Discover. Learn. Empower. ENGINEERING

Average Learner Complex Problems

Student Name: Devansh Raj

UID: 22BCS16933

Branch: CSE

Section/Group: NTPP-603-B

Semester: 6th

Date of Performance: 09/04/25

Subject Name: AP-2

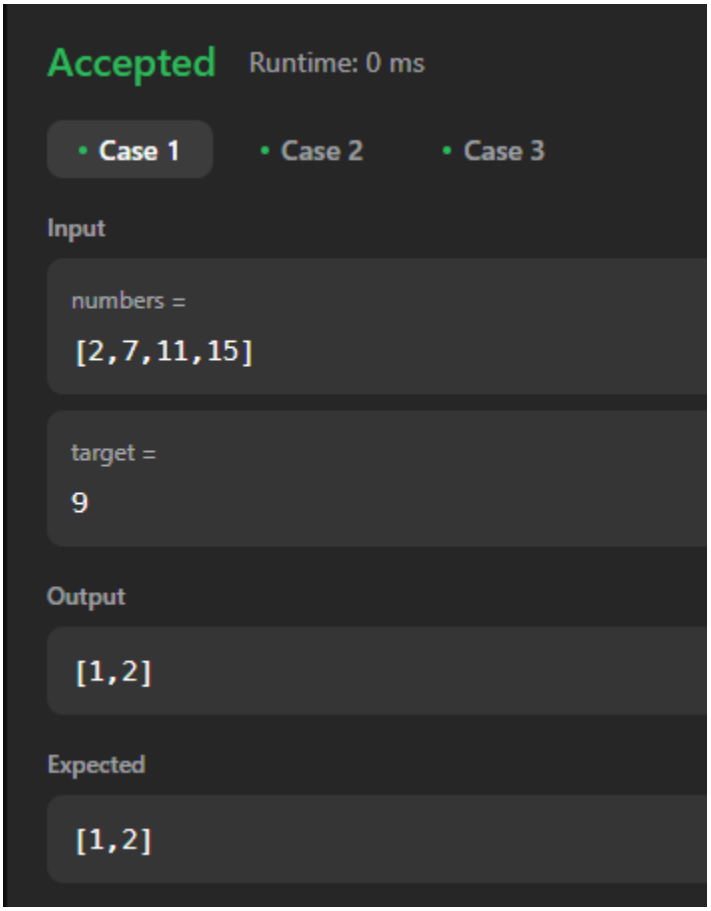
Subject Code: 22CSP-351

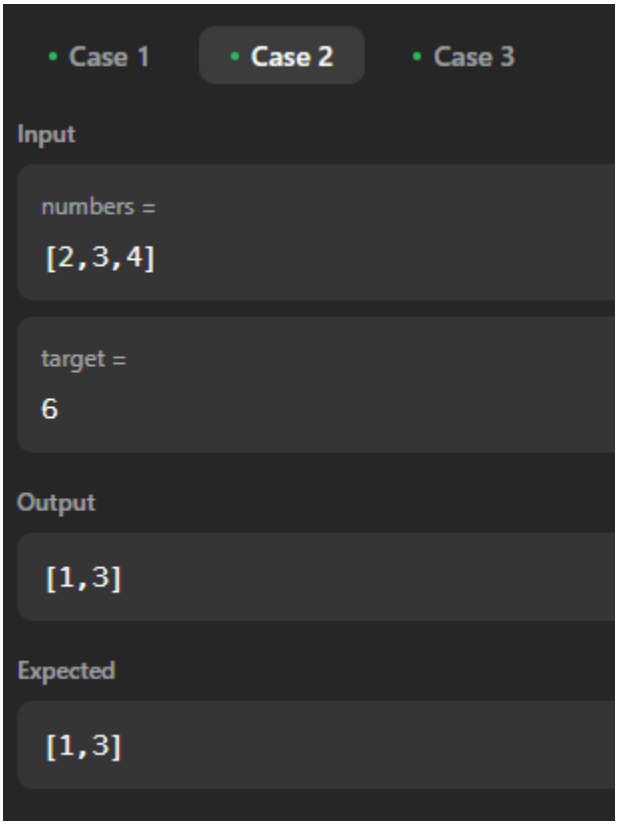
Aim(i): *Two Sum: Given an array of integers, return indices of the two numbers such that they add up to a specific target.*

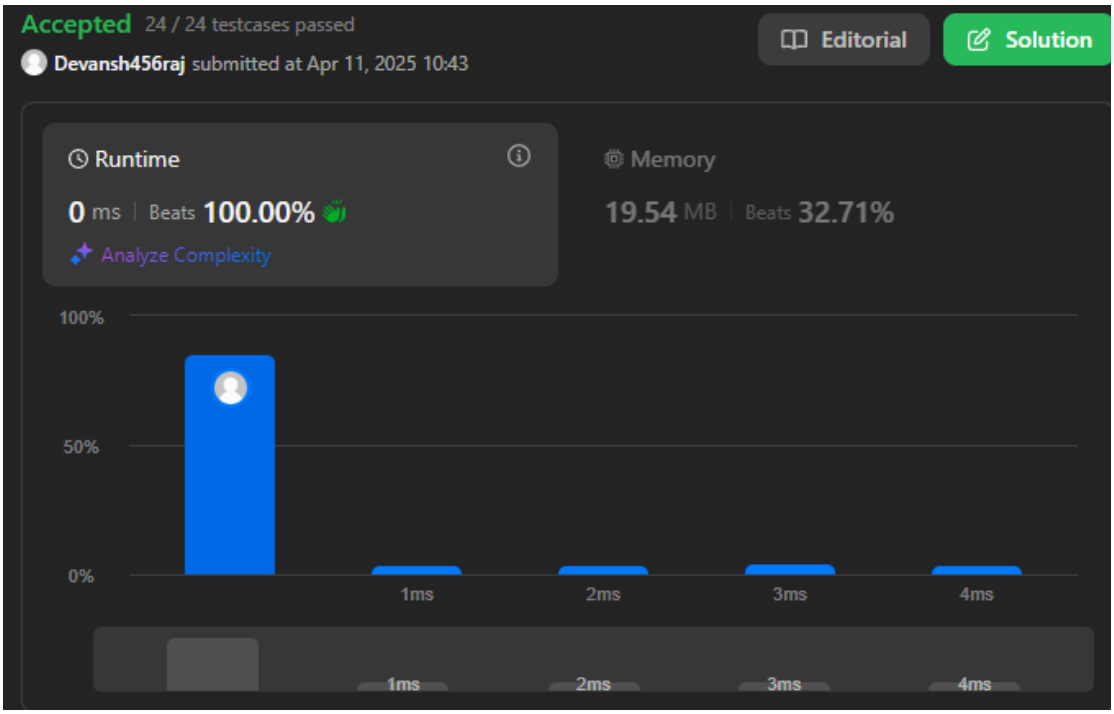
Source Code:

```
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        int left = 0;
        int right = numbers.size() - 1;

        while (left < right) {
            int total = numbers[left] + numbers[right];

            if (total == target) {
                return {left + 1, right + 1};
            } else if (total > target) {
                right--;
            } else {
                left++;
            }
        }
        return {-1, -1}; // If no solution is found
    }
};
```

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]


Case 1
Case 2
Case 3
Input
numbers =
[2,3,4]
target =
6
Output
[1,3]
Expected
[1,3]


Accepted
24 / 24 testcases passed
Devansh456raj submitted at Apr 11, 2025 10:43
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
19.54 MB | Beats 32.71%
100%
50%
0%
1ms
2ms
3ms
4ms

Aim(ii): *Longest Substring Without Repeating Characters: Given a string s, find the length of the longest substring that does not contain any repeating characters.*

Source Code:

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.length();
        int maxLength = 0;
        unordered_set<char> charSet;
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (charSet.count(s[right]) == 0) {
                charSet.insert(s[right]);
                maxLength = max(maxLength, right - left + 1);
            } else {
                while (charSet.count(s[right])) {
                    charSet.erase(s[left]);
                    left++;
                }
                charSet.insert(s[right]);
            }
        }

        return maxLength;
    }
};
```

OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =

"abcabcbb"

Output

3

Expected

3

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =

"bbbbb"

Output

1

Expected

1

Accepted 987 / 987 testcases passed

Devansh456raj submitted at Apr 11, 2025 10:48

Editorial Solution

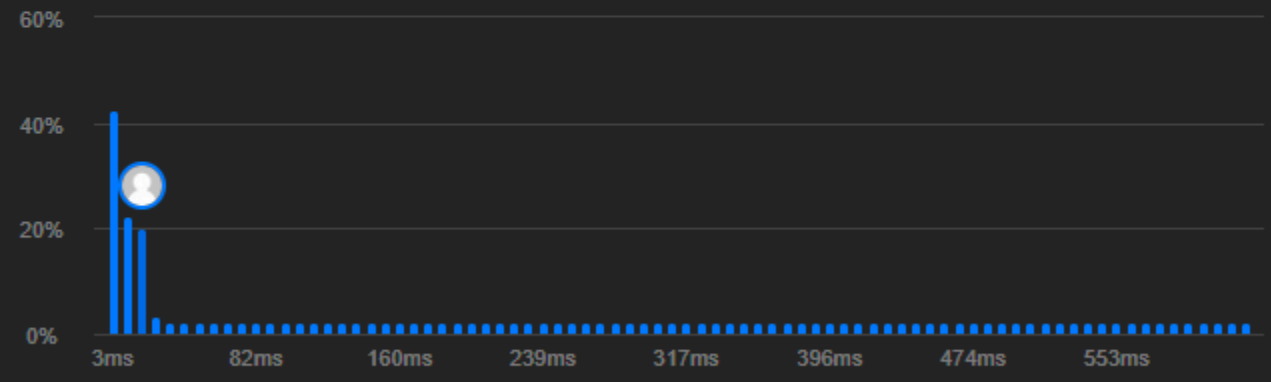
Runtime

22 ms | Beats 22.40%

Analyze Complexity

Memory

14.34 MB | Beats 23.20%



Aim(iii): *Palindrome Number: Determine whether an integer is a palindrome.*

Source Code:

```
class Solution {
public:
    bool isPalindrome(int x) {
        if (x < 0) {
            return false;
        }

        long reverse = 0;
        int xcopy = x;

        while (x > 0) {
            reverse = (reverse * 10) + (x % 10);
            x /= 10;
        }

        return reverse == xcopy;
    }
};
```

OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

x =
121

Output

true

Expected

true

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

x =
−121

Output

false

Expected

false

Accepted 11511 / 11511 testcases passed

Devansh456raj submitted at Apr 11, 2025 10:49

Editorial Solution

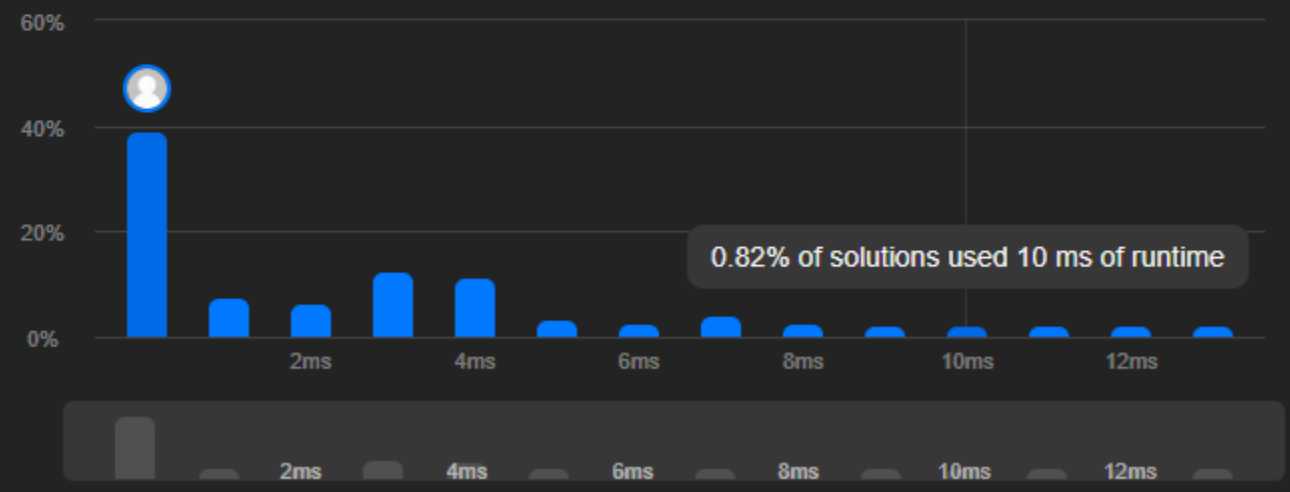
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

8.62 MB | Beats 43.11%

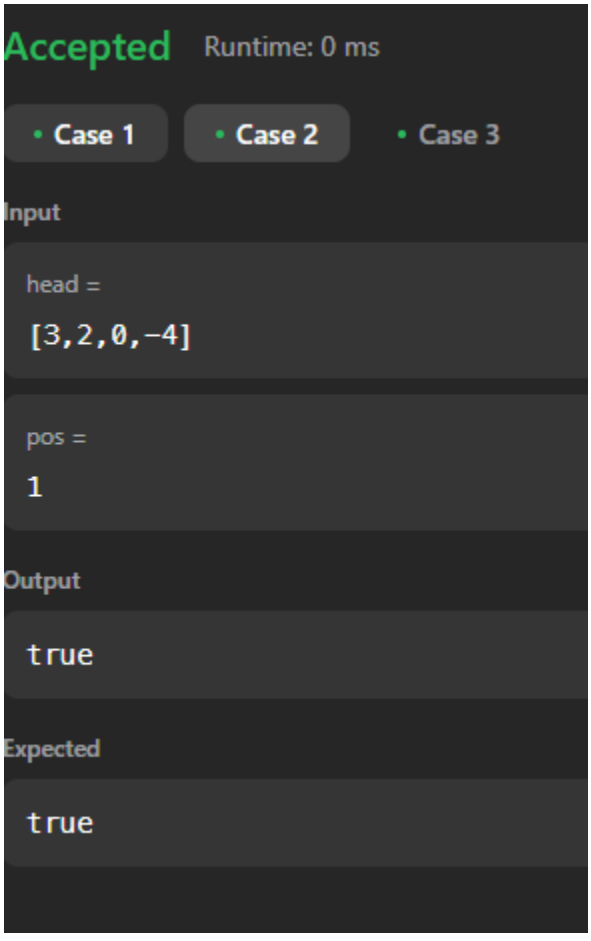


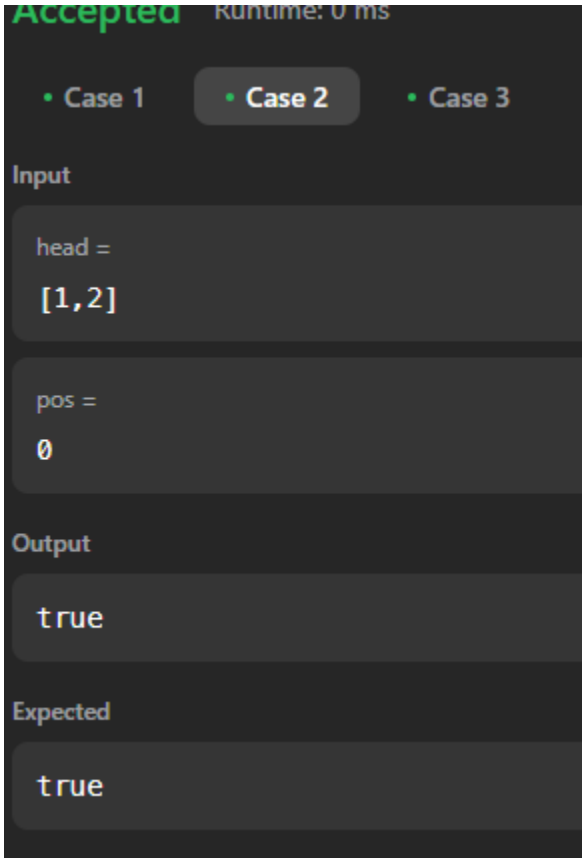
Aim(iv): *Detect a Cycle in a Linked List: Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.*

Source Code:

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head;
        ListNode *fast = head;

        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return true; // Cycle detected
            }
        }
        return false; // No cycle
    }
};
```

OUTPUT:





Devansh456raj submitted at Apr 11, 2025 10:52

Editorial Solution

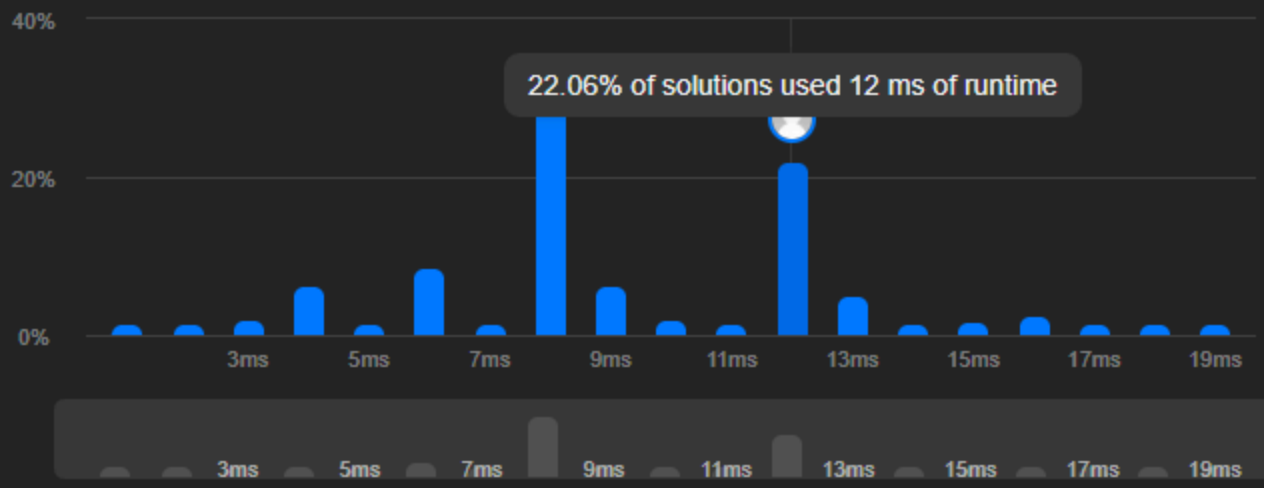
Runtime

12 ms | Beats 39.54%

Analyze Complexity

Memory

11.86 MB | Beats 53.33%

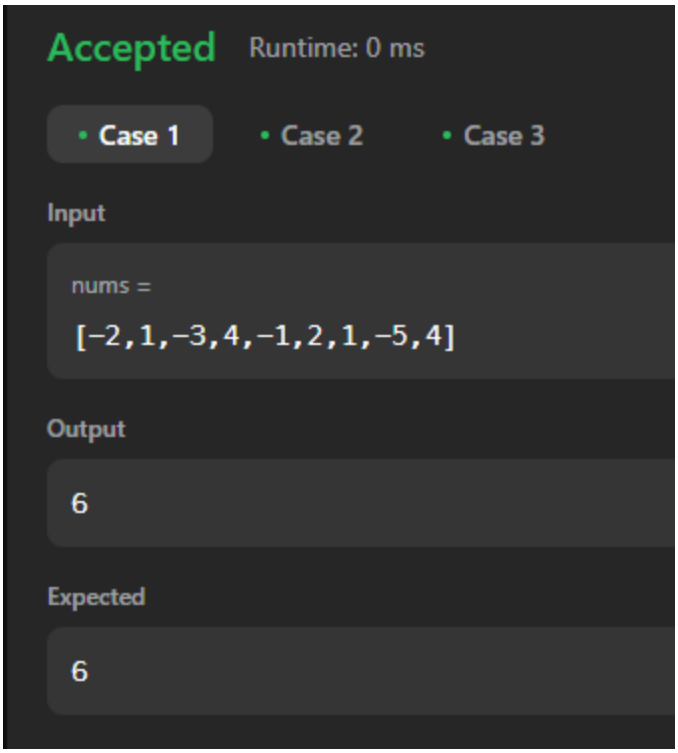


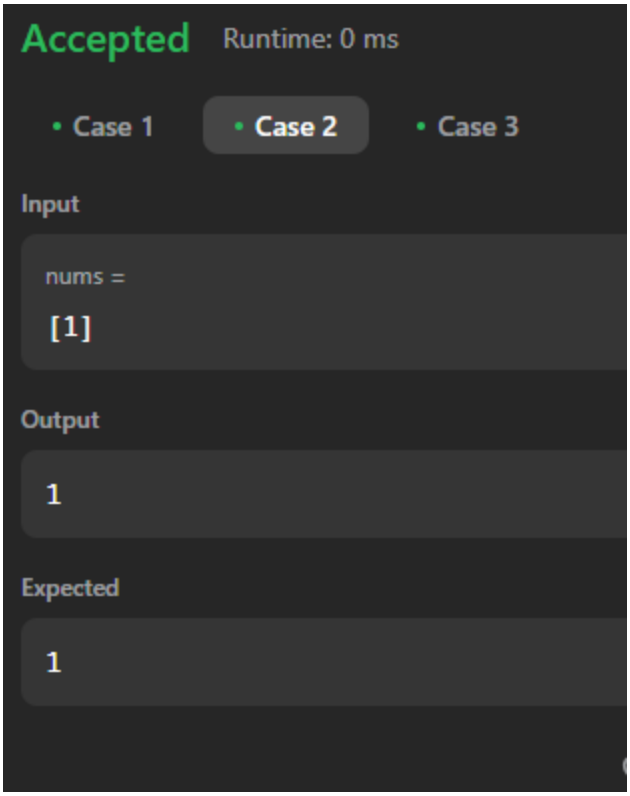
Aim(v): *Maximum Subarray: Find the contiguous subarray (containing at least one number) that has the largest sum and return its sum.*

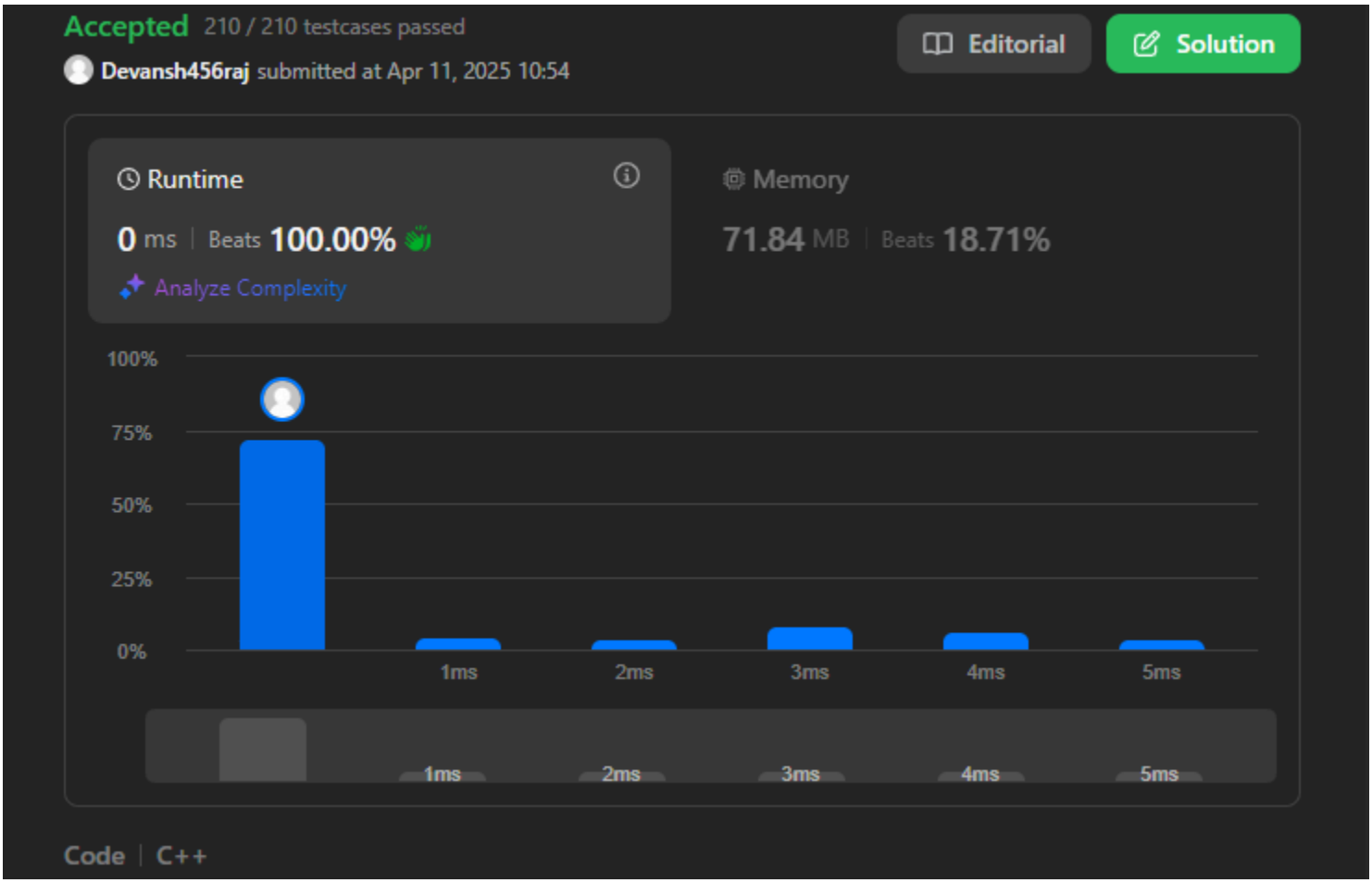
Source Code:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int res = nums[0];
        int total = 0;

        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }

            total += n;
            res = max(res, total);
        }

        return res;
    }
};
```

OUTPUT:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[-2,1,-3,4,-1,2,1,-5,4]
Output
6
Expected
6


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums =
[1]
Output
1
Expected
1


Accepted
210 / 210 testcases passed
Devansh456raj submitted at Apr 11, 2025 10:54
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
71.84 MB | Beats 18.71%
100%
75%
50%
25%
0%
1ms
2ms
3ms
4ms
5ms
Code | C++