Assignment- 3

Advanced Programming Lab

Aayush Tewari
UID-22BCS11444

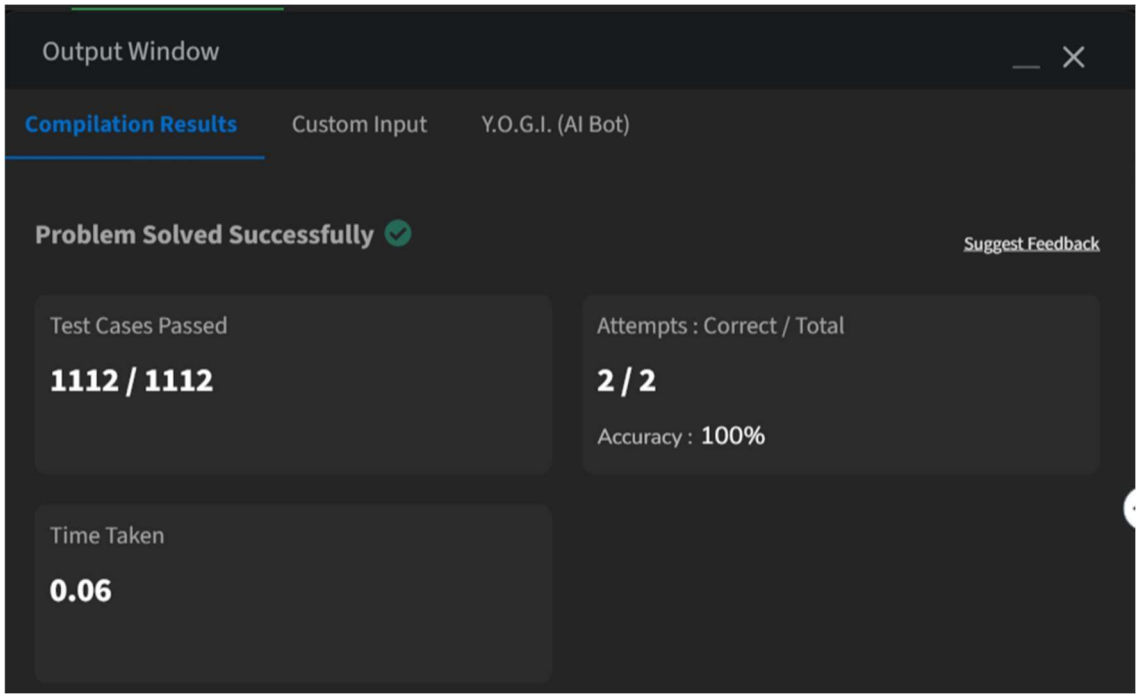
Question 1. Given a linked list. Print all the elements of the linked list separated by space followed.(PRINT LINKEDLIST)

Solution 1. Code Snippet:

C++ (g++ 5.4) Start Timer

```cpp
// } Driver Code Ends
/*
struct Node {
    int data;
    struct Node* next;

    Node(int x) {
        data = x;
        next = nullptr;
    }
};
*/
/*
    Print elements of a linked list on console
    Head pointer input could be NULL as well for empty list
*/

class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        // your code goes here
        Node *le=head;
        while(le!=NULL) {
            cout<<le->data<<" ";
            le=le->next;
        }
    }
};
// } Driver Code Ends
```

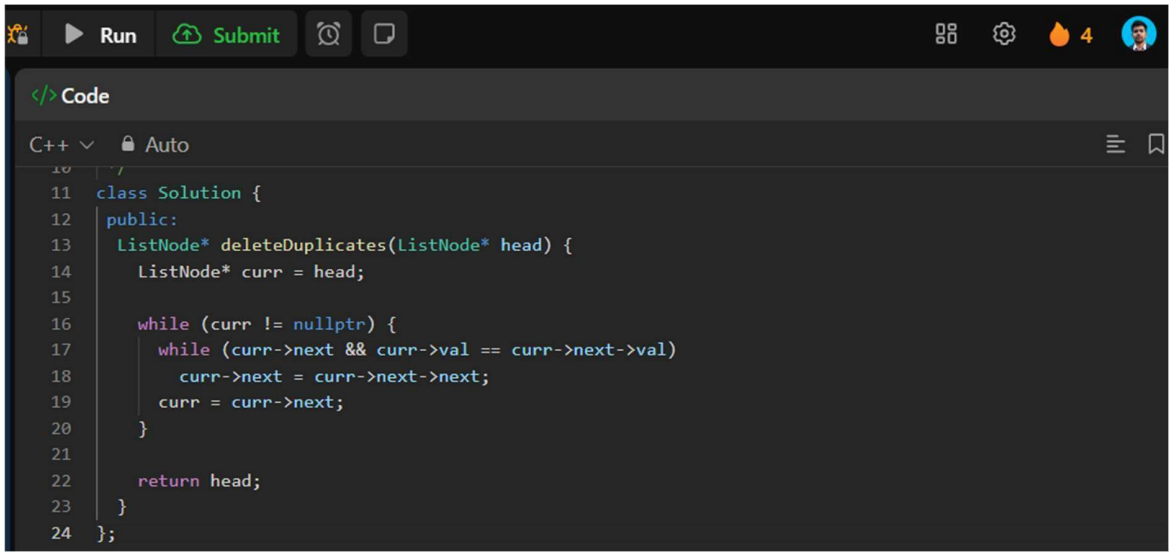
OUTPUT:



Question 2. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.(REMOVE DUPLICATES FROM LINKEDLIST)

Solution 2.

Code Snippet:



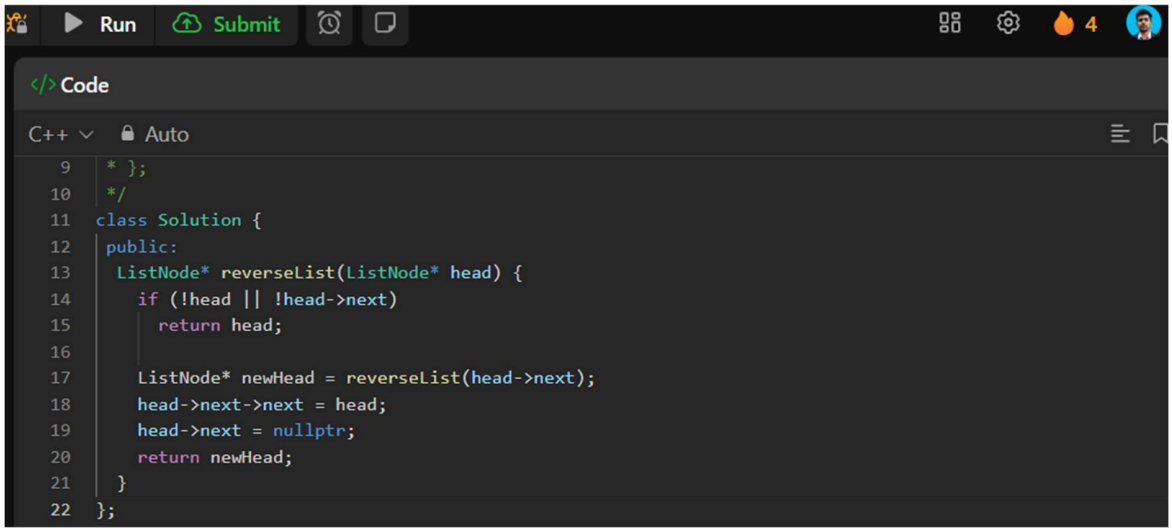

```
class Solution {
 public:
  ListNode* deleteDuplicates(ListNode* head) {
    ListNode* curr = head;

    while (curr != nullptr) {
      while (curr->next && curr->val == curr->next->val)
        curr->next = curr->next->next;
      curr = curr->next;
    }

    return head;
  }
};
```

Output:

Question 3. Given the head of a singly linked list, reverse the list, and return the reversed list.(REVERSE LINKEDLIST)

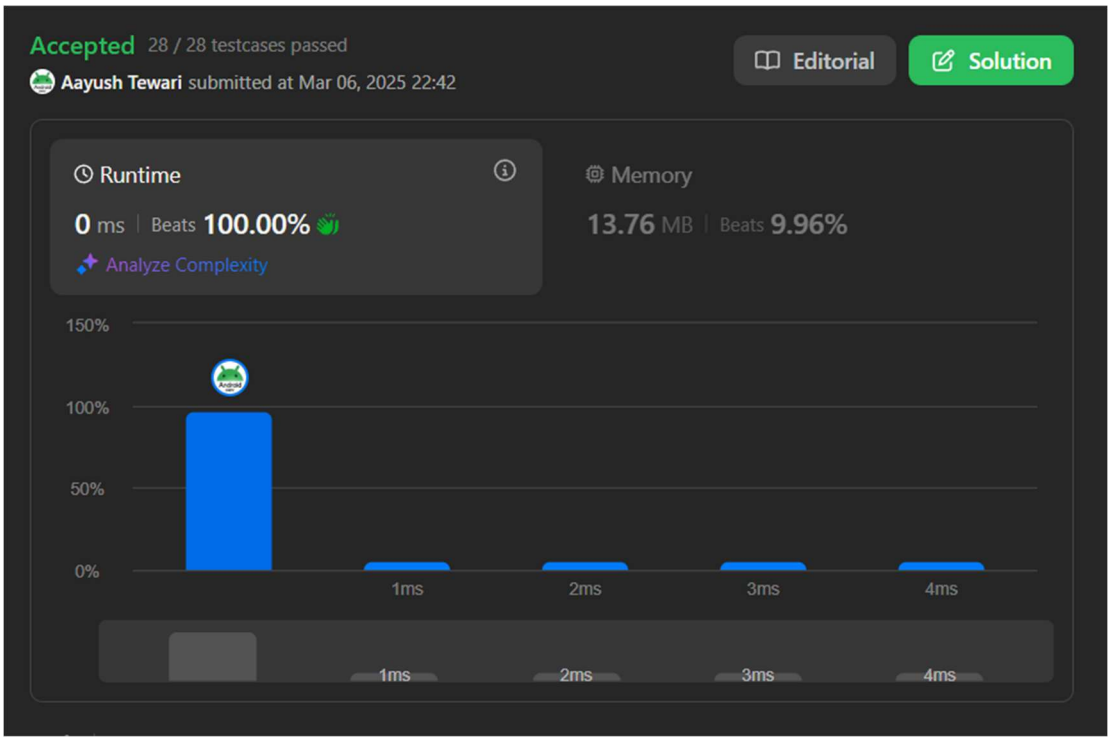
Solution 3.

Code Snippet




```
 * };
 */
class Solution {
 public:
  ListNode* reverseList(ListNode* head) {
    if (!head || !head->next)
      return head;

    ListNode* newHead = reverseList(head->next);
    head->next->next = head;
    head->next = nullptr;
    return newHead;
  }
};
```

Output:



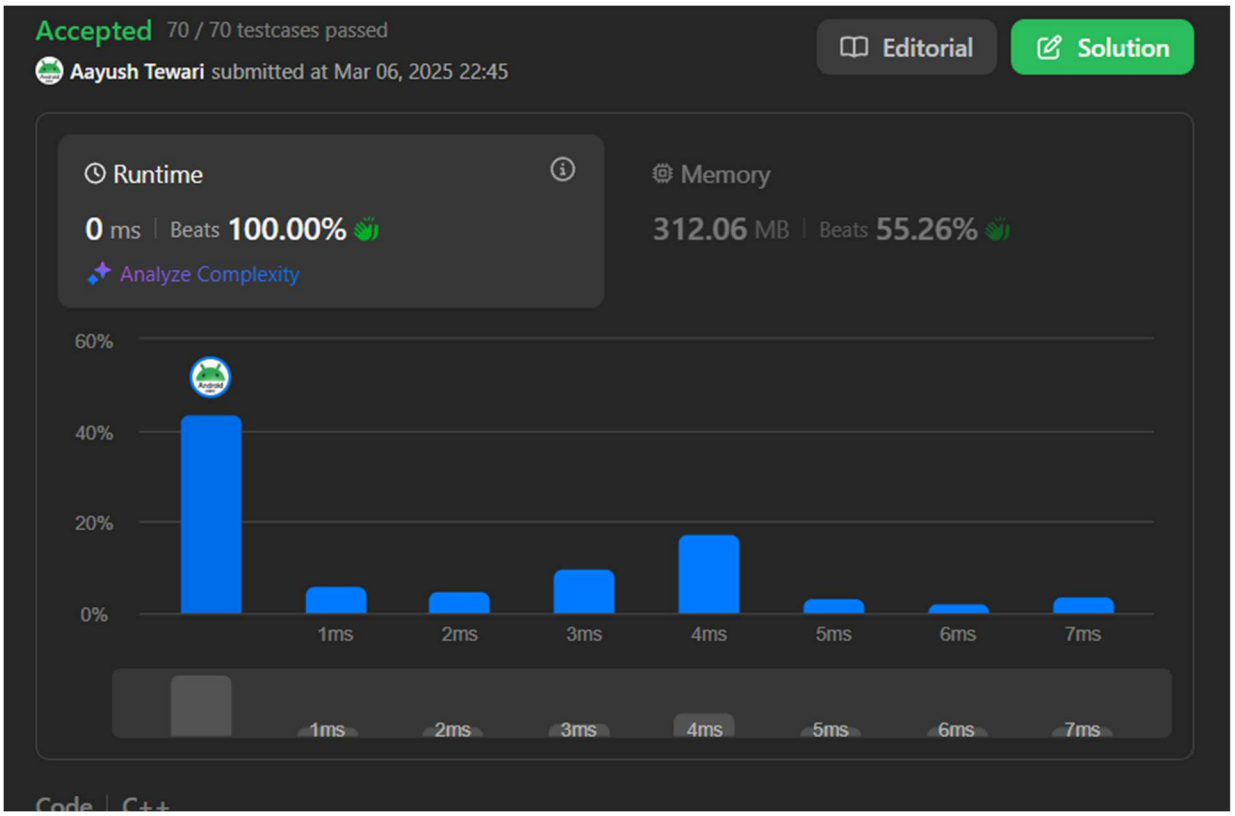
Question 4. You are given the head of a linked list. **Delete** the **middle node**, and return *the* head *of the modified linked list*.(**Delete the Middle Node of a Linked List**)

Solution 4.

Code Snippet:

```
class Solution {
 public:
  ListNode* deleteMiddle(ListNode* head) {
    if (!head || !head->next) return nullptr; // If there's only one node, return nullptr.

    ListNode* slow = head;
    ListNode* fast = head;
    ListNode* prev = nullptr;

    while (fast && fast->next) {
      prev = slow;
      slow = slow->next;
      fast = fast->next->next;
    }
    prev->next = slow->next;
    return head;
  }
};
```

Output:



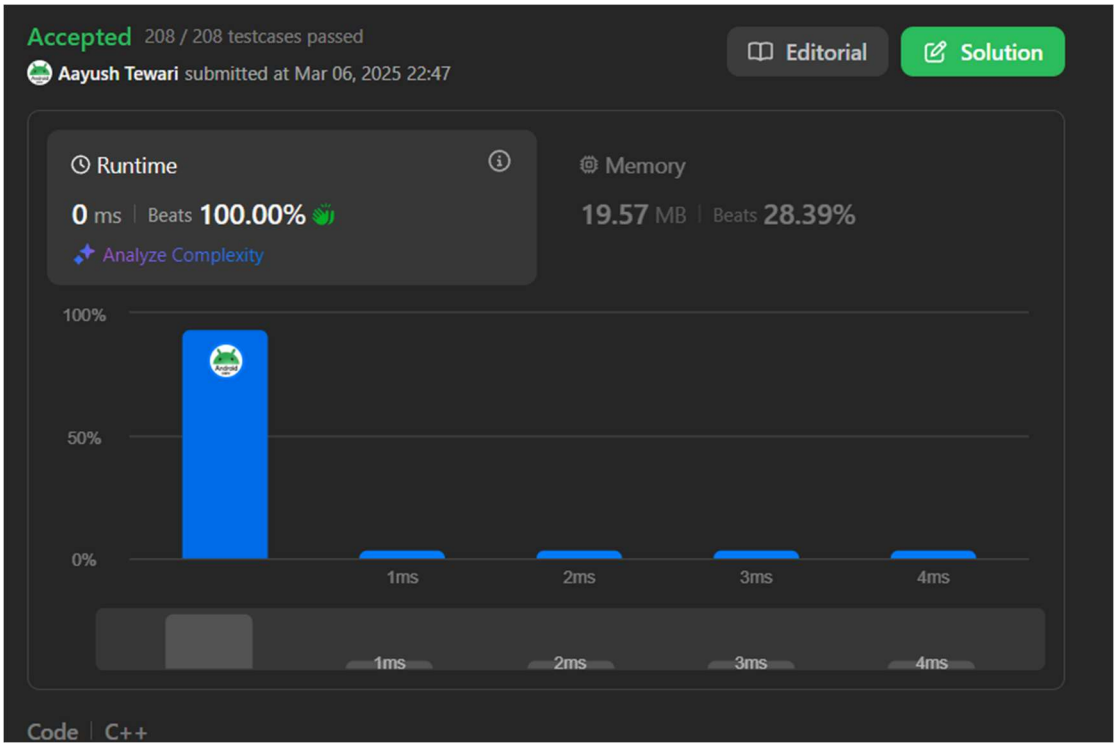
Question 5. You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.(**Merge Two Sorted Lists**)

Solution 5:

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode *head = new ListNode(-1);
        ListNode *cur = head;

        while(l1 != nullptr && l2 != nullptr){
            if(l1->val <= l2->val){
                cur->next = l1;
                l1 = l1->next;
            }else{
                cur->next = l2;
                l2 = l2->next;
            }
            cur = cur->next;
        }

        if(l1 != nullptr) cur->next = l1;
        else if(l2 != nullptr) cur->next = l2;

        return head->next;
    }
};
```

.

Output:



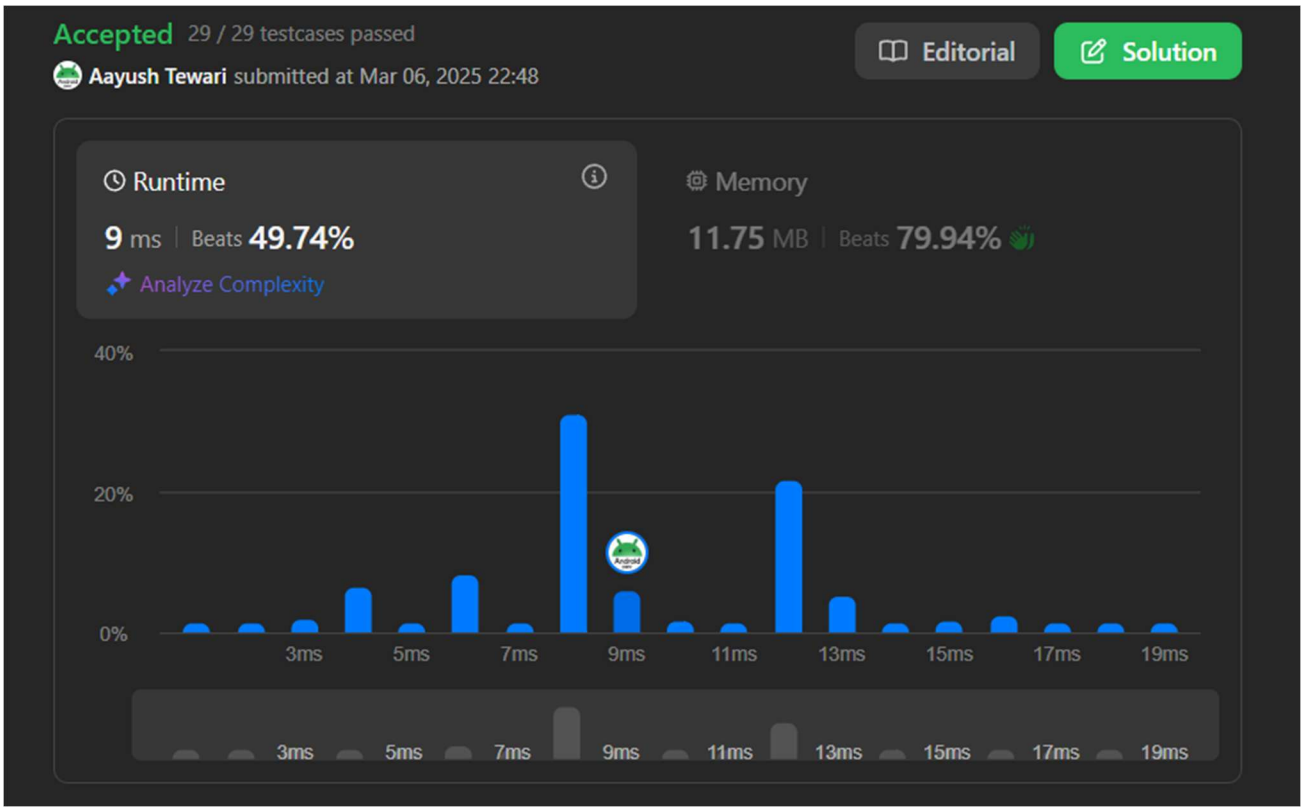
Question 6. Linked List Cycle

Solution 6.

Code Snippet:

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;

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

            if (slow == fast) return true;
        }

        return false;
    }
};
```

Output:



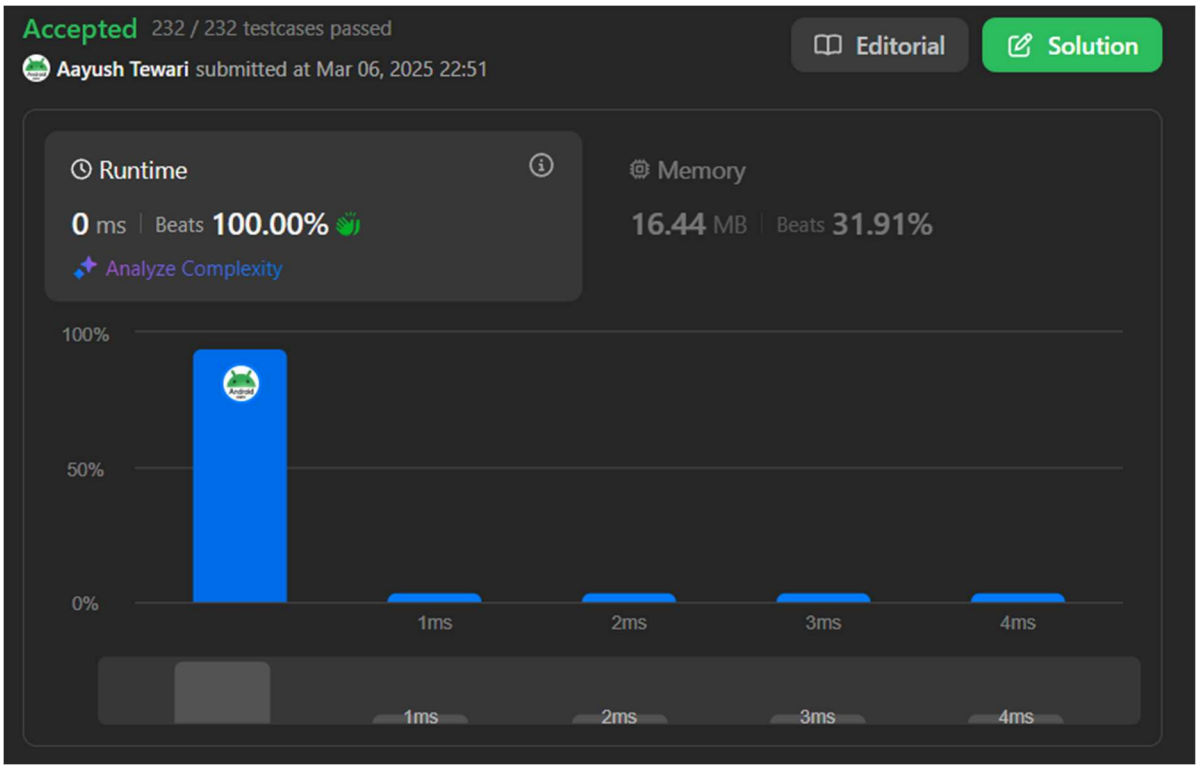
Question 7.Given the head of a linked list, rotate the list to the right by k places.(Rotate List)

Solution 7.

Code Snippet

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

    ListNode* tail;
    int length = 1;
    for (tail = head; tail->next; tail = tail->next)
      ++length;
    tail->next = head;  // Circle the list.

    const int t = length - k % length;
    for (int i = 0; i < t; ++i)
      tail = tail->next;
    ListNode* newHead = tail->next;
    tail->next = nullptr;

    return newHead;
  }
};
```

Output:



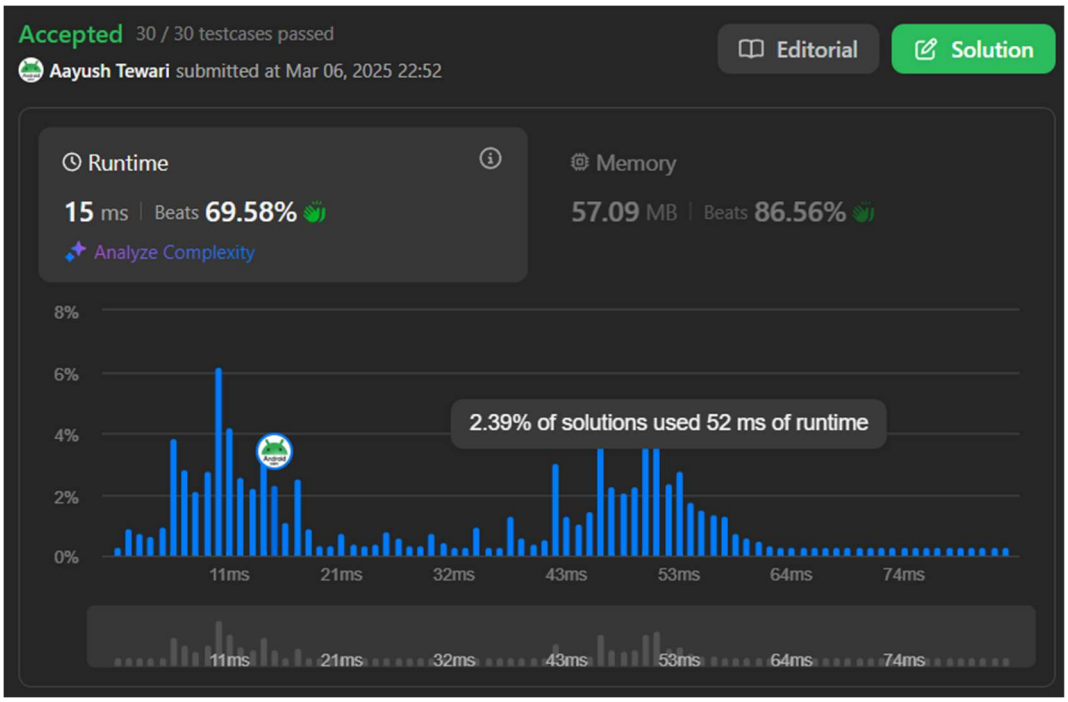
Question 8. Given the head of a linked list, return the list after sorting it in **ascending order**.(**Sort List**)

Solution 8.

Code Snippet:

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) {
            return head;
        }
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* l1 = head;
        ListNode* l2 = slow->next;
        slow->next = nullptr;
        l1 = sortList(l1);
        l2 = sortList(l2);
        ListNode* dummy = new ListNode();
        ListNode* tail = dummy;
        while (l1 && l2) {
            if (l1->val <= l2->val) {
                tail->next = l1;
                l1 = l1->next;
            } else {
                tail->next = l2;
                l2 = l2->next;
            }
            tail = tail->next;
        }
        tail->next = l1 ? l1 : l2;
        return dummy->next;
    }
};
```

Output:



Question 9. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

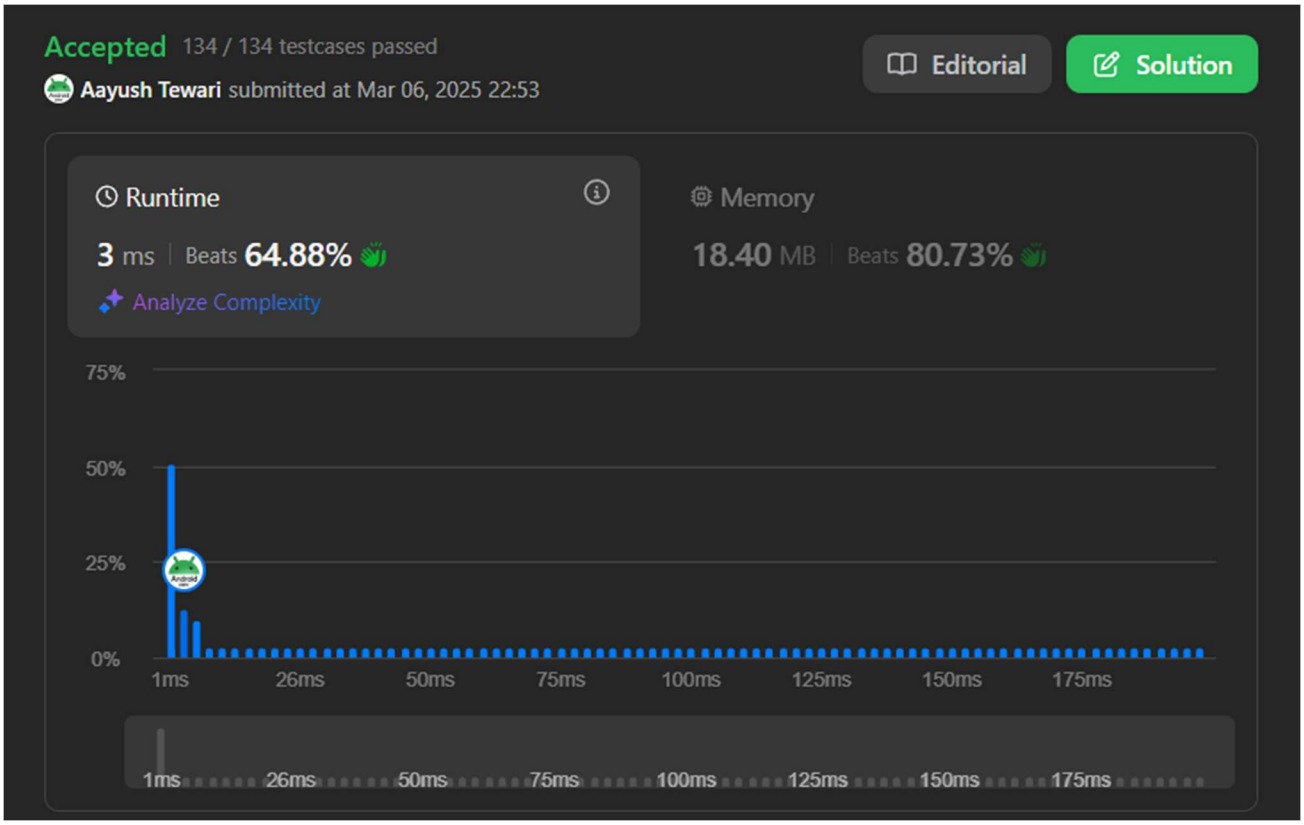
Merge all the linked-lists into one sorted linked-list and return it.(**Merge k Sorted Lists***)*

Solution 9.

Code Snippet:

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        auto cmp = [](ListNode* a, ListNode* b) { return a->val > b->val; };
        priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> pq;
        for (auto head : lists) {
            if (head) {
                pq.push(head);
            }
        }
        ListNode* dummy = new ListNode();
        ListNode* cur = dummy;
        while (!pq.empty()) {
            ListNode* node = pq.top();
            pq.pop();
            if (node->next) {
                pq.push(node->next);
            }
            cur->next = node;
            cur = cur->next;
        }
        return dummy->next;
    }
};
```

Output:


Accepted 134 / 134 testcases passed
Aayush Tewari submitted at Mar 06, 2025 22:53
Editorial
Solution
Runtime
3 ms | Beats 64.88%
Analyze Complexity
Memory
18.40 MB | Beats 80.73%
75%
50%
25%
0%
1ms
26ms
50ms
75ms
100ms
125ms
150ms
175ms