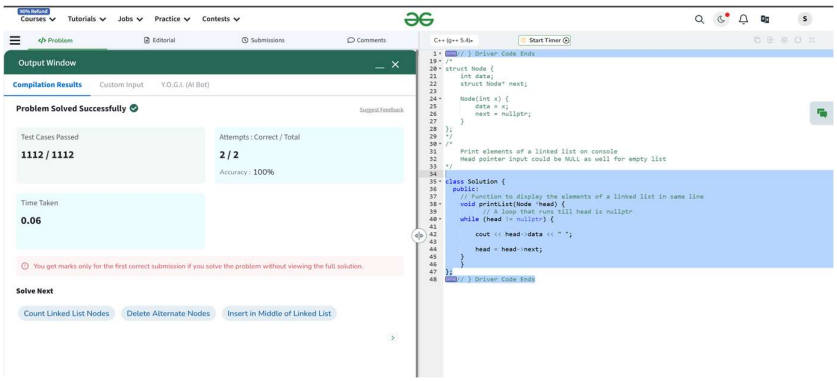
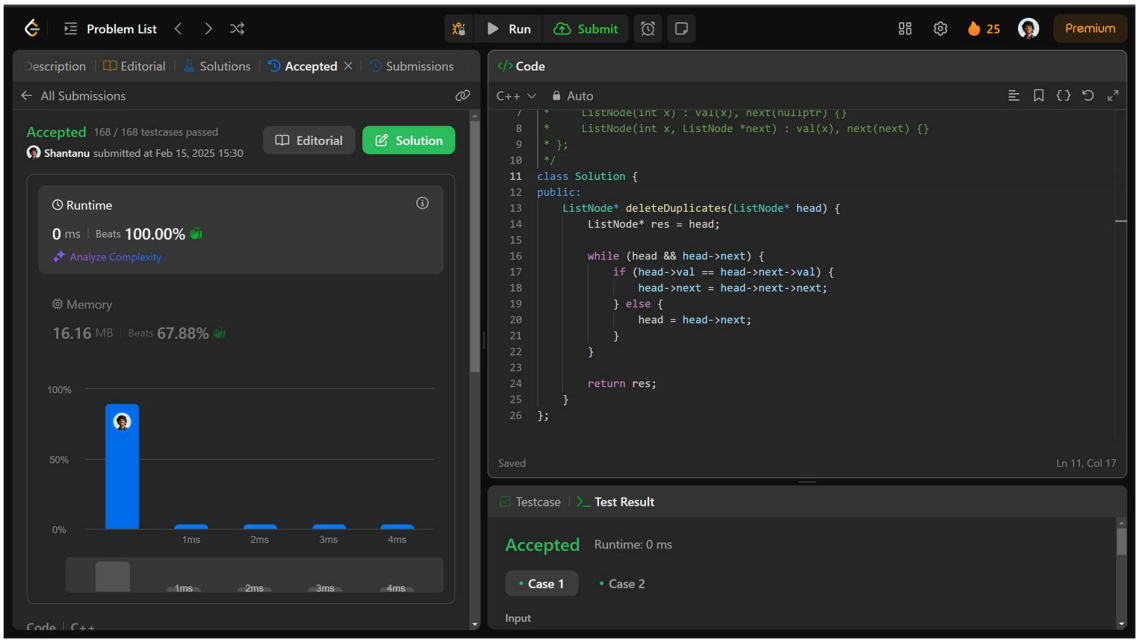
Q1. Print Linked List

```
class Solution {
 public:
  void printList(Node *head) {
  while (head != nullptr) {
     cout << head->data << " ";
     head = head->next;
  }
  }
};
```



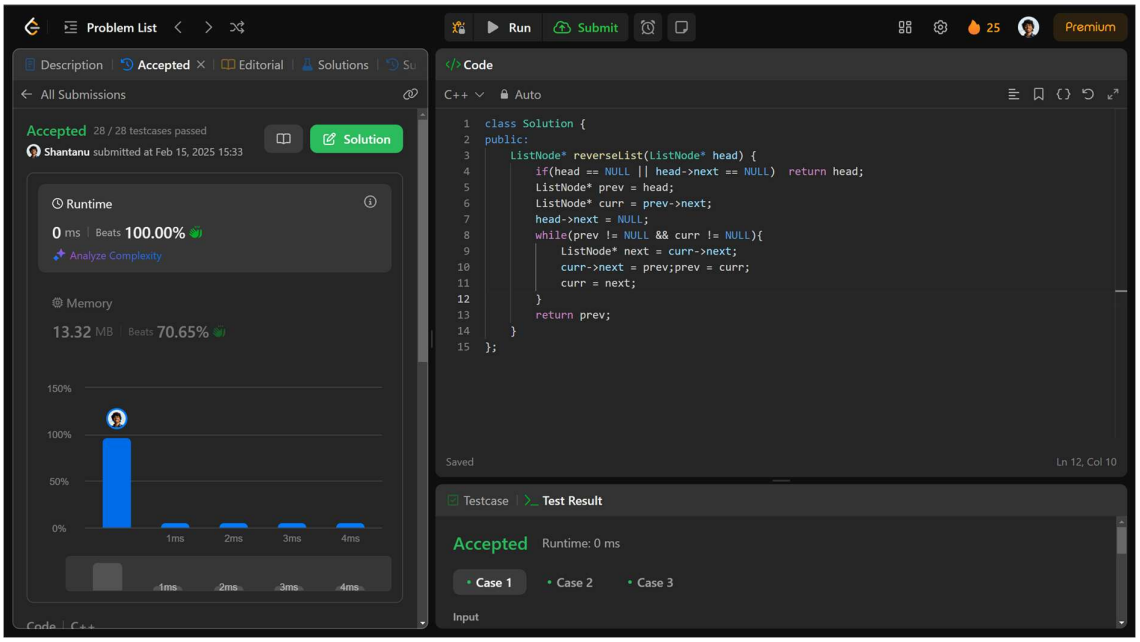
Q2. [83. Remove Duplicates from Sorted List]

```
class Solution {
public:
  ListNode* deleteDuplicates(ListNode* head) {
    ListNode* res = head;
    while (head && head->next) {
      if (head->val == head->next->val) {
        head->next = head->next->next;
      } else {
        head = head->next;
      }
    }
    return res;
  }
};
```



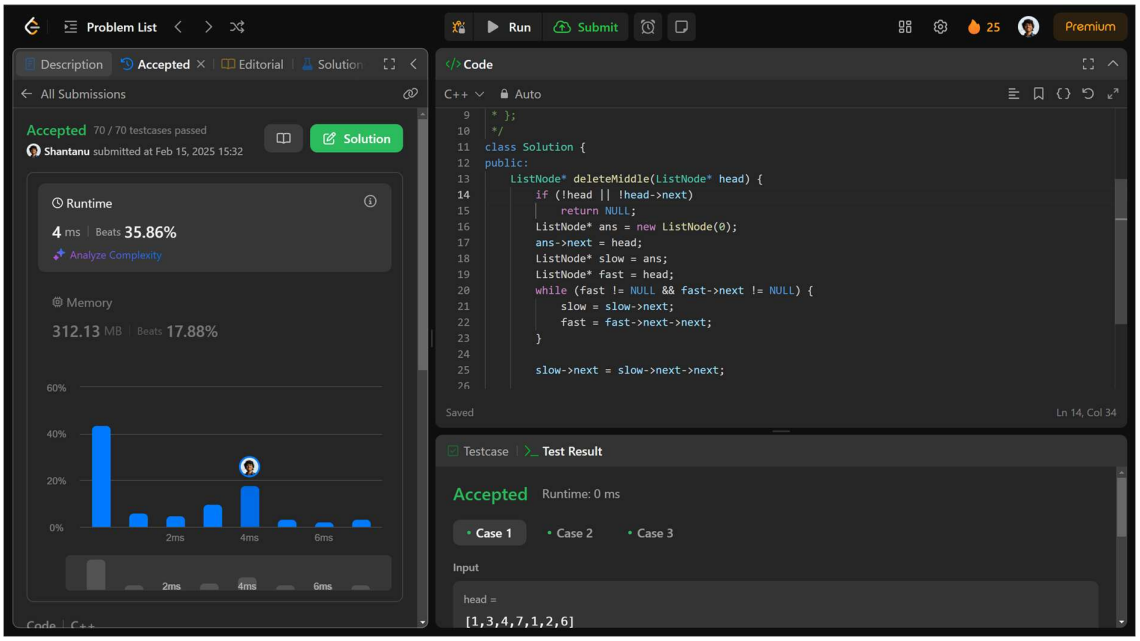
Q3. [206. Reverse Linked List]

```
class Solution {
public:
  ListNode* reverseList(ListNode* head) {
    if(head == NULL || head->next == NULL)  return head;
    ListNode* prev = head;
    ListNode* curr = prev->next;
    head->next = NULL;
    while(prev != NULL && curr != NULL){
      ListNode* next = curr->next;
      curr->next = prev;prev = curr;
      curr = next;
    }
    return prev;
  }
};
```



Q4. [2095. Delete the Middle Node of a Linked List]

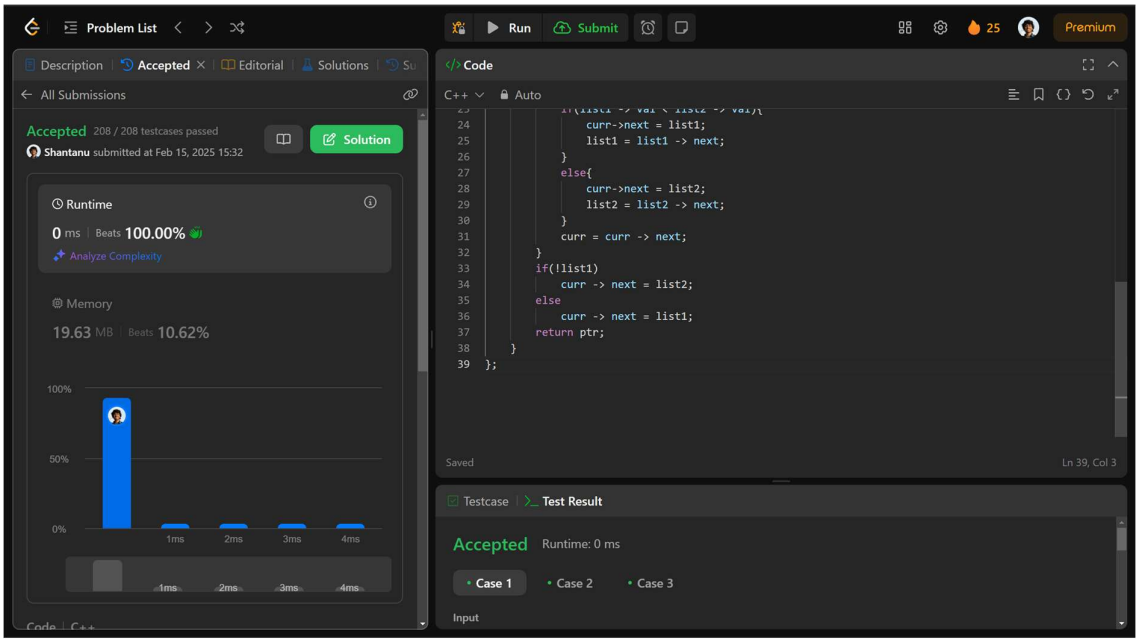
```
class Solution {
public:
  ListNode* deleteMiddle(ListNode* head) {
    if (!head || !head->next)
      return NULL;
    ListNode* ans = new ListNode(0);
    ans->next = head;
    ListNode* slow = ans;
    ListNode* fast = head;
    while (fast != NULL && fast->next != NULL) {
      slow = slow->next;
      fast = fast->next->next;
    }

    slow->next = slow->next->next;

    return ans->next;
  }
};
```



Q5. [21. Merge Two Sorted Lists]

```
class Solution {
public:
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if(list1 == NULL)
      return list2;
    if(list2 == NULL)
      return list1;
    ListNode * ptr = list1;
    if(list1 -> val > list2 -> val){
      ptr = list2;
      list2 = list2 -> next;
    }
    else{
      list1 = list1 -> next;
    }
    ListNode *curr = ptr;
    while(list1 &&  list2){
      if(list1 -> val < list2 -> val){
        curr->next = list1;
        list1 = list1 -> next;
      }
      else{
        curr->next = list2;
```

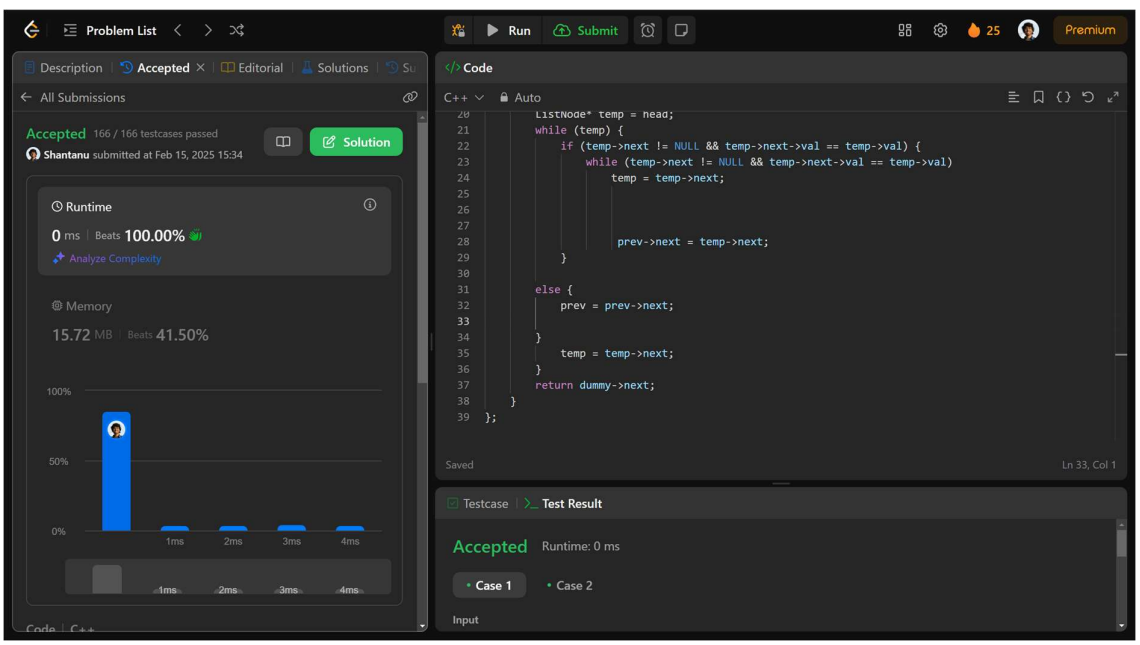
```
            list2 = list2 -> next;
        }
        curr = curr -> next;
    }
    if(!list1)
        curr -> next = list2;
    else
        curr -> next = list1;
    return ptr;
  }
};
```



Q6. [82. Remove Duplicates from Sorted List II]

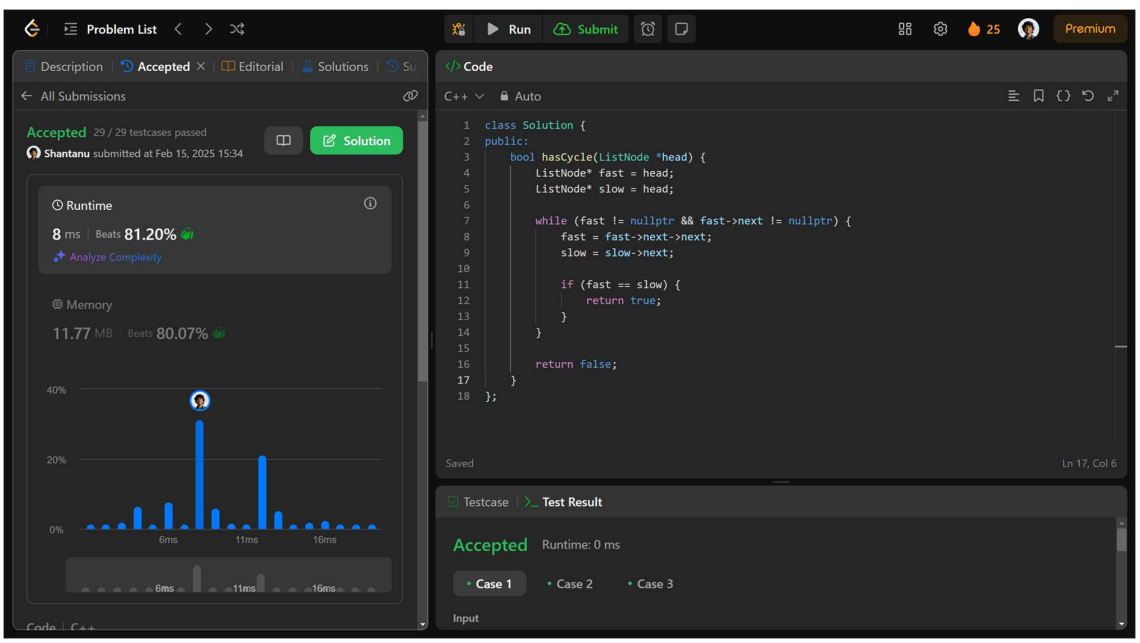
```
class Solution {
public:
  ListNode* deleteDuplicates(ListNode* head) {
    // Edge case
    if (head == NULL || head->next == NULL) {
      return head;
    }
    ListNode* dummy = new ListNode(0, head);
    ListNode* prev = dummy;
    ListNode* temp = head;
    while (temp) {
      if (temp->next != NULL && temp->next->val == temp->val) {
        while (temp->next != NULL && temp->next->val == temp->val)
```

```
            temp = temp->next;


            prev->next = temp->next;
        }

    else {
        prev = prev->next;

    }
        temp = temp->next;
    }
    return dummy->next;
  }
};
```



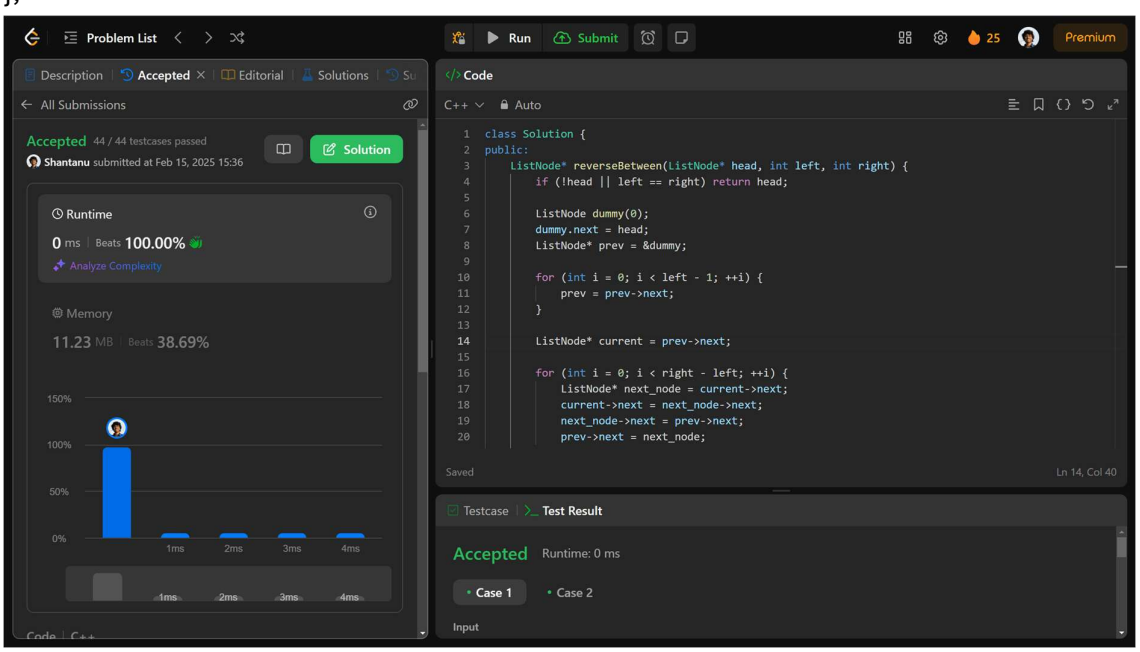
Q7. [141. Linked List Cycle]

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

    while (fast != nullptr && fast->next != nullptr) {
      fast = fast->next->next;
      slow = slow->next;

      if (fast == slow) {
        return true;
      }
    }

    return false;
  }
};
```



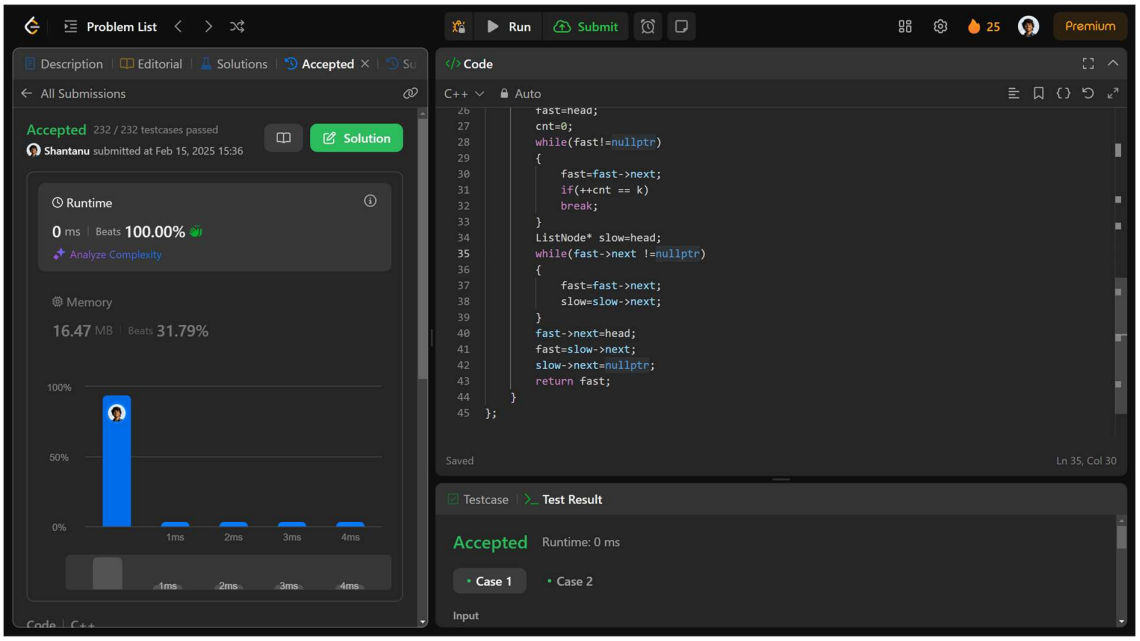
Q8. [92. Reverse Linked List II]

```
class Solution {
public:
  ListNode* reverseBetween(ListNode* head, int left, int right) {
    if (!head || left == right) return head;
    ListNode dummy(0);
    dummy.next = head;
    ListNode* prev = &dummy;
    for (int i = 0; i < left - 1; ++i) {
      prev = prev->next;
    }
    ListNode* current = prev->next;
    for (int i = 0; i < right - left; ++i) {
      ListNode* next_node = current->next;
      current->next = next_node->next;
      next_node->next = prev->next;
      prev->next = next_node;
    }

    return dummy.next;
  }
};
```



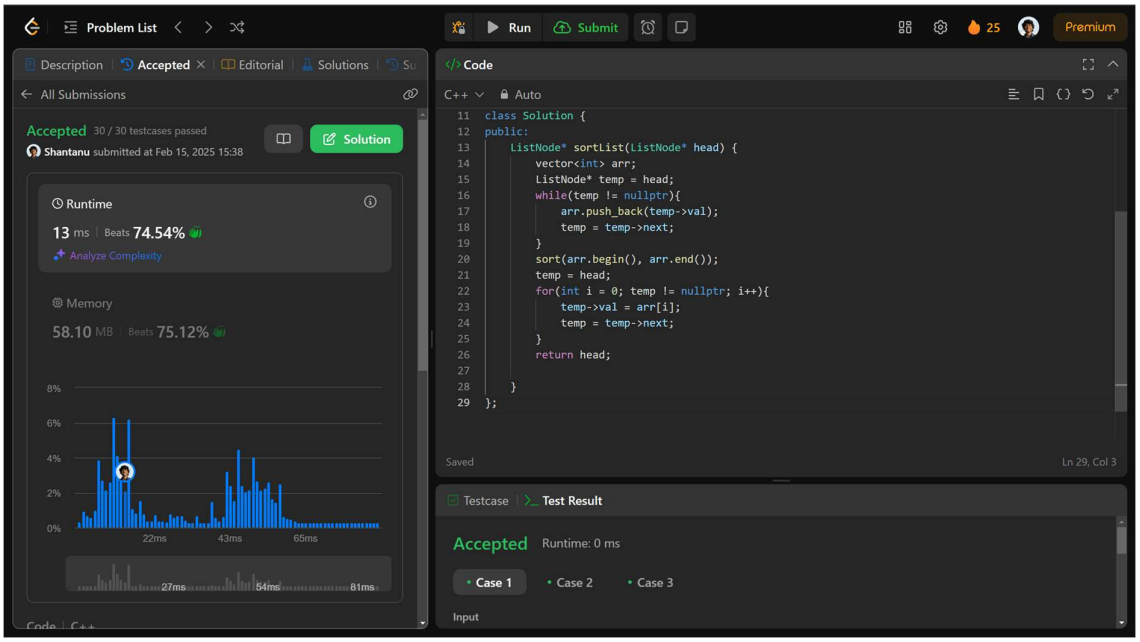
Q9. **61. Rotate List**

```
class Solution {
public:
  ListNode* rotateRight(ListNode* head, int k) {
    if(head==nullptr || head->next==nullptr)
    return head;
    ListNode* fast=head;
    int cnt=0;
    while(fast != nullptr)
    {
      cnt++;
      fast=fast->next;
    }
    k= k%cnt;
    if(k==0)
    return head;
    fast=head;
    cnt=0;
    while(fast!=nullptr)
    {
      fast=fast->next;
      if(++cnt == k)
      break;
    }
    ListNode* slow=head;
    while(fast->next !=nullptr)
    {
      fast=fast->next;
      slow=slow->next;
    }
    fast->next=head;
    fast=slow->next;
    slow->next=nullptr;
    return fast;
  }
};
```



Q10. [148. Sort List]

```
class Solution {
public:
  ListNode* sortList(ListNode* head) {
    vector<int> arr;
    ListNode* temp = head;
    while(temp != nullptr){
      arr.push_back(temp->val);
      temp = temp->next;
    }
    sort(arr.begin(), arr.end());
    temp = head;
    for(int i = 0; temp != nullptr; i++){
      temp->val = arr[i];
      temp = temp->next;
    }
    return head;
  }
};
```



Q11. [142. Linked List Cycle II]

```
class Solution {
public:
  ListNode *detectCycle(ListNode *head) {
    ListNode *slow = head, *fast = head;
    while (fast && fast->next) {
      slow = slow->next;
      fast = fast->next->next;
      if (slow == fast) break;
    }
    if (!(fast && fast->next)) return NULL;
    while (head != slow) {
      head = head->next;
      slow = slow->next;
    }
    return head;
  }
};
```

Problem List

Run Submit 25 Premium

Description | Accepted | Editorial | Solutions | Su

All Submissions

Accepted 18 / 18 testcases passed

Shantanu submitted at Feb 15, 2025 15:38

Solution

Runtime

10 ms | Beats 45.96%

Analyze Complexity

Memory

11.36 MB | Beats 53.95%

Code | C++

Code

C++ Auto

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) break;
        }
        if (!(fast && fast->next)) return NULL;
        while (head != slow) {
            head = head->next;
            slow = slow->next;
        }
        return head;
    }
};
```

Saved Ln 17, Col 3

Testcase | Test Result

Accepted Runtime: 2 ms

Case 1 Case 2 Case 3

Input