms. The input for the test case is:

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

Time complexity : $O(N)$

Space Complexity : $O(1)$

4. Maximum Water Container:

The screenshot displays a LeetCode submission for the 'Maximum Water Container' problem. On the left, the 'Runtime' section shows a time of 5 ms, beating 74.09% of solutions, and a memory usage of 57.95 MB, beating 42.14%. A bar chart below these metrics shows the submission's performance relative to others. The 'Code' section on the right contains the following Java code:

```
1 class Solution {
2     public int maxArea(int[] height) {
3         int max = 0, p1 = 0, p2 = height.length-1;
4         while(p1 < p2){
5             int area = Math.min(height[p1],height[p2]) * (p2-p1);
6             max = Math.max(area, max);
7             if(height[p1] < height[p2]) p1++;
8             else p2--;
9         }
10        return max;
11    }
12 }
```

The 'Test Result' section shows the submission is 'Accepted' with a runtime of 0 ms. The input for the test case is:

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

Time Complexity : $O(N)$

Space Complexity : $O(1)$

5. 3 Sum:

The screenshot shows a C++ solution for the 3Sum problem. The left panel displays the submission status as 'Accepted' with a runtime of 79 ms (Beats 28.38%) and memory usage of 37.80 MB (Beats 22.76%). The right panel shows the C++ code, which sorts the array and uses a two-pointer technique to find all unique triplets that sum to zero. The code is as follows:

```
1 class Solution {
2 public:
3     vector<vector<int>> threeSum(vector<int>& nums) {
4         int n=nums.size(); sort(nums.begin(), nums.end());
5         vector<vector<int>> res; set<vector<int>> ref;
6         int l=0, r=n-1, temp;
7         for (int i=0; i<n; i++){
8             if (nums[i]>0) break;
9             if(i > 0 && nums[i] == nums[i - 1]) continue;
10            l=i+1; r=n-1;
11            while (l<r){
12                temp = nums[i]+nums[l]+nums[r];
13                if (temp==0) {ref.insert((nums[i],nums[l],nums[r]));l++;r--;}
14                else if (temp<0) l++;
15                else r--;
16            }
17        }
18        for (auto i: ref) res.push_back(i);
19        return res;
20    }
21};
```

Time Complexity : $O(N^2)$

Space Complexity : $O(N)$

6. Minimum Size Subarray Sum :

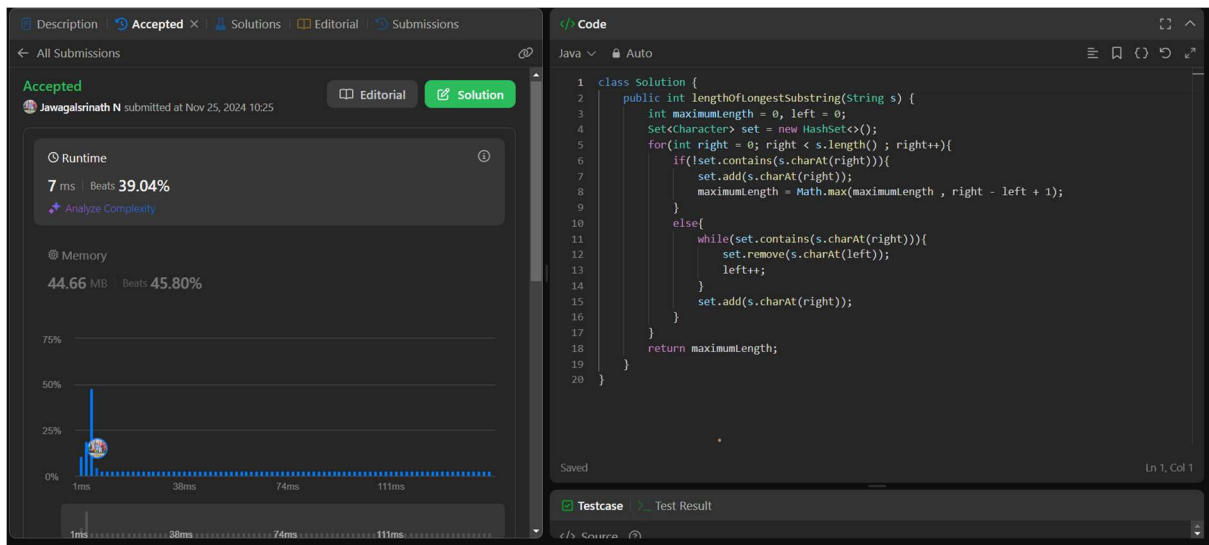
The screenshot shows a Java solution for the Minimum Size Subarray Sum problem. The left panel displays the submission status as 'Accepted' with a runtime of 1 ms (Beats 99.95%) and memory usage of 58.12 MB (Beats 32.24%). The right panel shows the Java code, which uses a sliding window approach to find the minimum length of a subarray that sums to at least the target. The code is as follows:

```
1 class Solution {
2     public int minSubArrayLen(int target, int[] nums) {
3         int left = 0;
4         int sum = 0;
5         int minLength = Integer.MAX_VALUE;
6         for(int right = 0; right < nums.length; right++){
7             sum += nums[right];
8             while(sum >= target){
9                 minLength = Math.min(minLength, (right - left)+1);
10                sum -= nums[left];
11                left++;
12            }
13        }
14        return minLength != Integer.MAX_VALUE ? minLength : 0;
15    }
16 }
17 }
```

Time Complexity : $O(N)$

Space Complexity : $O(1)$

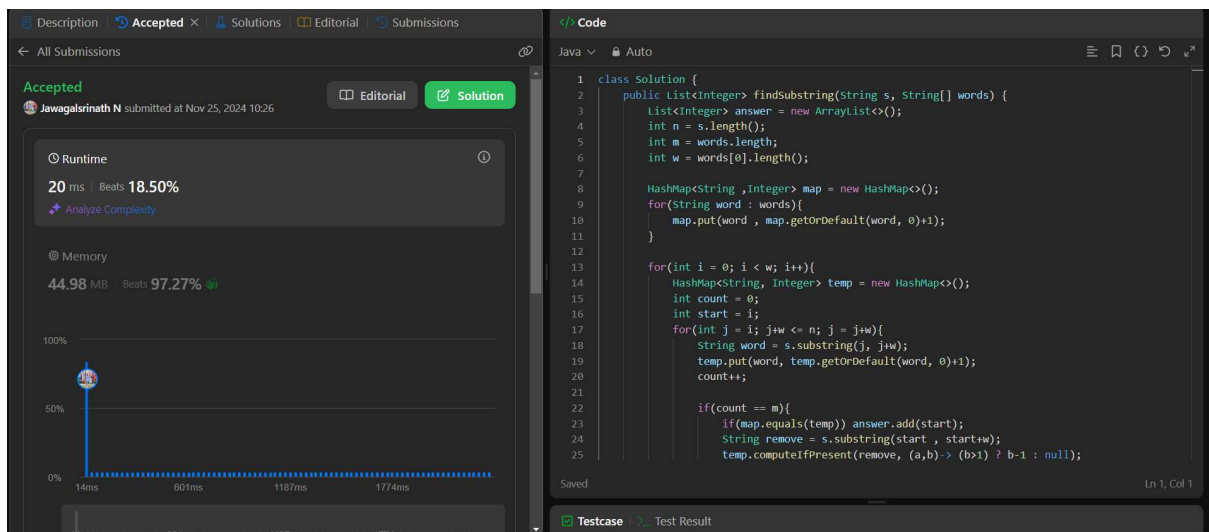
7. Longest Subarray , without repeating characters



Time Complexity : $O(N)$

Space Complexity : $O(1)$

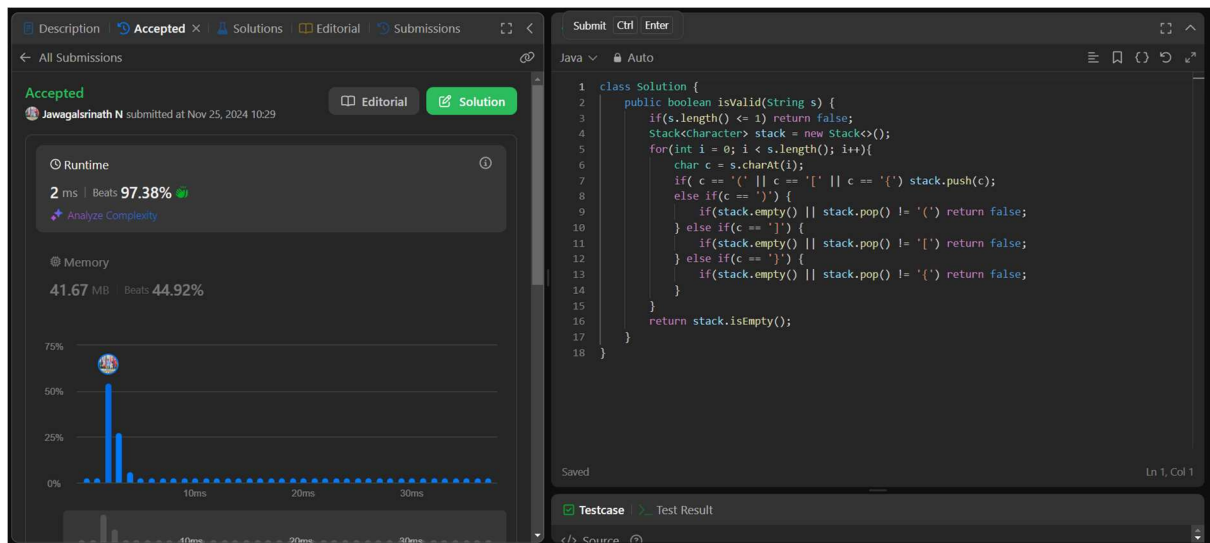
8. Substring with Concatenation of All Words



Time Complexity : $O(N+M)$

Space Complexity : $O(N)$

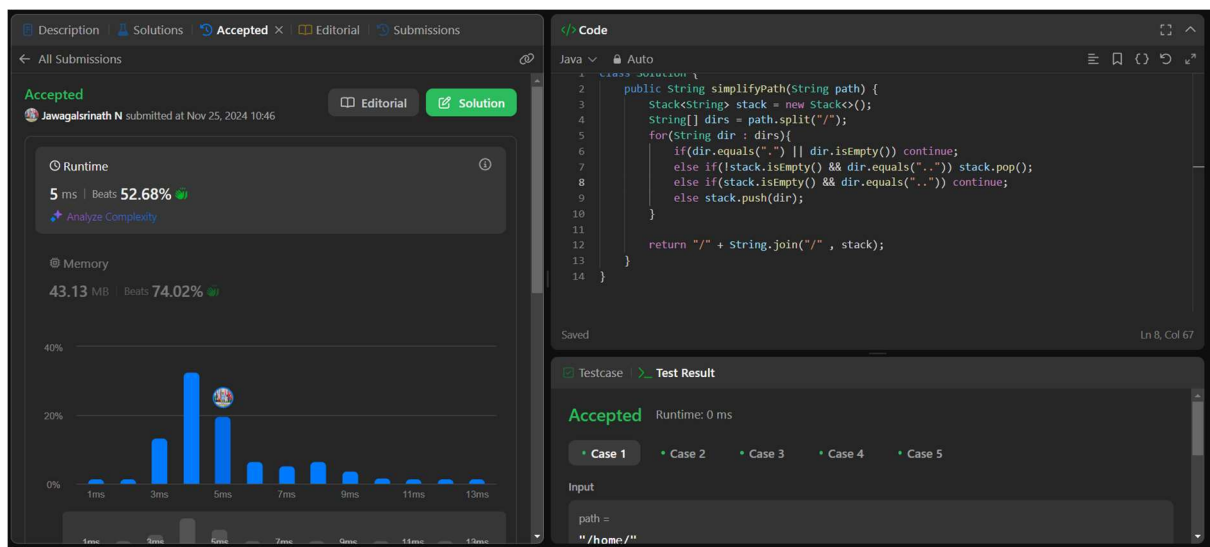
9. Valid Parenthesis



Time Complexity : $O(N)$

Space Complexity : $O(1)$

10. Simplify Path



Time Complexity : $O(N)$

Space Complexity : $O(N)$

11. Min Stack

The screenshot shows a LeetCode submission for the 'Min Stack' problem. The left panel displays the submission status as 'Accepted' with a runtime of 4 ms (beats 96.63%) and memory usage of 44.90 MB (beats 47.19%). A bar chart shows the runtime distribution. The right panel displays the Java code for the Min Stack implementation.

```
Java  
Auto  
8  
9  
10 public void push(int val) {  
11     stack.add(val);  
12     if(minArray.isEmpty() || val <= minArray.get(minArray.size() - 1)){  
13         minArray.add(val);  
14     }  
15 }  
16  
17 public void pop() {  
18     if(!stack.isEmpty()){  
19         int res = stack.remove(stack.size() - 1);  
20         if(res == minArray.get(minArray.size()-1)) minArray.remove(minArray.size()-1);  
21     }  
22 }  
23  
24 public int top() {  
25     return stack.get(stack.size()-1);  
26 }  
27  
28 public int getMin() {  
29     return minArray.get(minArray.size()-1);  
30 }  
31 }  
32  
Saved  
Ln 15, Col 6  
Testcase Test Result
```

Time Complexity : $O(1)$

Space Complexity : $O(N)$

12. Evaluate reverse polish notation

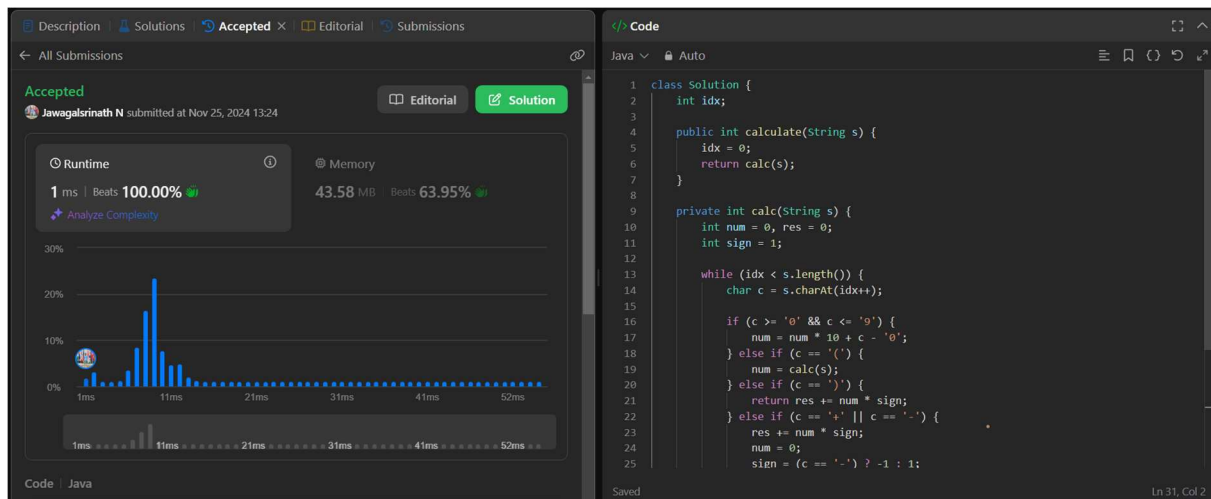
The screenshot shows a LeetCode submission for the 'Evaluate reverse polish notation' problem. The left panel displays the submission status as 'Accepted' with a runtime of 6 ms (beats 75.52%) and memory usage of 44.18 MB (beats 95.21%). A bar chart shows the runtime distribution. The right panel displays the Java code for the evaluation.

```
Java  
Auto  
1 class Solution {  
2     public int evalRPN(String[] tokens) {  
3         Stack<Integer> stack = new Stack<>();  
4         int v1,v2;  
5         for(String token:tokens){  
6             switch(token){  
7                 case "+":  
8                     v1 = stack.pop();  
9                     v2 = stack.pop();  
10                    stack.push(v1+v2);  
11                    break;  
12                    case "-":  
13                        v1 = stack.pop();  
14                        v2 = stack.pop();  
15                        stack.push(v2-v1);  
16                        break;  
17                        case "*":  
18                            v1 = stack.pop();  
19                            v2 = stack.pop();  
20                            stack.push(v2*v1);  
21                            break;  
22                            case "/":  
23                                v1 = stack.pop();  
24                                v2 = stack.pop();  
25                                stack.push(v2/v1);  
26                                break;  
27                            default:  
28                                stack.push(Integer.parseInt(token));  
29                            }  
30            }  
31        }  
32        return stack.pop();  
33    }  
34 }  
Saved  
Ln 34, Col 2  
Testcase Test Result  
Accepted Runtime: 0 ms  
Case 1 Case 2 Case 3
```

Time Complexity : $O(N)$

Space Complexity : $O(N)$

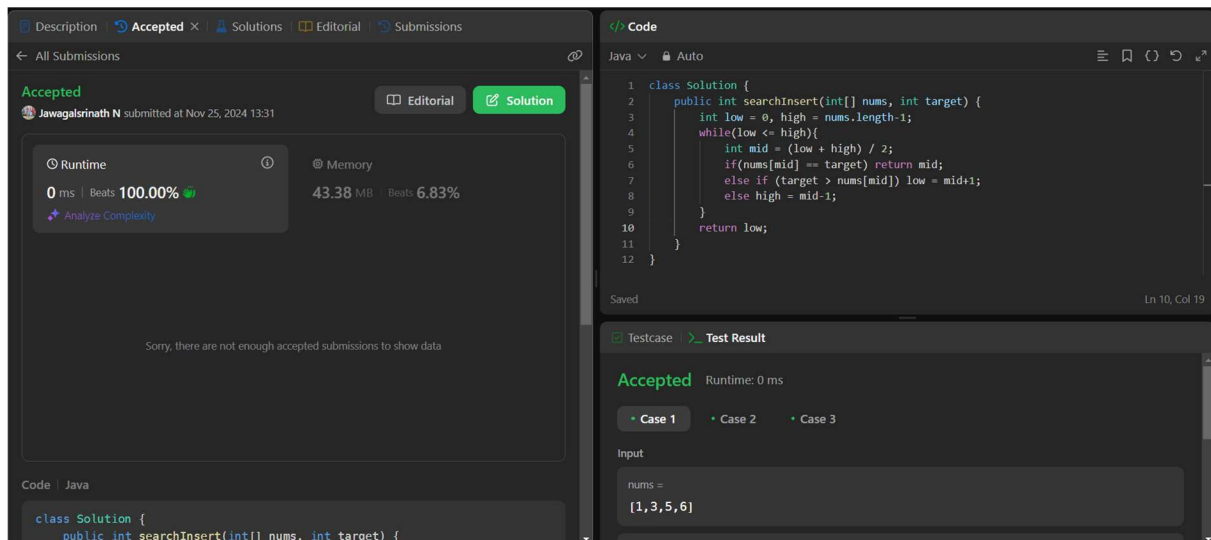
13. Basic Calculator



Time Complexity : $O(N)$

Space Complexity : $O(1)$

14. Search insert position



Time Complexity : $O(\log n)$

Space Complexity : $O(1)$

15. Search in 2D matrix

The screenshot shows a LeetCode submission interface for the problem 'Search in 2D matrix'. The submission is accepted, with a runtime of 0 ms (beats 100.00%) and memory usage of 42.04 MB (beats 54.08%). The code is written in Java and implements a binary search algorithm. The test case shows a matrix of [[1,3,5,7],[10,11,16,20],[23,30,34,60]] and a target of 3.

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

Time Complexity : $O(\log(N+M))$

Space Complexity : $O(1)$

16. Find the peak element

The screenshot shows a LeetCode submission interface for the problem 'Find the peak element'. The submission is accepted, with a runtime of 0 ms (beats 100.00%) and memory usage of 42.28 MB (beats 59.60%). The code is written in Java and implements a binary search algorithm. The test case shows an input array of [1,2,3,1].

```
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0, 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 : $O(\log N)$

Space Complexity : $O(1)$

17. Search in rotated sorted array

The screenshot shows a LeetCode submission interface. On the left, the 'Accepted' status is confirmed for user 'Jawagalsrinath N' at 14:55. Performance metrics show 0 ms runtime (100.00% beats) and 42.11 MB memory (31.79% beats). The code editor on the right contains a Java solution for the 'search' method, which uses a binary search approach to find the target in a rotated 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;

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

Time Complexity : $O(\log N)$

Space Complexity : $O(1)$

18. Find the first and last index of target

The screenshot shows a LeetCode submission interface for the 'Find the first and last index of target' problem. The submission is 'Accepted' by 'Jawagalsrinath N' at 19:15. Performance metrics show 0 ms runtime (100.00% beats) and 45.59 MB memory (92.21% beats). A blue box highlights the user's profile icon in the performance comparison chart. The code editor on the right contains a Java solution for the 'searchRange' method, which uses binary search to find the first and last indices of the target. The code is as follows:

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] ans = {-1, -1};
        int low = 0, high = nums.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (nums[mid] == target) {
                ans[0] = mid;
                ans[1] = mid;
                while (ans[0] > 0 && nums[ans[0] - 1] == target) {
                    ans[0]--;
                }
                while (ans[1] < nums.length - 1 && nums[ans[1] + 1] == target) {
                    ans[1]++;
                }
                return ans;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
}
```

Time complexity : $O(\log n)$

Space complexity : $O(1)$

19. Minimum in sorted rotated array

The screenshot shows a LeetCode submission interface. On the left, the 'Accepted' status is confirmed, with a submission by 'Jawagalsrinath N' from Nov 25, 2024. Performance metrics show 0 ms runtime (100.00% beats) and 41.82 MB memory (61.02% beats). A bar chart visualizes the runtime performance. On the right, the Java code is displayed, implementing a binary search algorithm to find the minimum element in a sorted rotated array.

Runtime: 0 ms | Beats 100.00%
Memory: 41.82 MB | Beats 61.02%

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0, 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 : $O(\log N)$

Space Complexity : $O(1)$
