

## DSA Practice – 9

## 1. Valid Palindrome

```
class Solution {
    public boolean isPalindrome(String s) {
        StringBuilder filteredString = new StringBuilder();
        for (char c : s.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                filteredString.append(Character.toLowerCase(c));
            }
        }
        int left = 0;
        int right = filteredString.length() - 1;
        while (left < right) {
            if (filteredString.charAt(left) != filteredString.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

The screenshot displays the LeetCode interface for problem 125, "Valid Palindrome". The problem statement asks if a string is a palindrome after ignoring non-alphanumeric characters and case. A Java solution is provided, which uses a two-pointer approach with `StringBuilder` and `Character` methods to compare characters from both ends of the string.

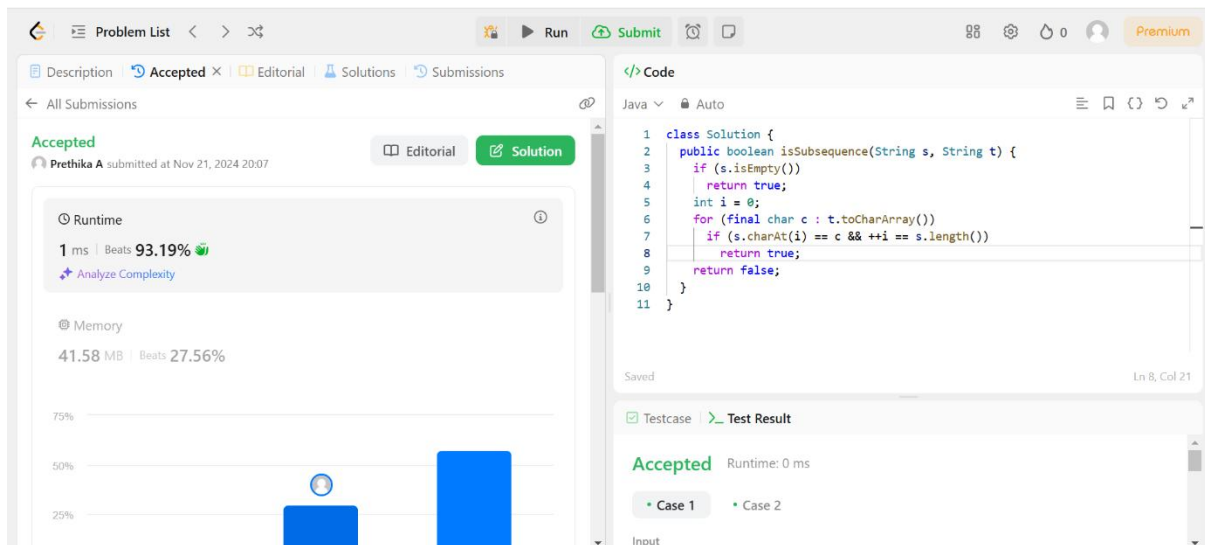
```

1 class Solution {
2     public boolean isPalindrome(String s) {
3         StringBuilder filteredString = new StringBuilder();
4         for (char c : s.toCharArray()) {
5             if (Character.isLetterOrDigit(c)) {
6                 filteredString.append(Character.toLowerCase(c));
7             }
8         }
9         int left = 0;
10        int right = filteredString.length() - 1;
11        while (left < right) {
12            if (filteredString.charAt(left) != filteredString.charAt(right)) {
13                return false;
14            }
15            left++;
16            right--;
17        }
18        return true;
19    }
20 }

```

The runtime analysis shows 3 ms execution time, beating 72.13% of submissions, and 43.33 MB memory usage, beating 52.40% of submissions. The code is marked as "Accepted".

## 2.Is Subsequence

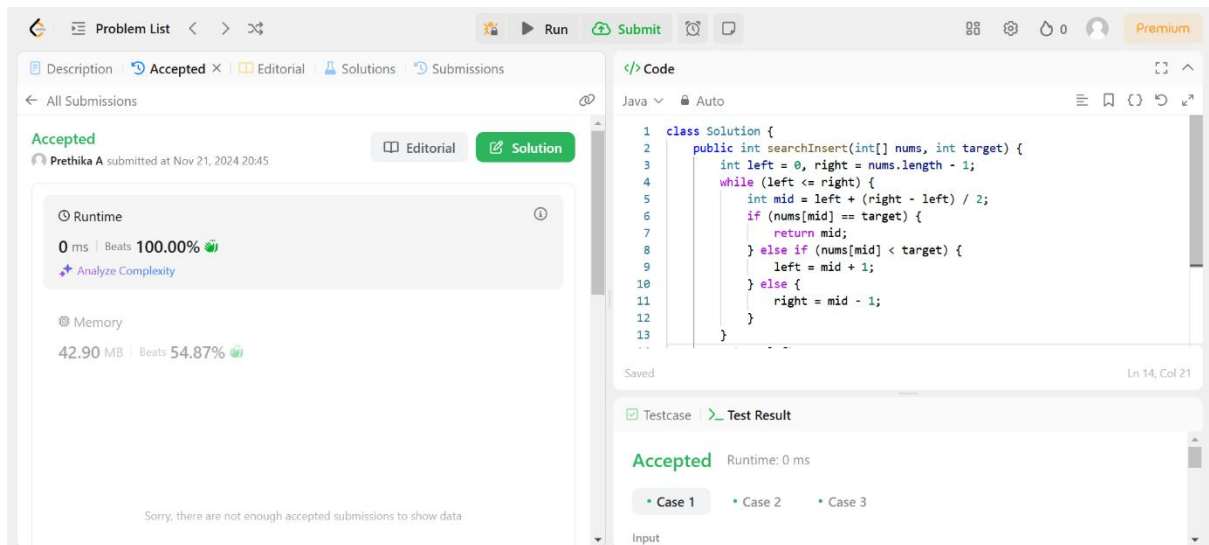


The screenshot displays a coding platform interface for the 'Is Subsequence' problem. The left panel shows the submission status as 'Accepted' with a runtime of 1 ms, beating 93.19% of other solutions. The right panel shows the Java code for the solution, which uses a two-pointer approach to check if string 's' is a subsequence of string 't'.

```
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;
    }
}
```

## 3.Search Insert Position

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



## 4.Longest Substring without repeating characters

```

class Solution {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> seen = new HashSet<>();
        int maxLength = 0;
        int left = 0;
        for (int right = 0; right < s.length(); right++) {
            char currentChar = s.charAt(right);
            while (seen.contains(currentChar)) {
                seen.remove(s.charAt(left));
                left++;
            }
            seen.add(currentChar);
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}

```

The screenshot shows a LeetCode submission for the problem "Length of Longest Substring". The submission is "Accepted" and was made by "Prethika A" on Nov 21, 2024 at 20:53. The runtime is 6 ms, which beats 70.21% of other submissions. The memory usage is 44.92 MB, which beats 20.66%. The code is written in Java and implements a sliding window algorithm using a HashSet to track characters in the current window.

```

1 class Solution {
2     public int lengthOfLongestSubstring(String s) {
3         Set<Character> seen = new HashSet<>();
4         int maxLength = 0;
5         int left = 0;
6         for (int right = 0; right < s.length(); right++) {
7             char currentChar = s.charAt(right);
8             while (seen.contains(currentChar)) {
9                 seen.remove(s.charAt(left));
10                left++;
11            }
12            seen.add(currentChar);
13            maxLength = Math.max(maxLength, right - left + 1);
14        }
15        return maxLength;
16    }
17 }

```

## 5. 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};
    }
}

```

Problem List

Accepted

Editorial

Solutions

Submissions

All Submissions

Accepted

Prethika A submitted at Nov 21, 2024 21:23

Editorial

Solution

Runtime

2 ms | Beats 93.14%

Analyze Complexity

Memory

46.65 MB | Beats 90.71%

100%

50%

</> Code

Java

Auto

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

Saved

Ln 18, Col 2

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input