Name: Biswapratic Parija

UID : 22BCS12444

Sec : 607 ‘B’

Subject : AP

1.Print Linked List

```cpp
// } Driver Code Ends


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

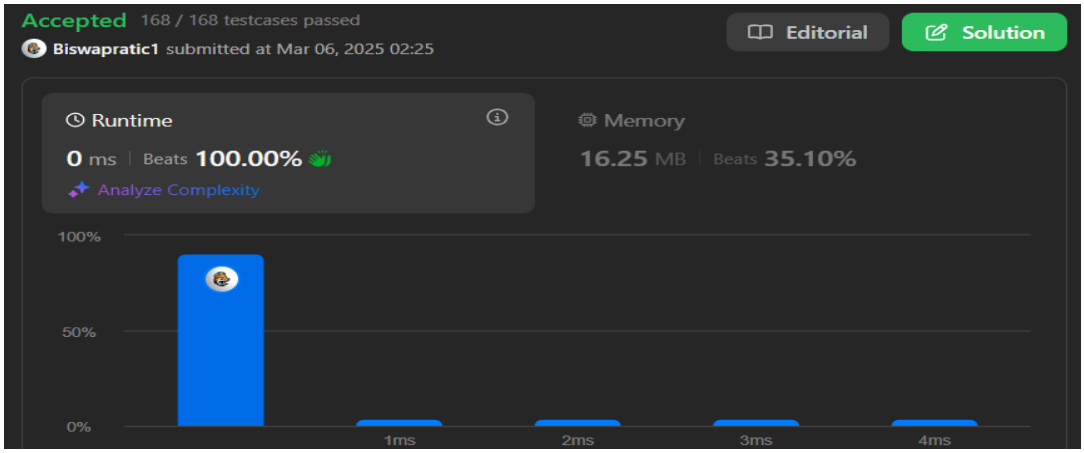
My Submissions All Submissions

Refresh

Time (IST)	Status	Marks	Lang	Test Cases	Code
2025-02-17 15:51:09	Correct	1	cpp	1112 / 1112	View

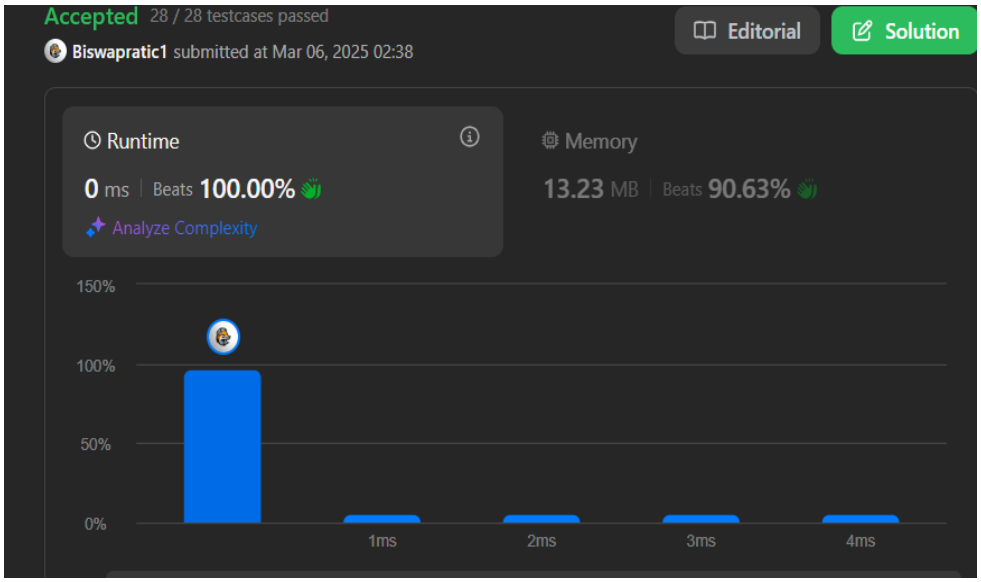
2. Remove duplicates from a sorted list

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

        while (current && current->next) {
            if (current->val == current->next->val) {
                current->next = current->next->next;
            } else {
                current = current->next;
            }
        }

        return head;
    }
};
```



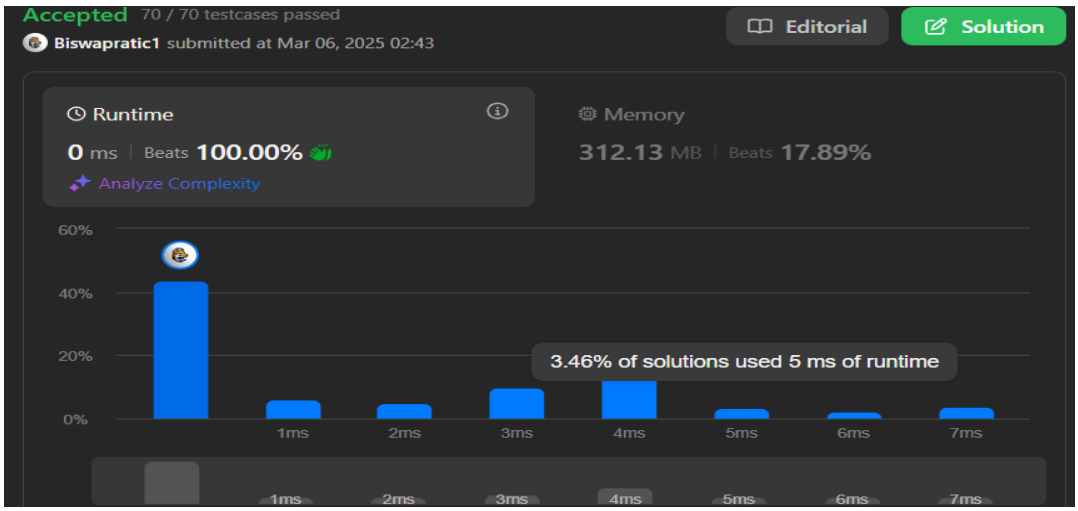
3. Reverse a linked list

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* current = head;

        while (current) {
            ListNode* next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }

        return prev;
    }
};
```



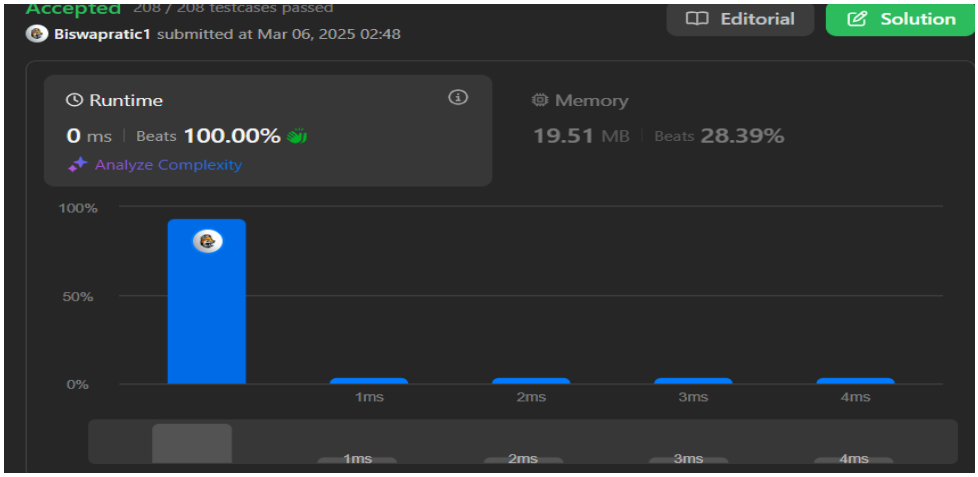
4. Delete middle node of a list

```cpp
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) 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;
        delete slow;
        return head;
```



5. Merge two sorted linked lists

```cpp
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* current = &dummy;

        while (list1 && list2) {
            if (list1->val < list2->val) {
                current->next = list1;
                list1 = list1->next;
            } else {
                current->next = list2;
                list2 = list2->next;
            }
            current = current->next;
        }

        current->next = list1 ? list1 : list2;
        return dummy.next;
    }
};
```

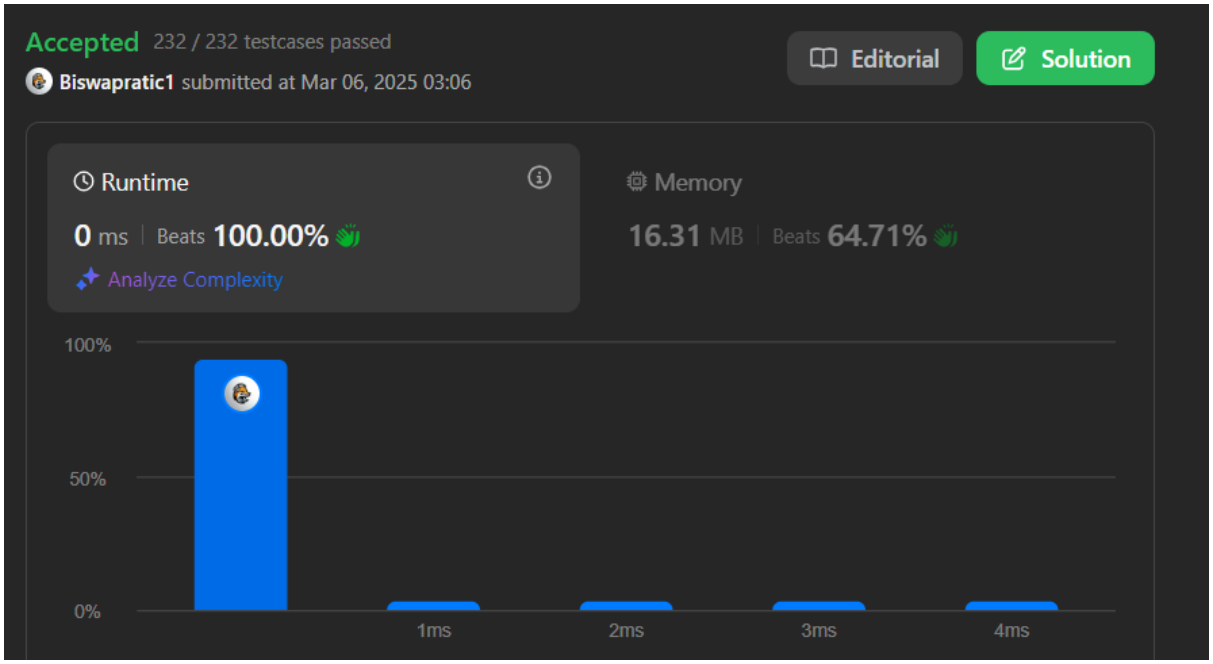


6. Detect a cycle in a linked list

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

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

            if (slow == fast) return true;
        }
        return false;
    }
};
```

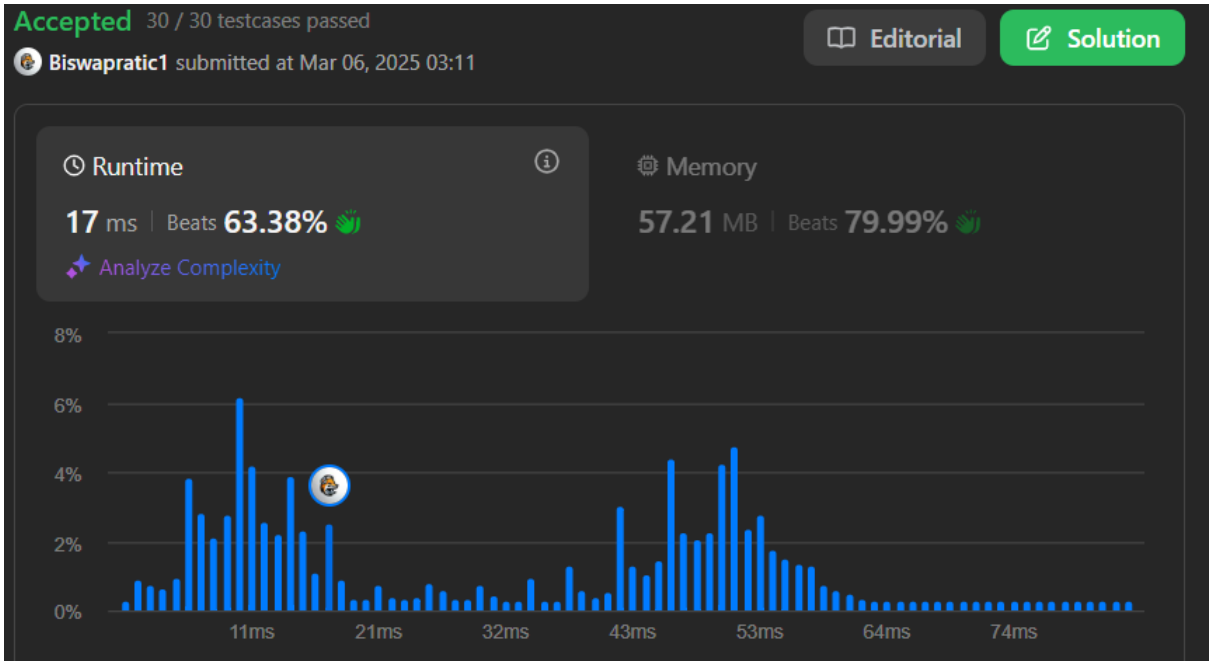
7. Rotate a list

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

        int length = 1;
        ListNode* tail = head;

        while (tail->next) {
            tail = tail->next;
            length++;
        }

        k = k % length;
        if (k == 0) return head;
        tail->next = head;
        int stepsToNewHead = length - k;
        ListNode* newTail = head;

        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        return newHead;
    }
};
```



8. Sort List

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* mid = getMiddle(head);
        ListNode* left = head;
        ListNode* right = mid->next;
        mid->next = nullptr;
        left = sortList(left);
        right = sortList(right);
        return merge(left, right);
    }
private:
    ListNode* getMiddle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
    ListNode* merge(ListNode* list1, ListNode* list2) {
        ListNode dummy(0);
        ListNode* current = &dummy;
        while (list1 && list2) {
            if (list1->val < list2->val) {
                current->next = list1;
                list1 = list1->next;
            } else {
                current->next = list2;
                list2 = list2->next;
            }
            current = current->next;
        }
        current->next = list1 ? list1 : list2;
        return dummy.next;
    }
};
```



9. Merge k sorted lists

```cpp
class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val;
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
        for (auto list : lists) {
            if (list) minHeap.push(list);
        }
        ListNode dummy(0);
        ListNode* tail = &dummy;

        while (!minHeap.empty()) {
            ListNode* node = minHeap.top();
            minHeap.pop();
            tail->next = node;
            tail = tail->next;

            if (node->next) minHeap.push(node->next);
        }
        return dummy.next;
    }
};
```