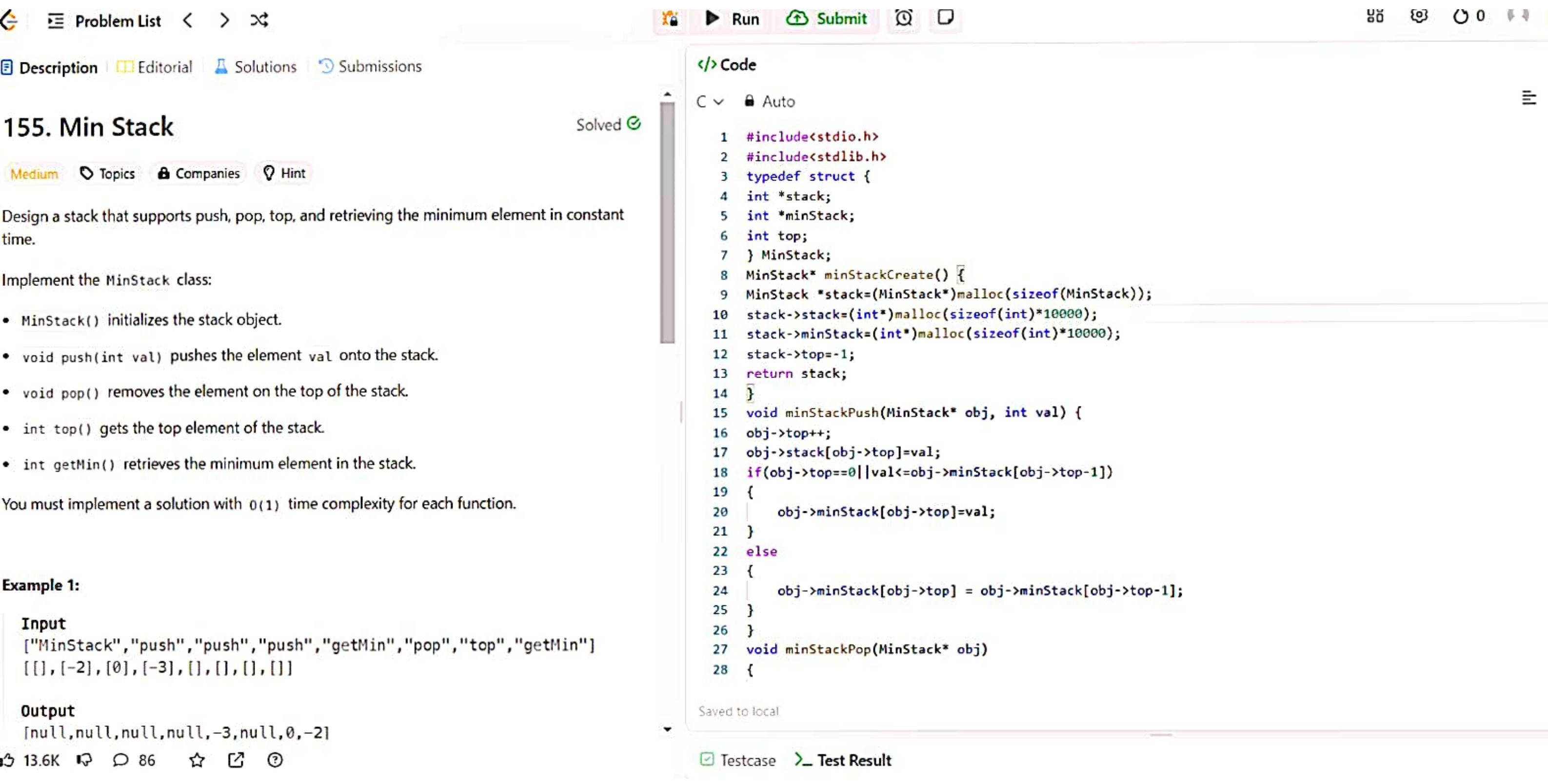


155. Min Stack

Solved

Medium · Topics · Companies · Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

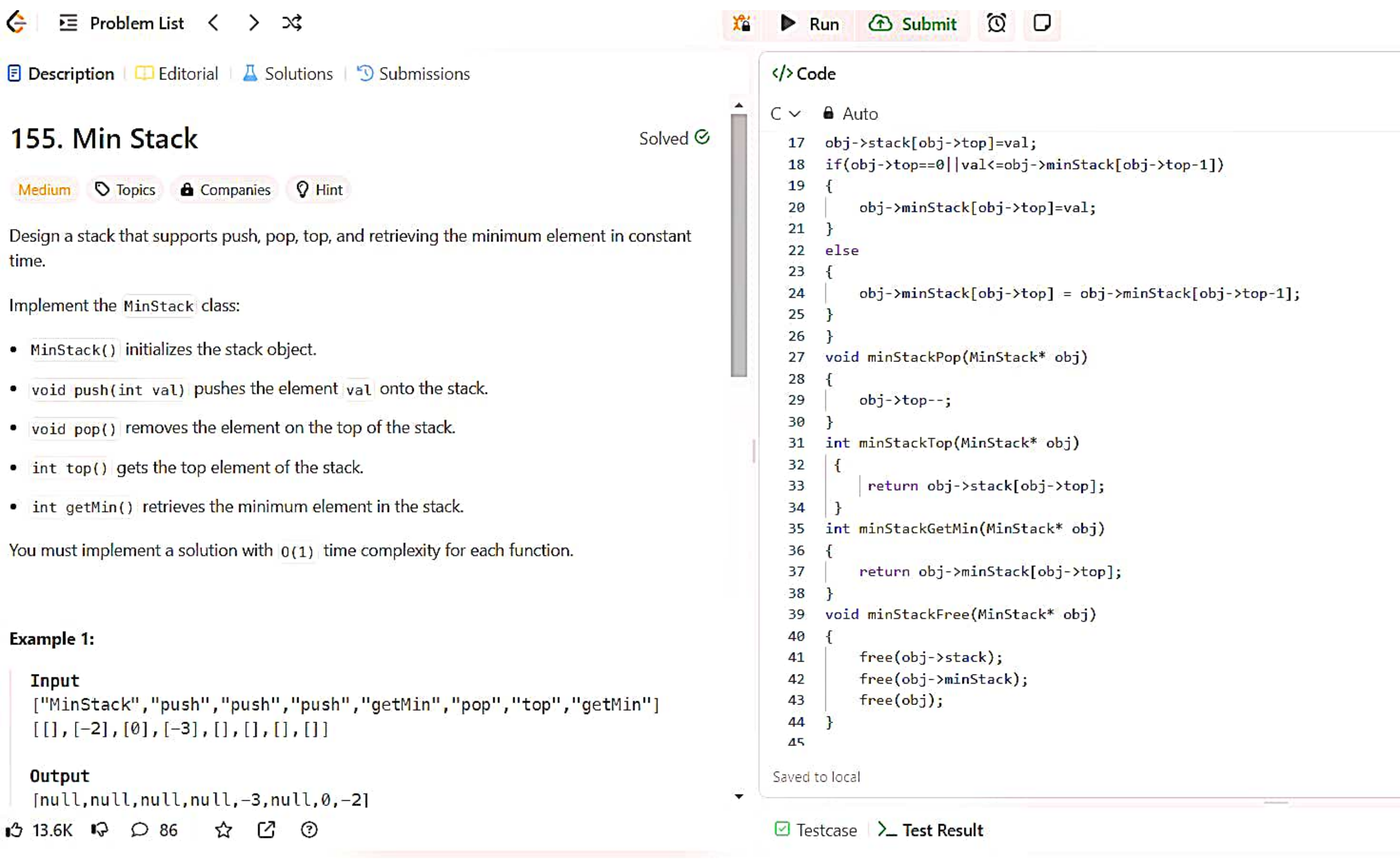
- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]
```

```c
#include<stdio.h>
#include<stdlib.h>
typedef struct {
int *stack;
int *minStack;
int top;
} MinStack;
MinStack* minStackCreate() {
MinStack *stack=(MinStack*)malloc(sizeof(MinStack));
stack->stack=(int*)malloc(sizeof(int)*10000);
stack->minStack=(int*)malloc(sizeof(int)*10000);
stack->top=-1;
return stack;
}
void minStackPush(MinStack* obj, int val) {
obj->top++;
obj->stack[obj->top]=val;
if(obj->top==0||val<=obj->minStack[obj->top-1])
{
    obj->minStack[obj->top]=val;
}
else
{
    obj->minStack[obj->top] = obj->minStack[obj->top-1];
}
}
void minStackPop(MinStack* obj)
{
```



155. Min Stack

Solved

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

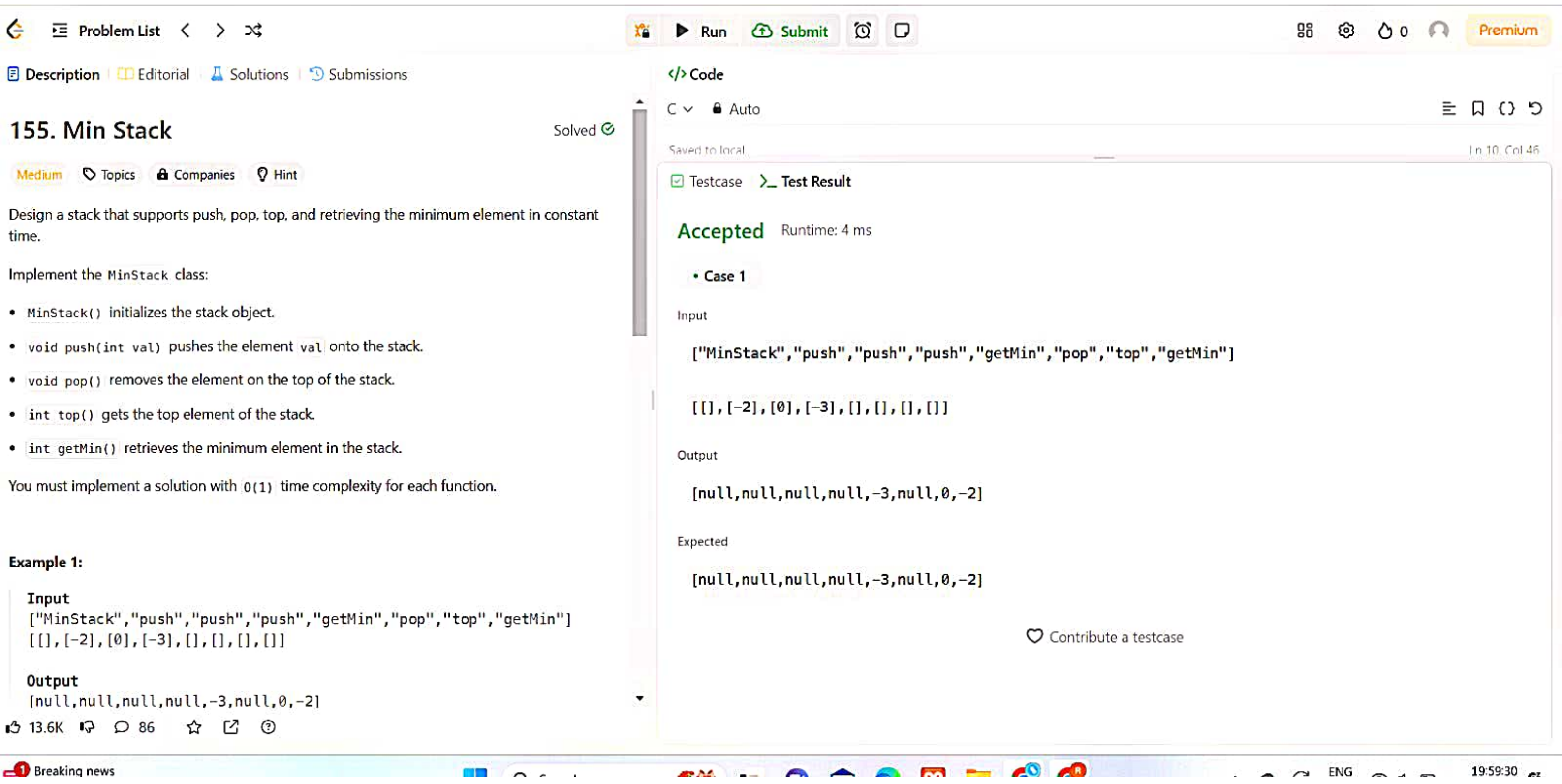
You must implement a solution with `O(1)` time complexity for each function.

Example 1:

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]
```

```c
obj->stack[obj->top]=val;
if(obj->top==0||val<=obj->minStack[obj->top-1])
{
    obj->minStack[obj->top]=val;
}
else
{
    obj->minStack[obj->top] = obj->minStack[obj->top-1];
}
}
void minStackPop(MinStack* obj)
{
    obj->top--;
}
int minStackTop(MinStack* obj)
 {
    return obj->stack[obj->top];
 }
int minStackGetMin(MinStack* obj)
{
    return obj->minStack[obj->top];
}
void minStackFree(MinStack* obj)
{
    free(obj->stack);
    free(obj->minStack);
    free(obj);
}
```

Saved to local



155. Min Stack

Solved

Medium Topics Companies Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]
```

Accepted Runtime: 4 ms

• Case 1

Input

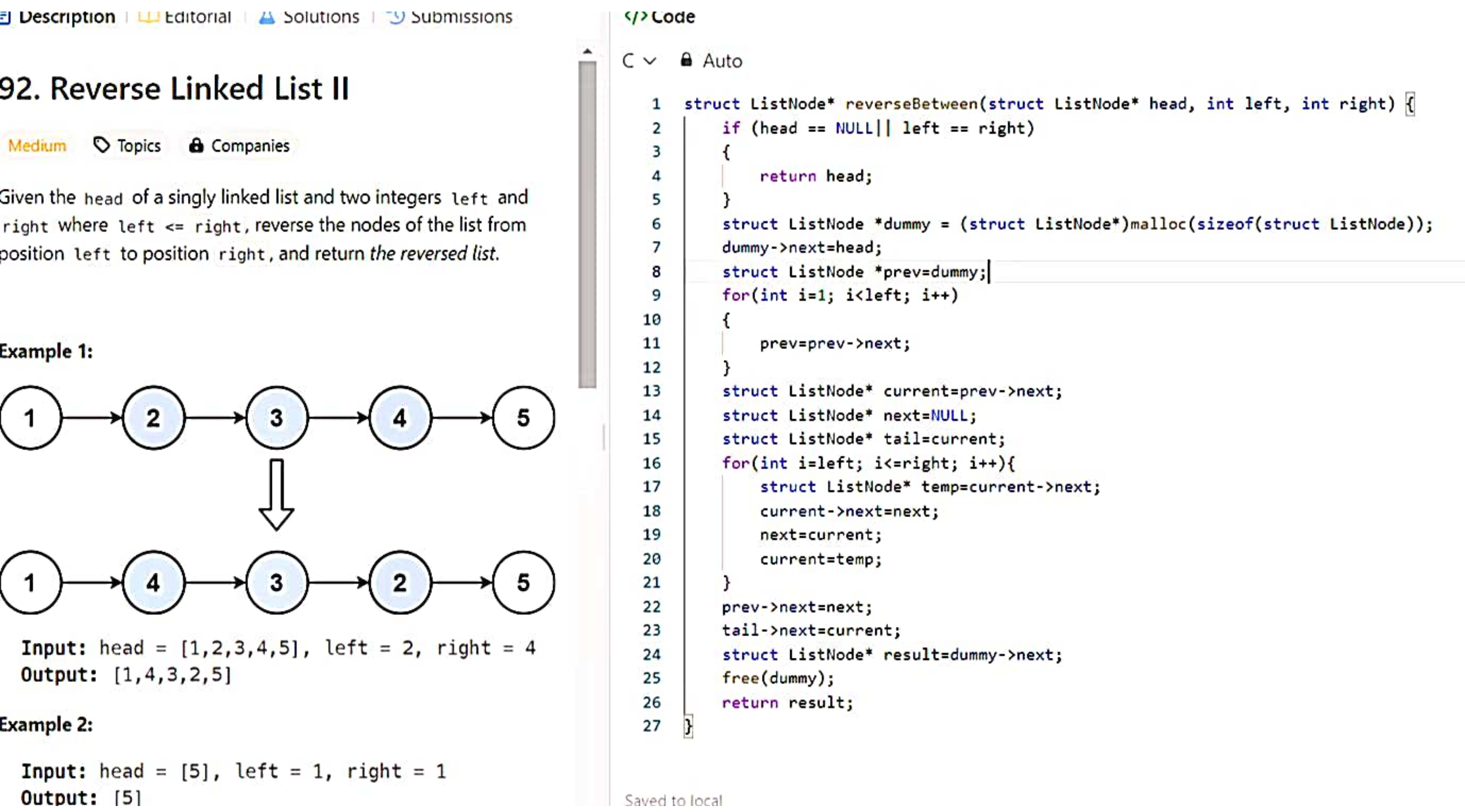
```
["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]
```

Output

```
[null,null,null,null,-3,null,0,-2]
```

Expected

```
[null,null,null,null,-3,null,0,-2]
```

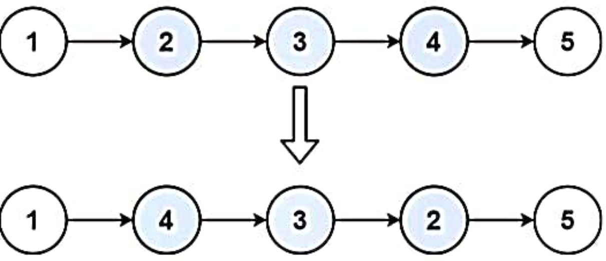


92. Reverse Linked List II

Medium Topics Companies

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from position `left` to position `right`, and return *the reversed list*.

Example 1:



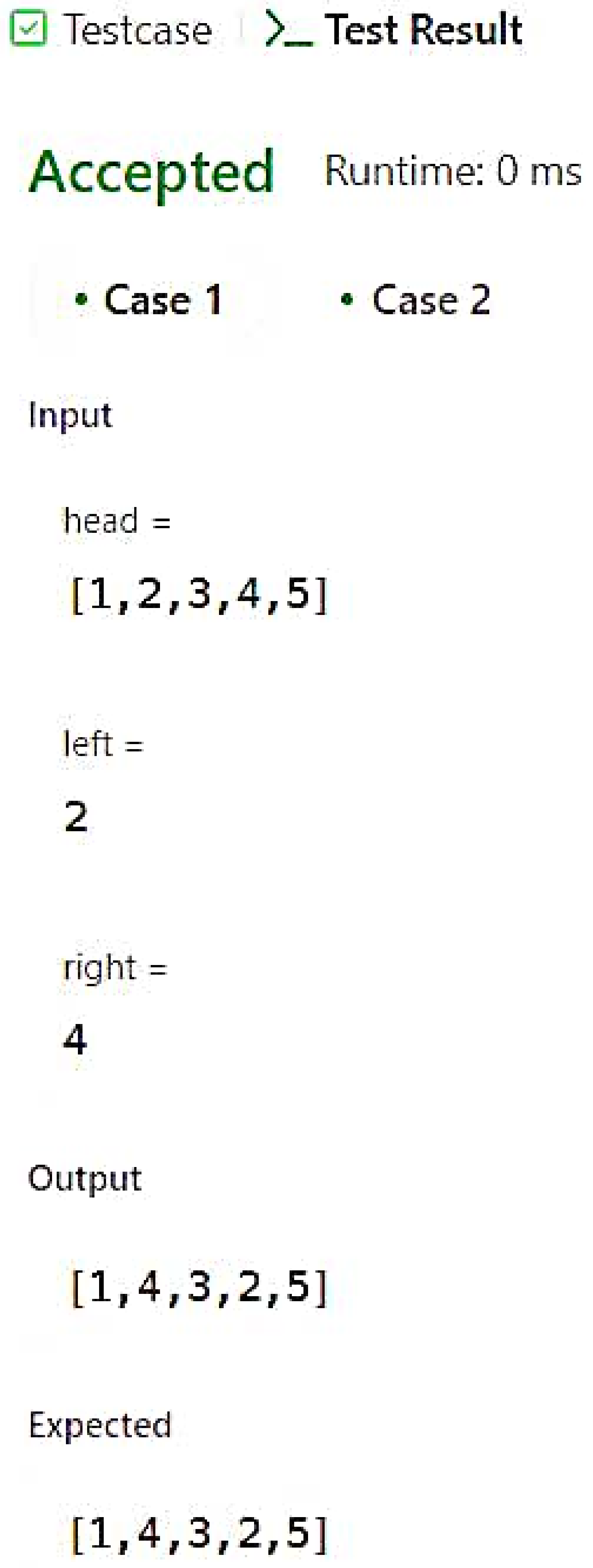
```
Input: head = [1,2,3,4,5], left = 2, right = 4
Output: [1,4,3,2,5]
```

Example 2:

```
Input: head = [5], left = 1, right = 1
Output: [5]
```

```c
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (head == NULL|| left == right)
    {
        return head;
    }
    struct ListNode *dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->next=head;
    struct ListNode *prev=dummy;
    for(int i=1; i<left; i++)
    {
        prev=prev->next;
    }
    struct ListNode* current=prev->next;
    struct ListNode* next=NULL;
    struct ListNode* tail=current;
    for(int i=left; i<=right; i++){
        struct ListNode* temp=current->next;
        current->next=next;
        next=current;
        current=temp;
    }
    prev->next=next;
    tail->next=current;
    struct ListNode* result=dummy->next;
    free(dummy);
    return result;
}
```

Saved to local



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

```
[1,2,3,4,5]
```

left =

```
2
```

right =

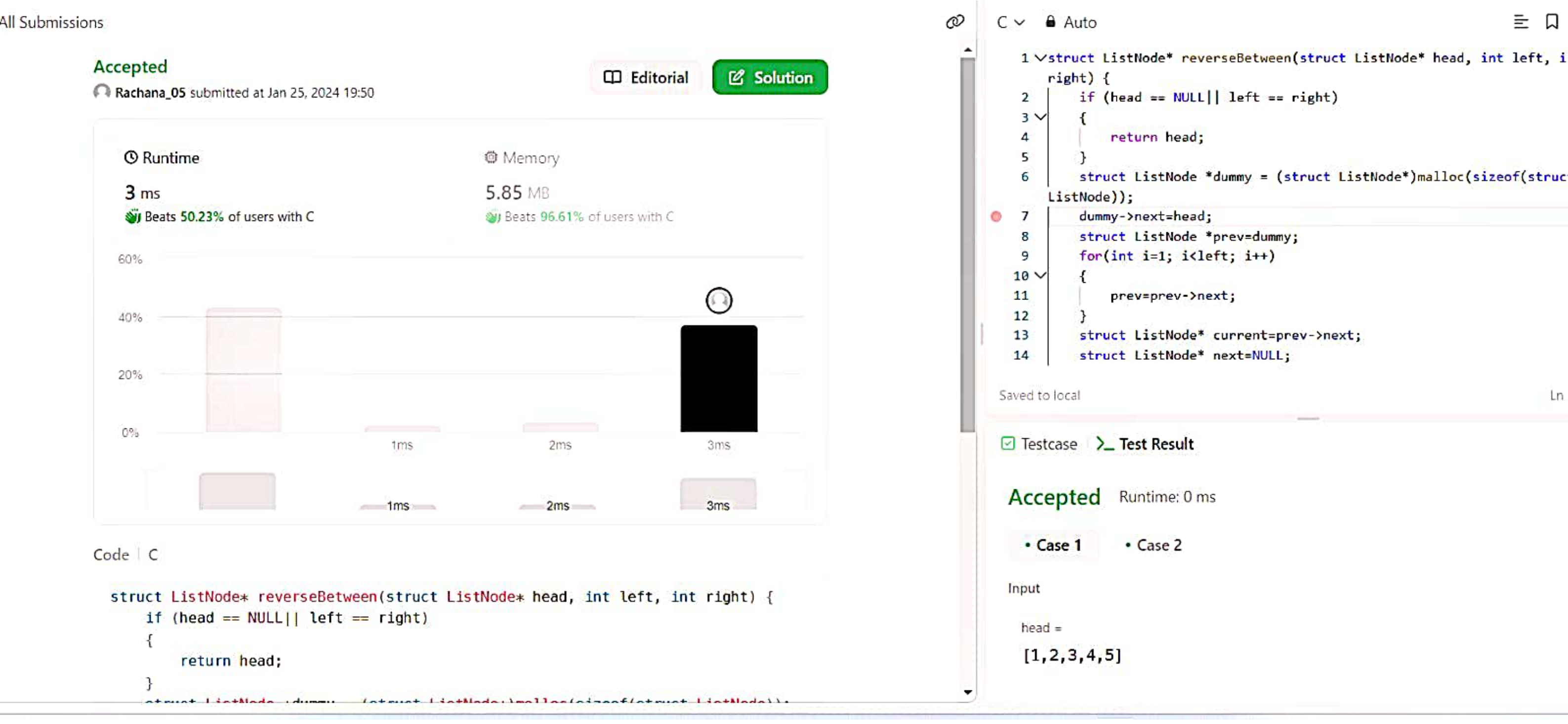
```
4
```

Output

```
[1,4,3,2,5]
```

Expected

```
[1,4,3,2,5]
```



Accepted

Rachana_05 submitted at Jan 25, 2024 19:50

Runtime

3 ms

Beats 50.23% of users with C

Memory

5.85 MB

Beats 96.61% of users with C

Code | C

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right) {
    if (head == NULL|| left == right)
    {
        return head;
    }
```

```
struct ListNode* reverseBetween(struct ListNode* head, int left, i
right) {
    if (head == NULL|| left == right)
    {
        return head;
    }
    struct ListNode *dummy = (struct ListNode*)malloc(sizeof(struc
ListNode));
    dummy->next=head;
    struct ListNode *prev=dummy;
    for(int i=1; i<left; i++)
    {
        prev=prev->next;
    }
    struct ListNode* current=prev->next;
    struct ListNode* next=NULL;
```

Saved to local

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

head =

[1,2,3,4,5]