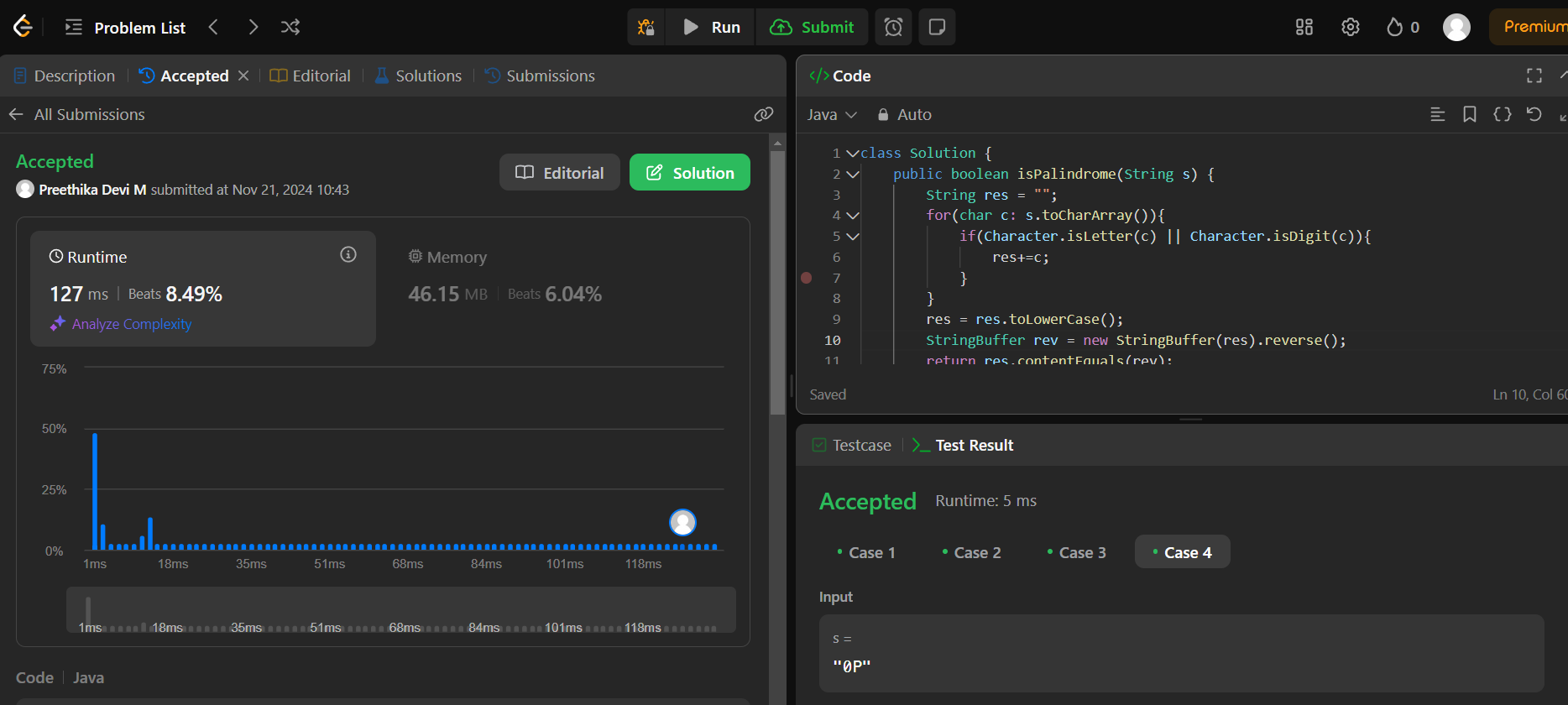
**CODING PRACTICE (21/11/2024)**

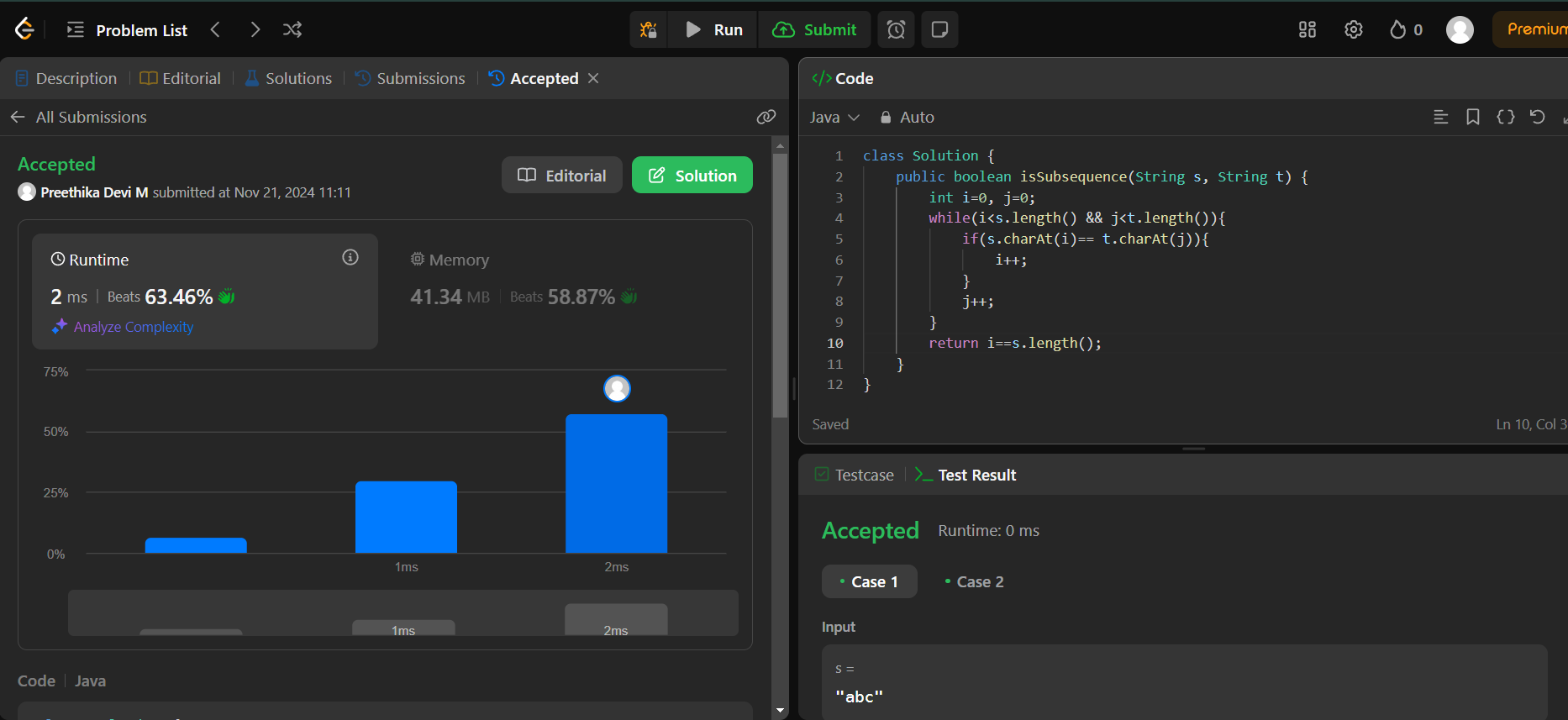
1. **Valid Palindrome:**



Time Complexity : O(N)

Space Complexity : O(N)

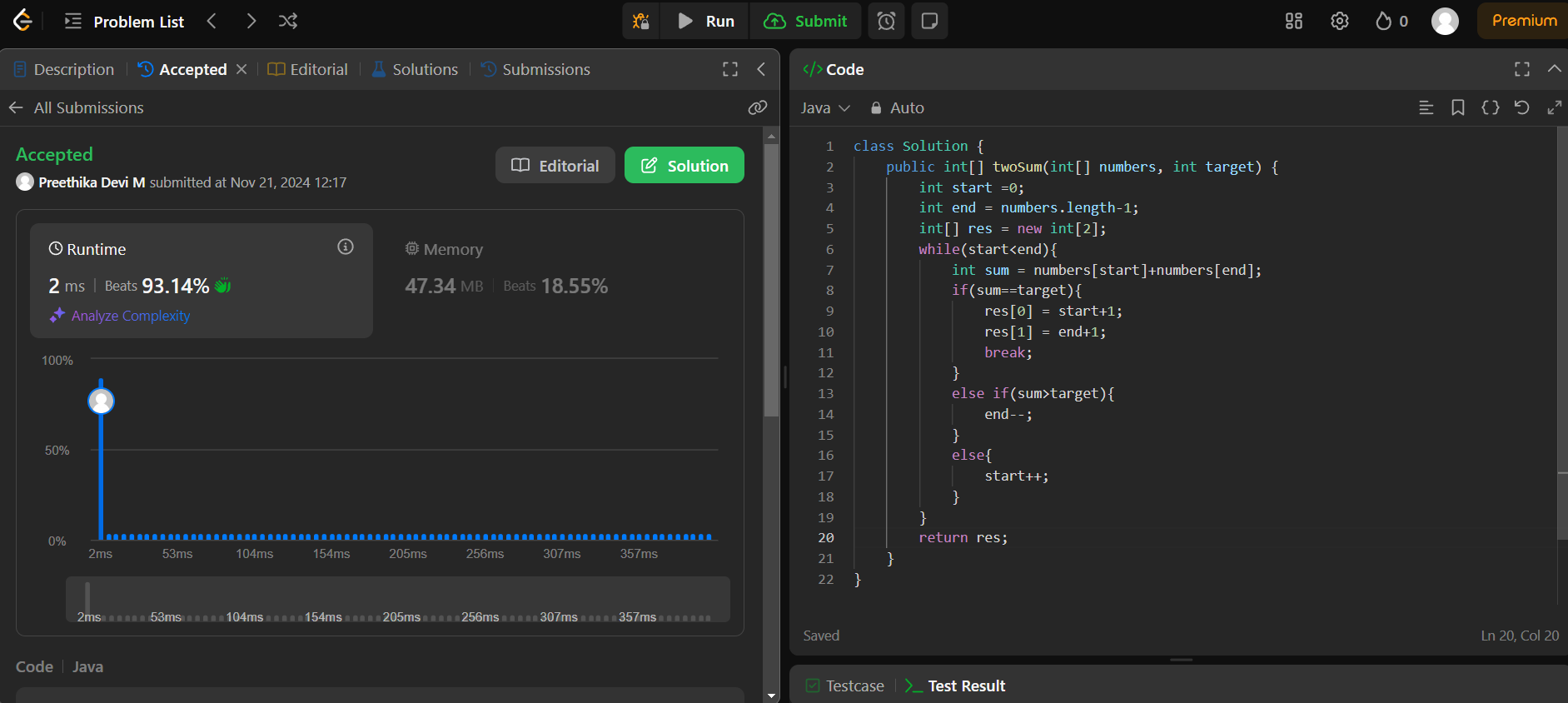
1. **IS SUBSEQUENCE:**



Time Complexity : O(logN)

Space Complexity : O(1)

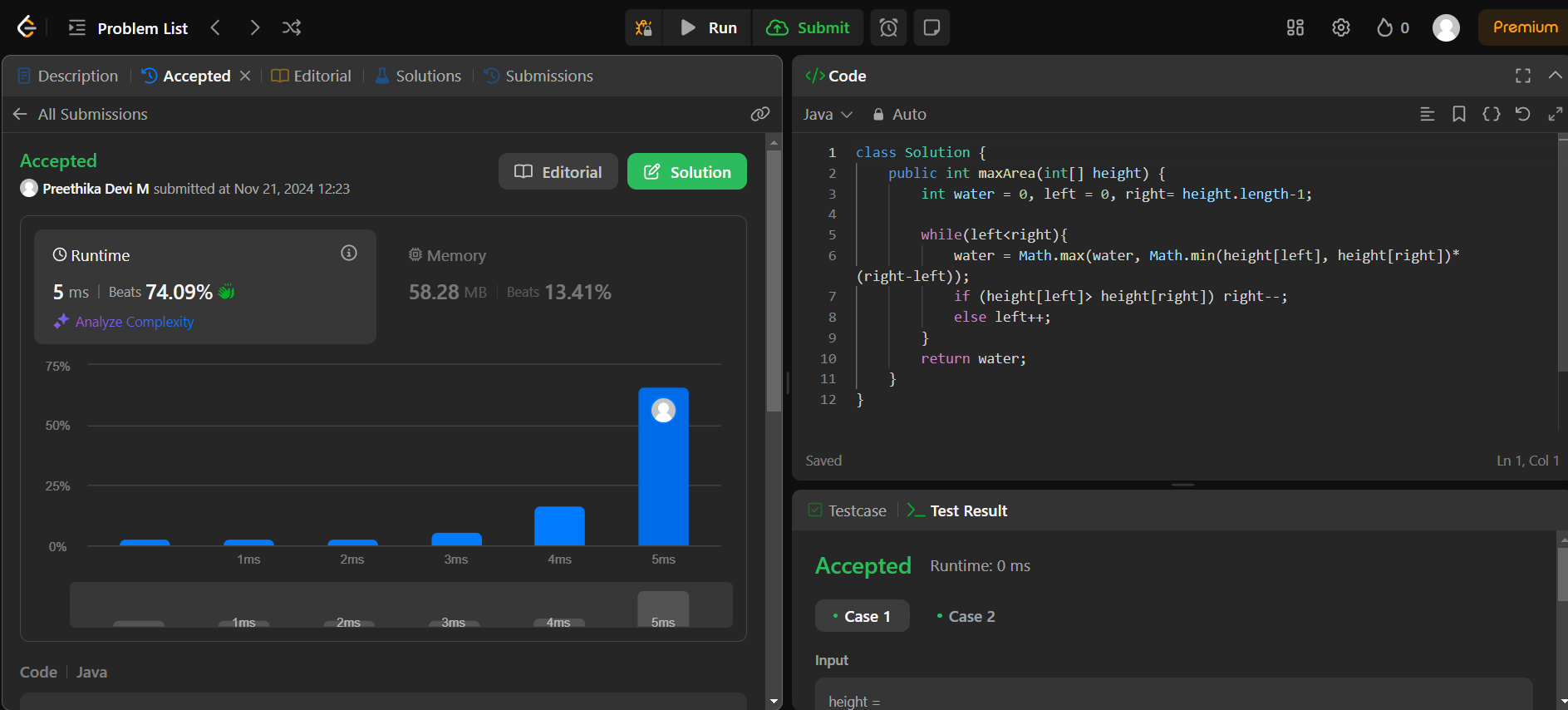
1. TWO SUM – Input array is sorted:



Time complexity : O(logN)

Space Complexity : O(1)

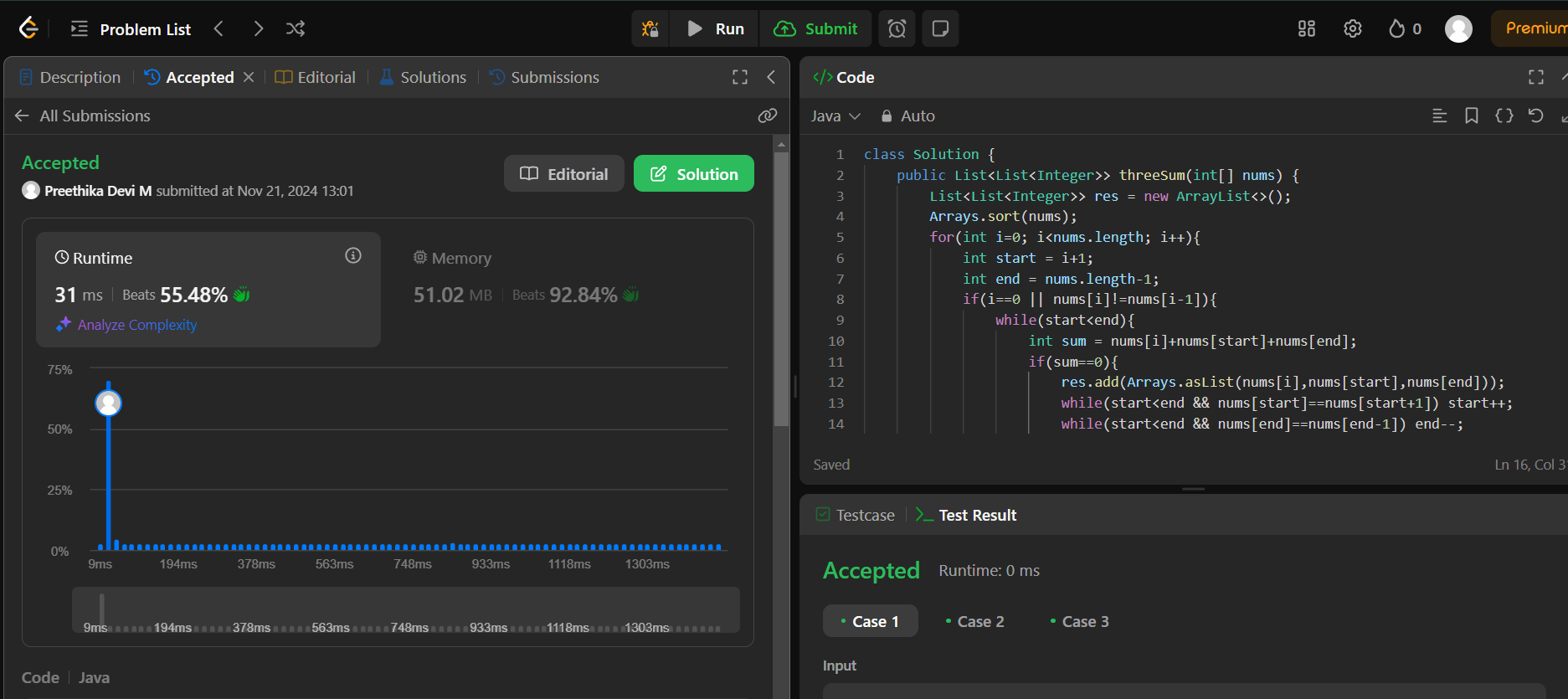
1. Container with Most Water:



Time Compleity : O(logN)

Space Complexity : O(1)

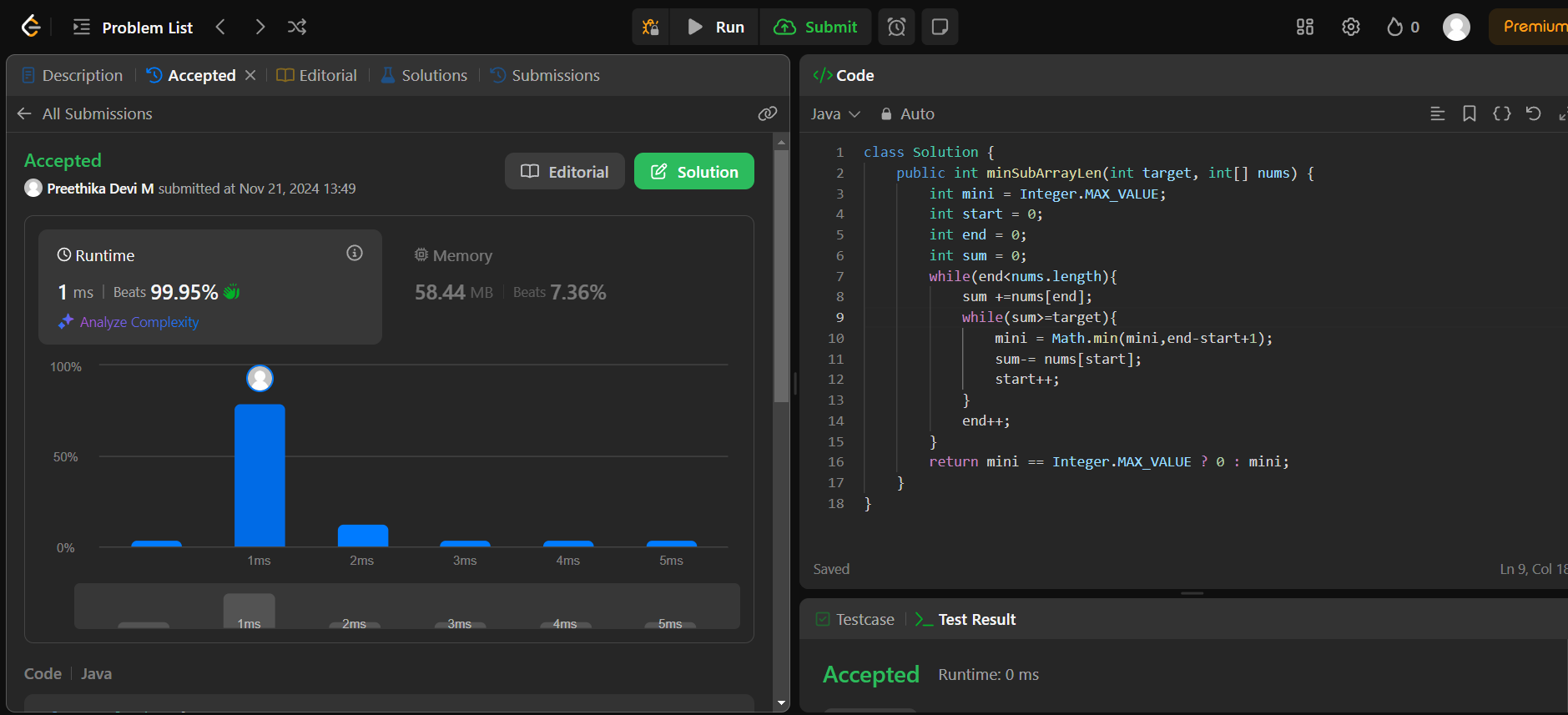
1. 3SUM:



Time Complexity : O(N^2)

Space Complexity : O(N^2)

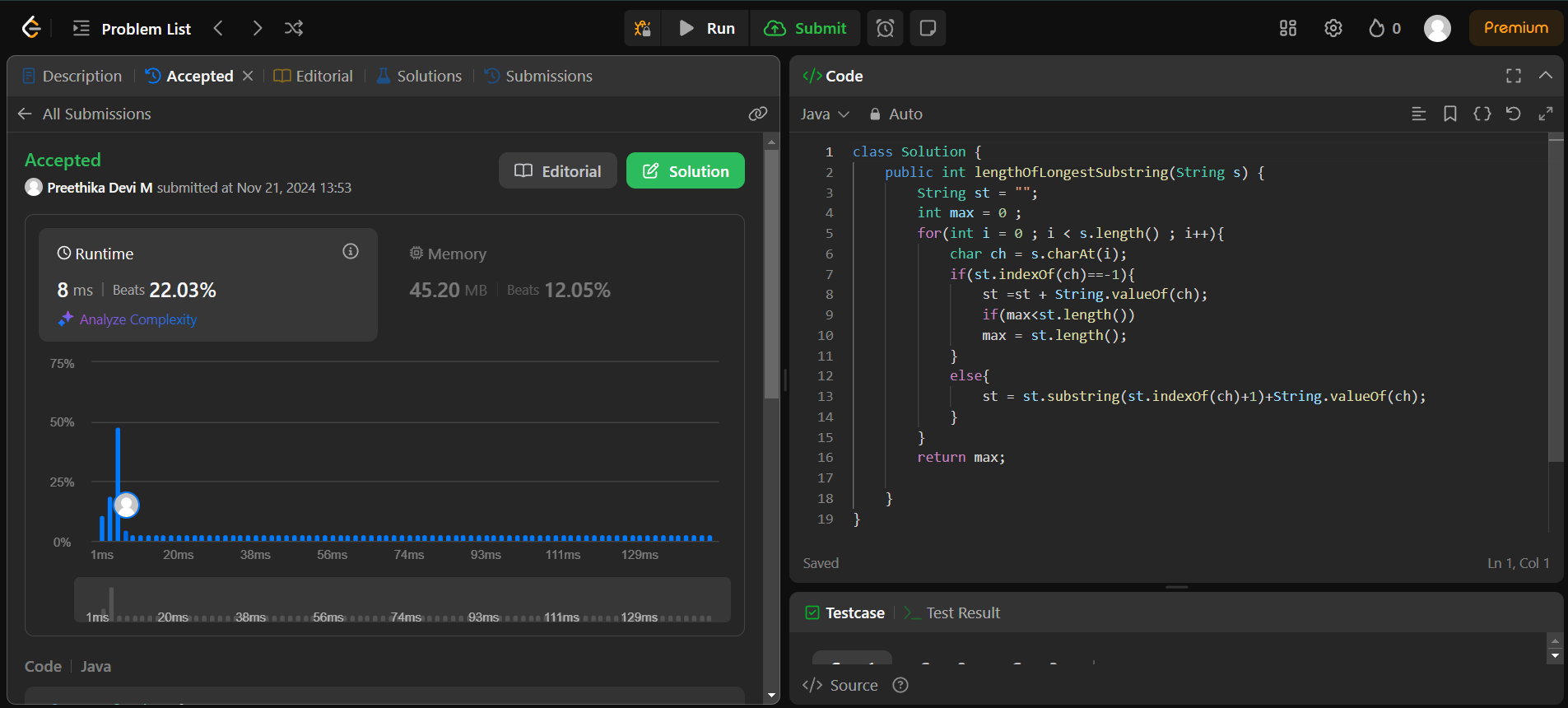
1. **Minimum Size Subarray Sum:**



**Time Complexity : O(N)**

**Space Complexity : O(1)**

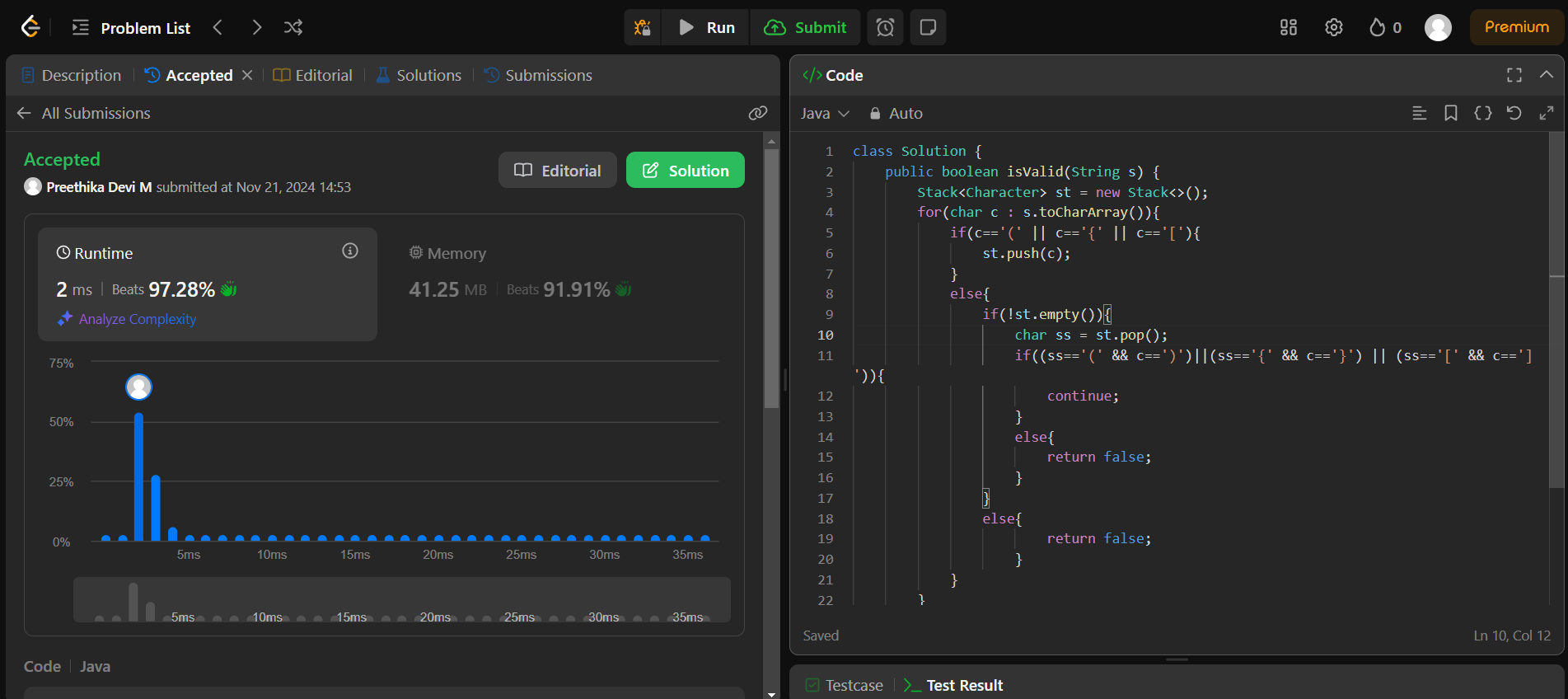
1. **Longest Substring Without Repeating Characters:**

****

**Time Complexity : O(N)**

**Space Complexity : O(1)**

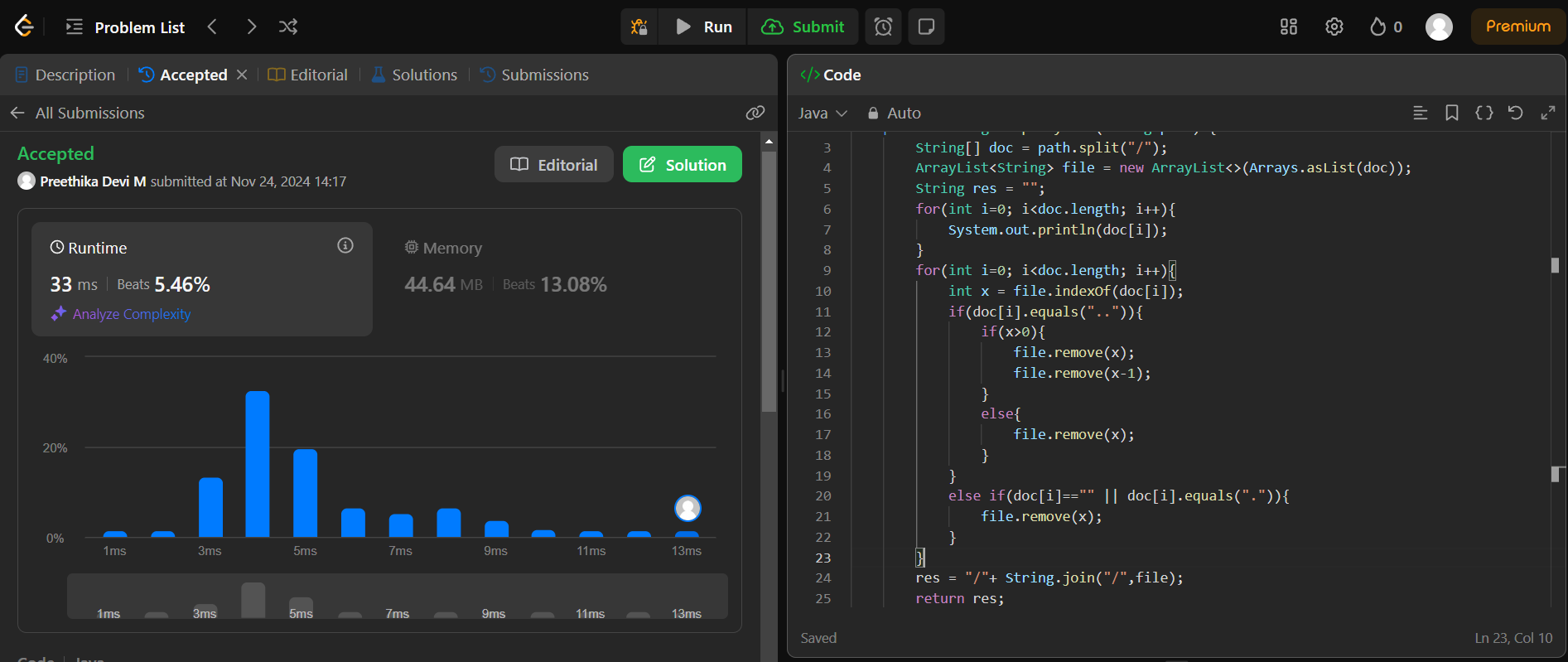
1. **Valid Parentheses:**

****

**Time Complexity : O(N)**

**Space Complexity : O(1)**

1. **Simplify Path:**

****

**Time Complexity : O(N)**

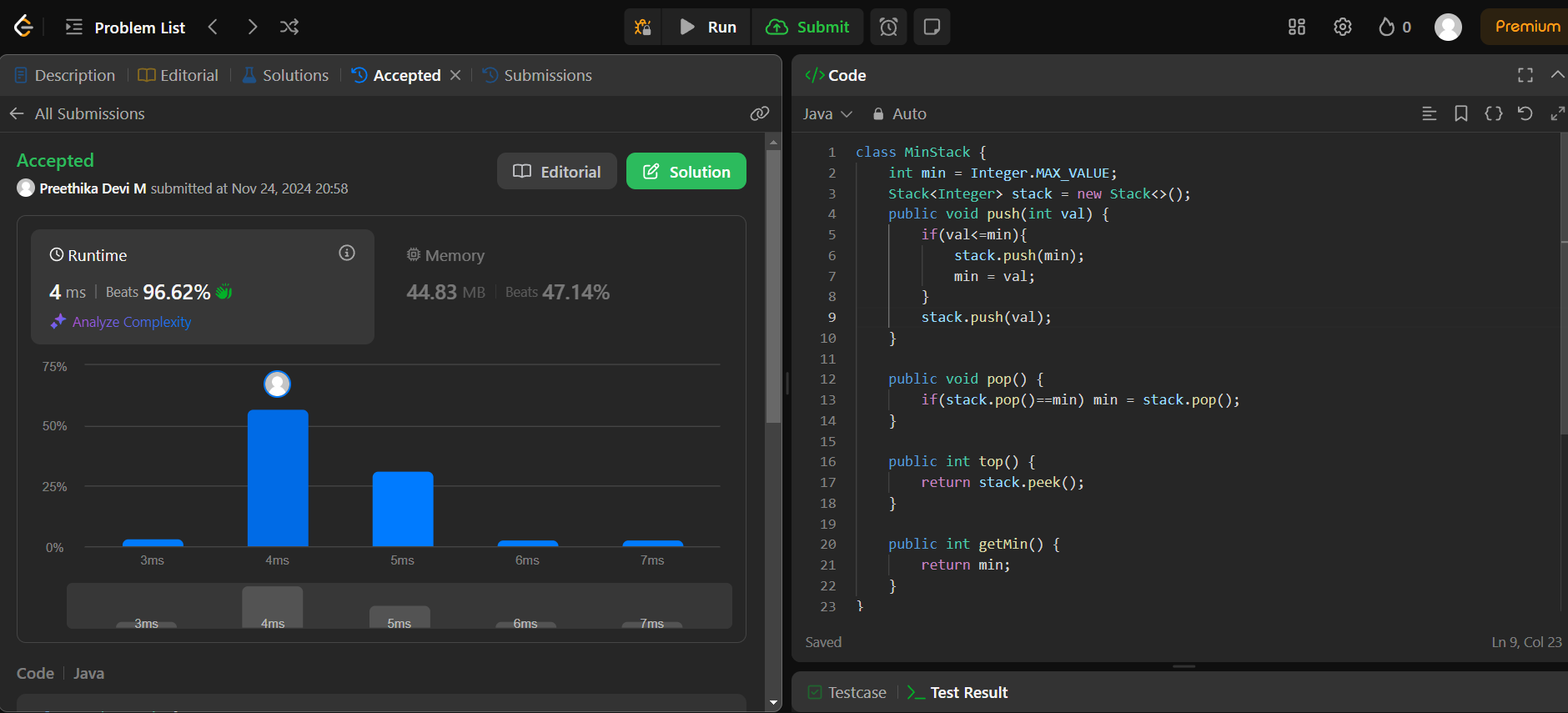
**Space Complexity : O(N)**

1. **Substring with concatenation with all words:**
2. public class Solution {
3. public List<Integer> findSubstring(String s, String[] words) {
4. List<Integer> result = new ArrayList<>();
5. if (s.length() == 0 || words.length == 0)
6. return result;
7. int wordLength = words[0].length();
8. int totalWords = words.length;
9. int substringLength = wordLength \* totalWords;
10. if (s.length() < substringLength)
11. return result;
12. Map<String, Integer> wordCount = new HashMap<>();
13. for (String word : words) {
14. wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
15. }
16. for (int i = 0; i < wordLength; i++) {
17. Map<String, Integer> currentWordCount = new HashMap<>();
18. int wordsFound = 0;
19. int left = i;
20. for (int j = i; j <= s.length() - wordLength; j += wordLength) {
21. String currentWord = s.substring(j, j + wordLength);
22. if (wordCount.containsKey(currentWord)) {
23. currentWordCount.put(currentWord, currentWordCount.getOrDefault(currentWord, 0) + 1);
24. wordsFound++;
25. while (currentWordCount.get(currentWord) > wordCount.get(currentWord)) {
26. String leftWord = s.substring(left, left + wordLength);
27. currentWordCount.put(leftWord, currentWordCount.get(leftWord) - 1);
28. wordsFound--;
29. left += wordLength;
30. }
31. if (wordsFound == totalWords) {
32. result.add(left);
33. String leftWord = s.substring(left, left + wordLength);
34. currentWordCount.put(leftWord, currentWordCount.get(leftWord) - 1);
35. wordsFound--;
36. left += wordLength;
37. }
38. }
39. else {
40. currentWordCount.clear();
41. wordsFound = 0;
42. left = j + wordLength;
43. }
44. }
45. }
46. return result;
47. }
48. }

**Time complexity : O(N^2)**

**Space complexity: O(N^2)**

**11. Min Stack:**

****

**12. Evaluate reverse polish Notation:**

