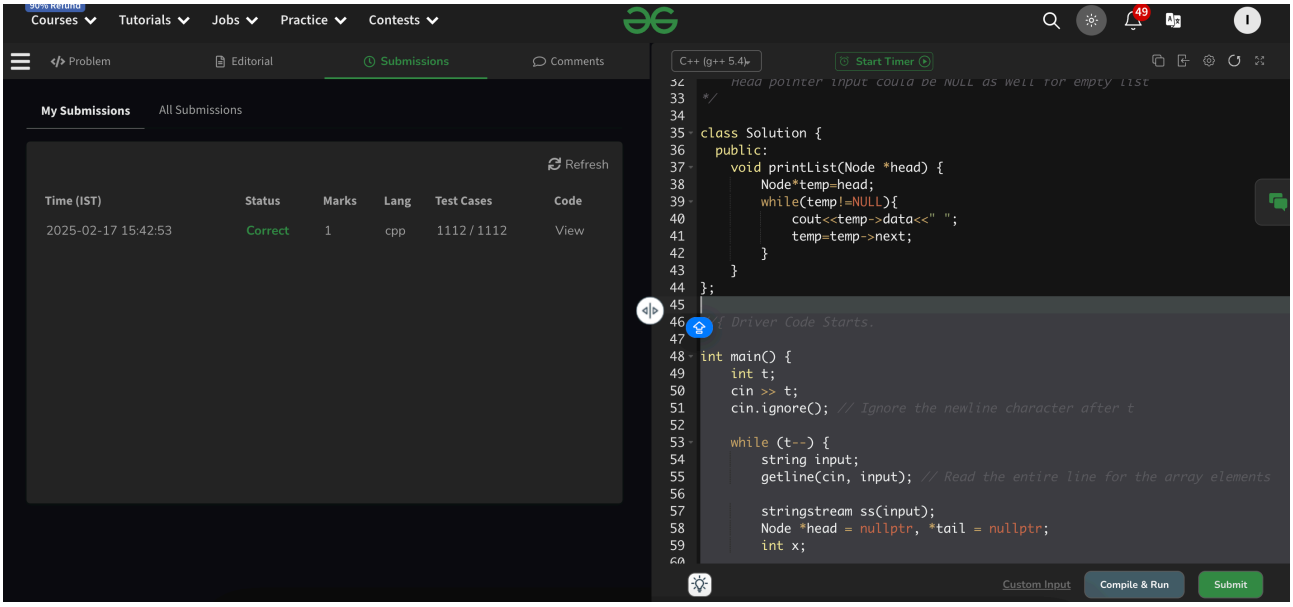
Name : Ishita	**Date of Submission : 05/03/2025**
UID : 22BCS14845	**Subject Code : 22CSP - 351**
Subject : Advance Programming Lab	**Submitted to : Er. Pratima Sonali**

ASSIGNMENT - 2

Question 1

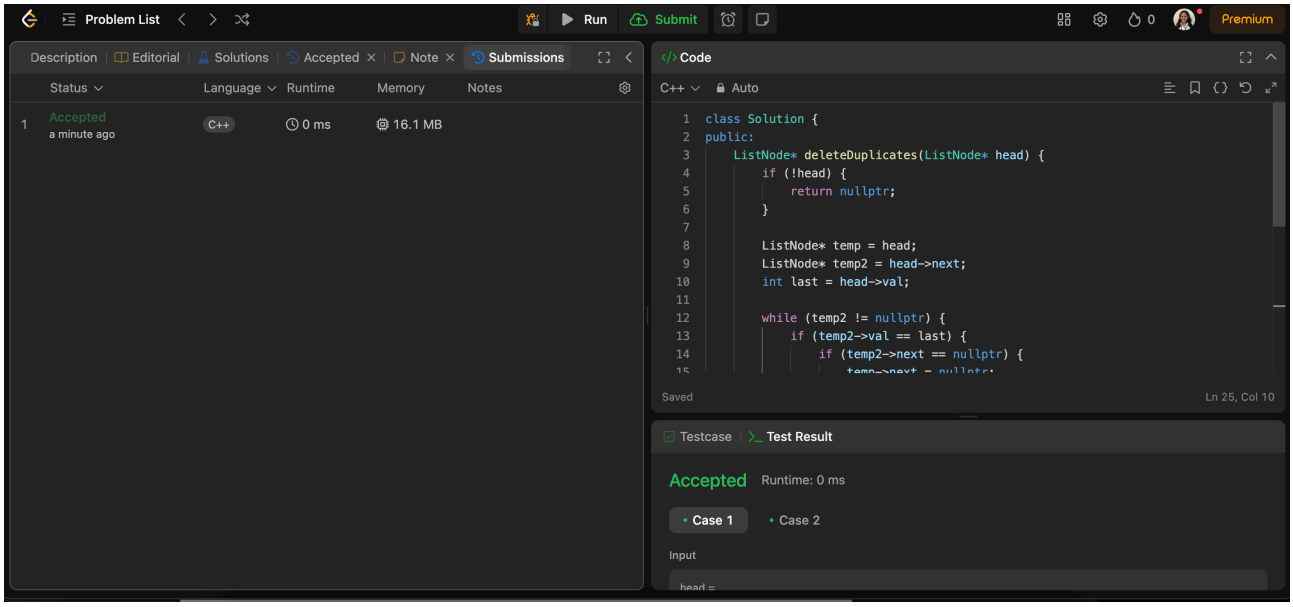
```
class Solution {
  public:
    void printList(Node *head) {
        Node*temp=head;
        while(temp!=NULL){
            cout<<temp->data<<" ";
            temp=temp->next;
        }
    }
};
```



Question 2

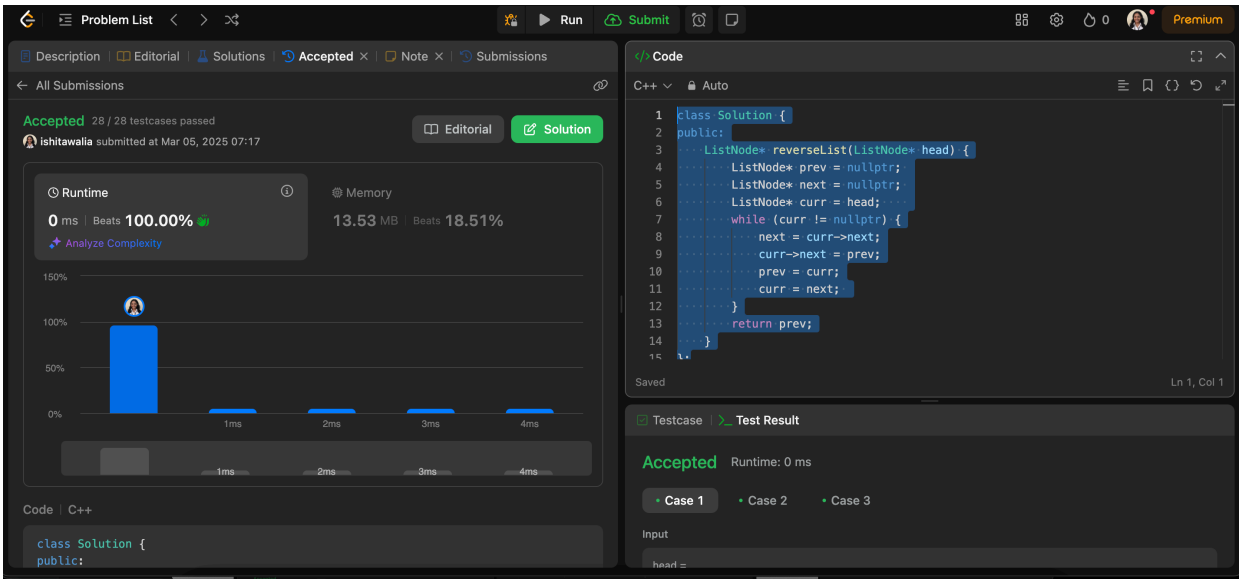
```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (!head) {
            return nullptr;
        }

        ListNode* temp = head;
        ListNode* temp2 = head->next;
        int last = head->val;

        while (temp2 != nullptr) {
            if (temp2->val == last) {
                if (temp2->next == nullptr) {
                    temp->next = nullptr;
                    break;
                }
                temp2 = temp2->next; /
                temp->next = temp2;
            } else {
```

```
            temp = temp2;
            last = temp->val;
            temp2 = temp2->next;
        }
    }
    return head;
  }
};
```



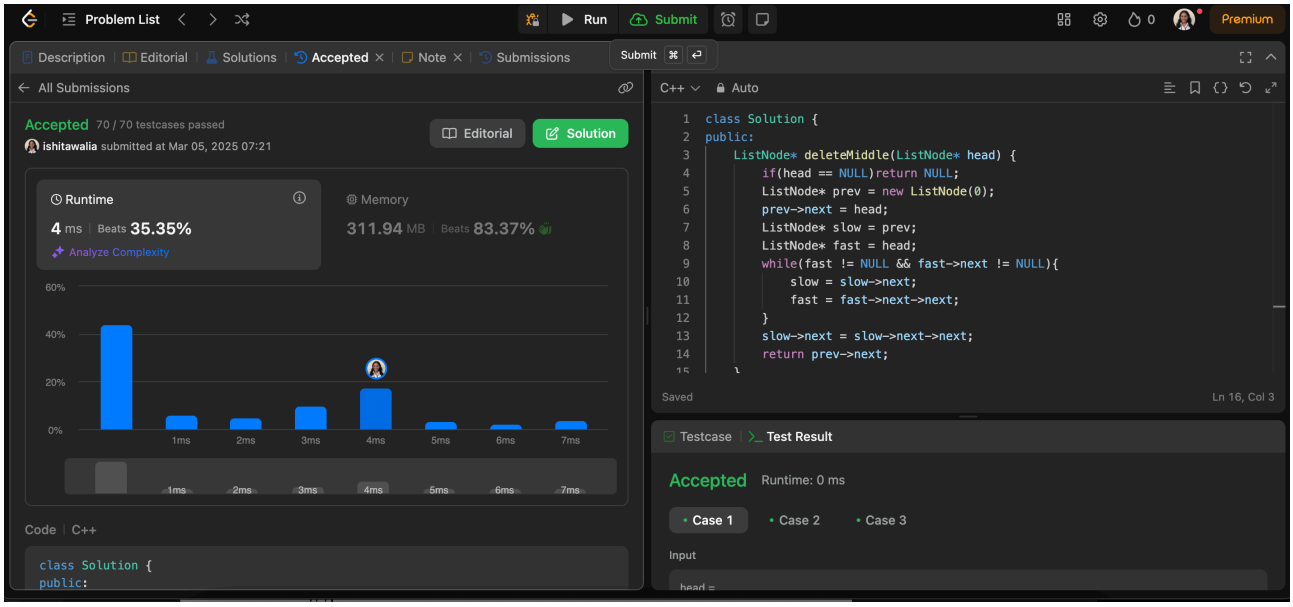
Question 3

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

};
```



Question 4

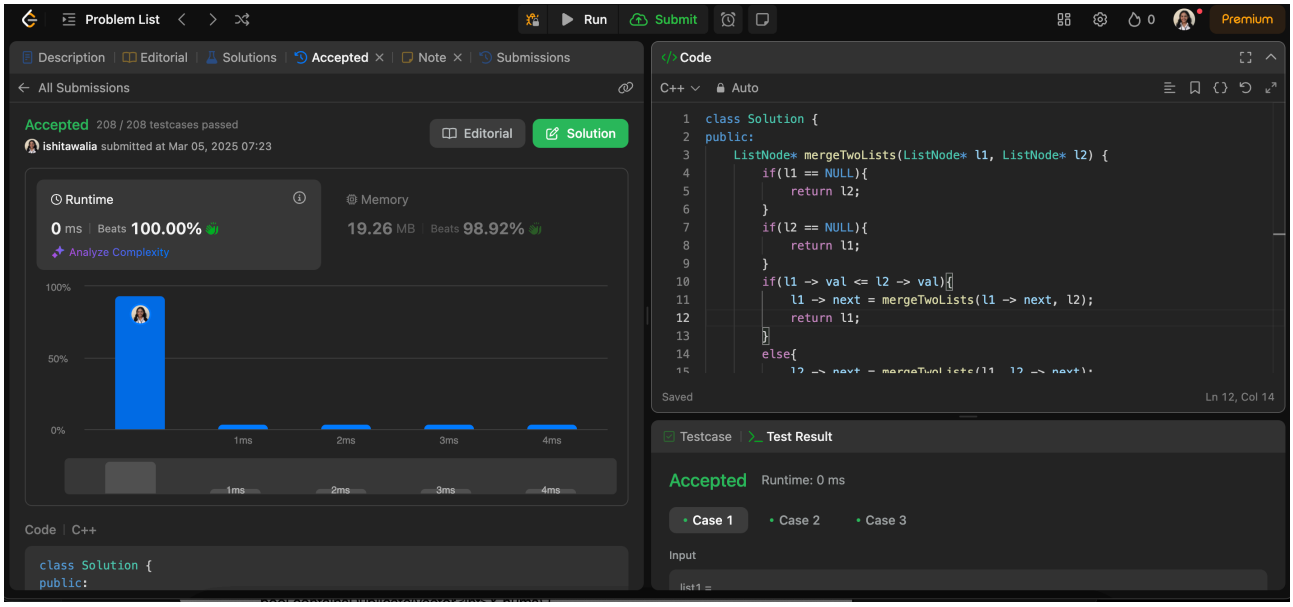
```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(head == NULL)return NULL;
        ListNode* prev = new ListNode(0);
        prev->next = head;
        ListNode* slow = prev;
        ListNode* fast = head;
        while(fast != NULL && fast->next != NULL){
       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



Question 5

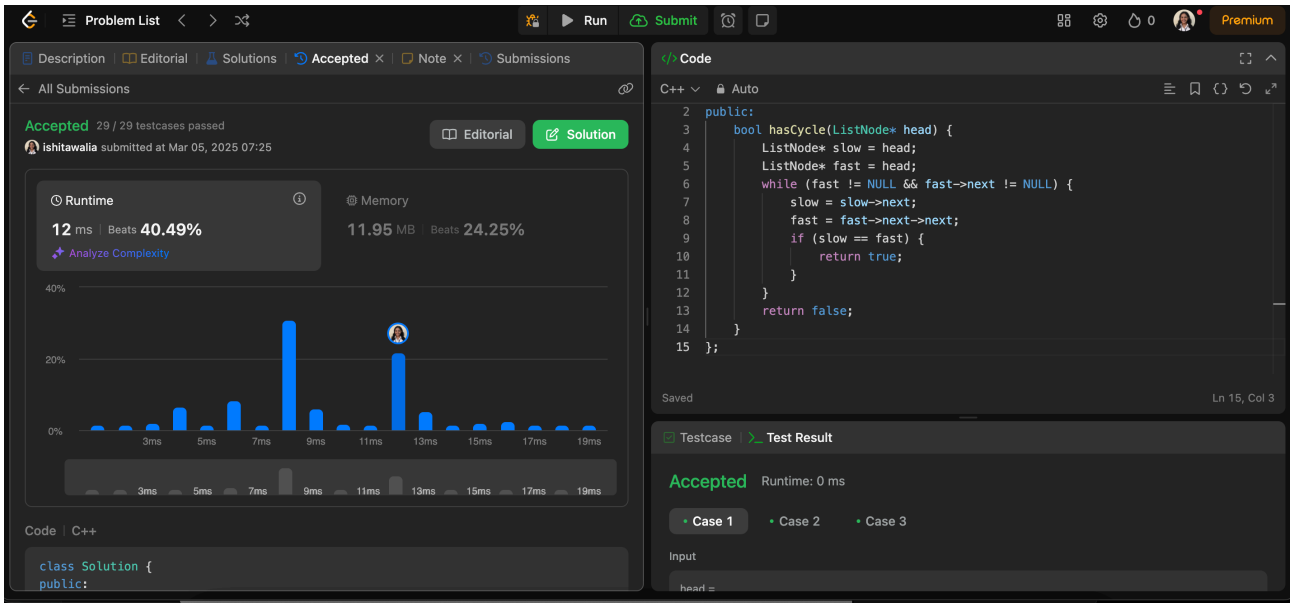
```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 == NULL){
```

```
        return l2;
    }
    if(l2 == NULL){
        return l1;
    }
    if(l1 -> val <= l2 -> val){
        l1 -> next = mergeTwoLists(l1 -> next, l2);
        return l1;
    }
    else{
        l2 -> next = mergeTwoLists(l1, l2 -> next);
        return l2;
    }
  }
};
```



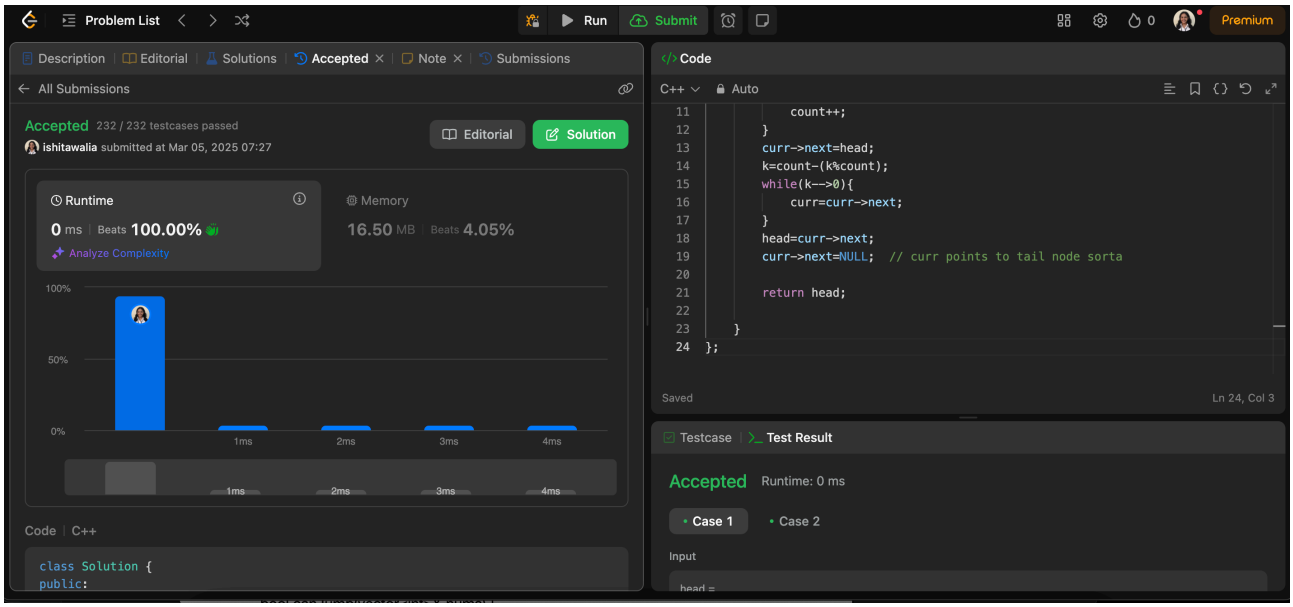
Question 6

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        while (fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return true;
            }
        }
        return false;
    }
};
```



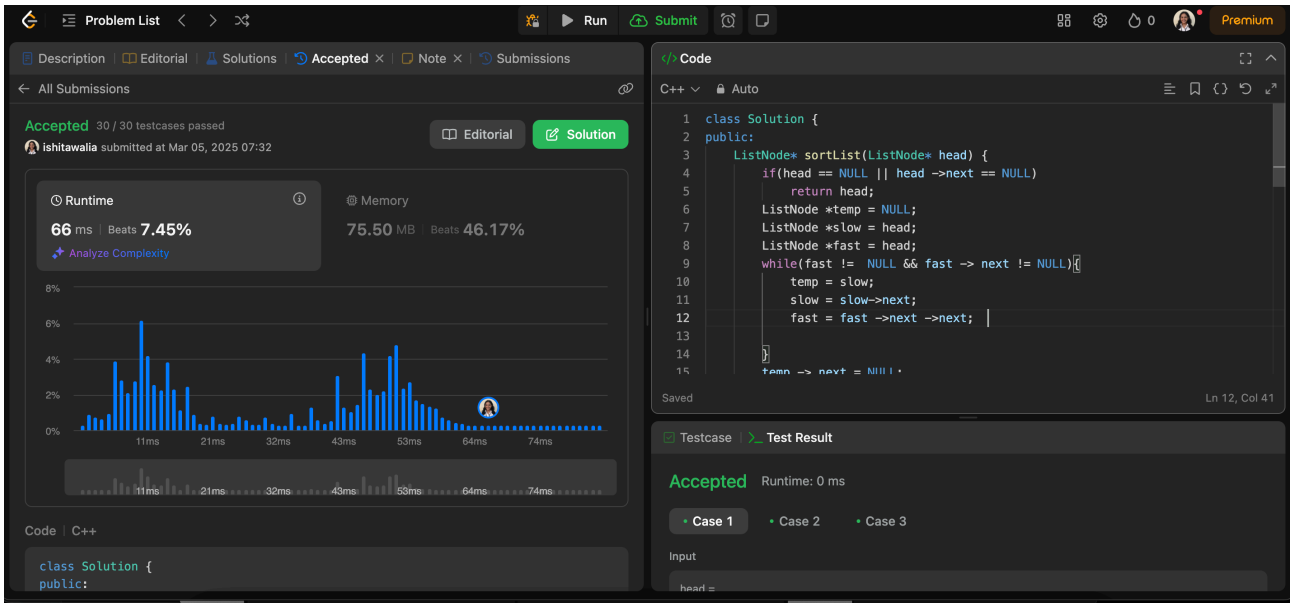
Question 7

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        // base condition
        if(head==NULL || head->next==NULL || k==0) return head;

        ListNode* curr=head;
        int count=1;
        while(curr->next!=NULL){
            curr=curr->next;
            count++;
        }
        curr->next=head;
        k=count-(k%count);
        while(k-->0){
            curr=curr->next;
        }
        head=curr->next;
        curr->next=NULL;  // curr points to tail node sorta

        return head;

    }
};
```



Question 8

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if(head == NULL || head ->next == NULL)
            return head;
        ListNode *temp = NULL;
        ListNode *slow = head;
        ListNode *fast = head;
        while(fast !=  NULL && fast -> next != NULL){
            temp = slow;
            slow = slow->next;
            fast = fast ->next ->next;

        }
        temp -> next = NULL;
        ListNode* l1 = sortList(head);
        ListNode* l2 = sortList(slow);
        return mergelist(l1, l2);
    }
    ListNode* mergelist(ListNode *l1, ListNode *l2){
        ListNode *ptr = new ListNode(0);
        ListNode *curr = ptr;
        while(l1 != NULL && l2 != NULL){
            if(l1->val <= l2->val){
                curr -> next = l1;
                l1 = l1 -> next;
            }
            else{
                curr -> next = l2;
                l2 = l2 -> next;
            }
        curr = curr ->next;
        }
        if(l1 != NULL){
            curr -> next = l1;
            l1 = l1->next;
        }
        if(l2 != NULL){
            curr -> next = l2;
            l2 = l2 ->next;
        }
        return ptr->next;
    }
};
```



Question 9

```
#include <vector>
using namespace std;
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;
        return divideAndConquer(lists, 0, lists.size() - 1);
    }

    ListNode* divideAndConquer(vector<ListNode*>& lists, int left, int right) {
        if (left == right) return lists[left];

        int mid = left + (right - left) / 2;
        ListNode* l1 = divideAndConquer(lists, left, mid);
        ListNode* l2 = divideAndConquer(lists, mid + 1, right);
        return mergeTwoLists(l1, l2);
    }
};
```

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Accepted | Note | Submissions

All Submissions

Accepted 134 / 134 testcases passed

ishitawalia submitted at Mar 05, 2025 07:33

Editorial | Solution

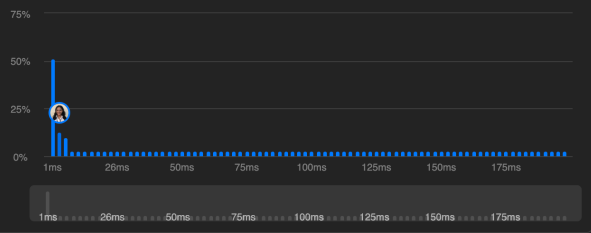
Runtime

4 ms | Beats 48.89%

Analyze Complexity

Memory

18.69 MB | Beats 40.18%



Code | C++

```cpp
#include <vector>
using namespace std;
```

Code

C++ | Auto

```cpp
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;
        return divideAndConquer(lists, 0, lists.size() - 1);
    }

    ListNode* divideAndConquer(vector<ListNode*>& lists, int left, int right) {
        if (left == right) return lists[left];

        int mid = left + (right - left) / 2;
        ListNode* l1 = divideAndConquer(lists, left, mid);
        ListNode* l2 = divideAndConquer(lists, mid + 1, right);
        return mergeTwoLists(l1, l2);
    }
};
```

Saved | Ln 31, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

lists =