

Name: Biswapratic Parija

UID : 22BCS12444

Sec : 607 'B'


Subject : AP

1. Print Linked List

```
// } 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
```

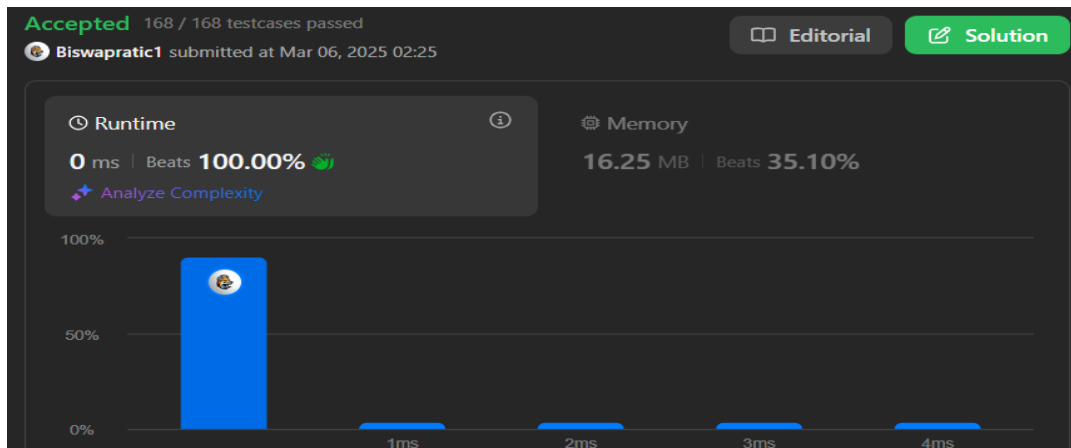
My Submissions All Submissions

 Refresh

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

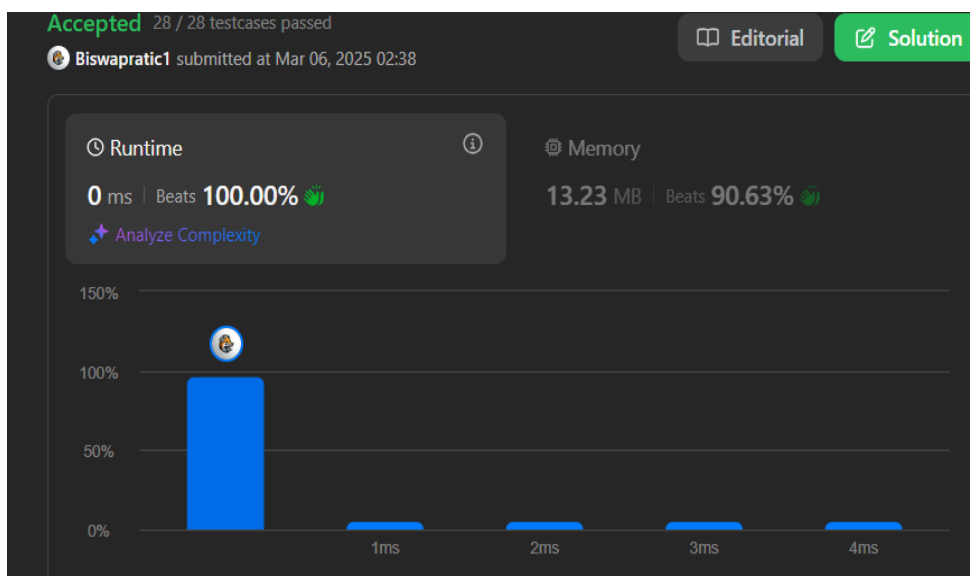
2. Remove duplicates from a sorted list

```
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         ListNode* current = head;
15
16         while (current && current->next) {
17             if (current->val == current->next->val) {
18                 current->next = current->next->next;
19             } else {
20                 current = current->next;
21             }
22         }
23
24         return head;
25     }
26 };
```



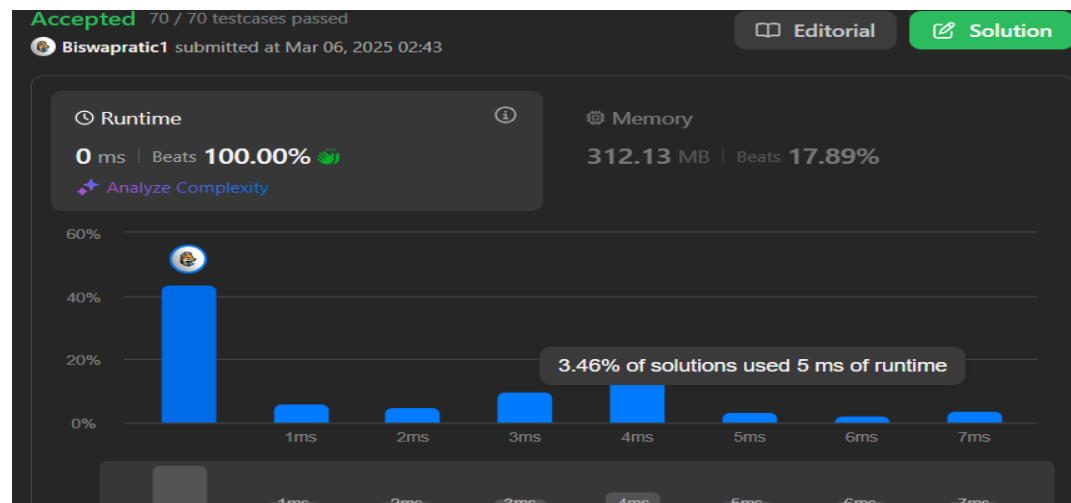
3. Reverse a linked list

```
1 class Solution {
2 public:
3     ListNode* reverseList(ListNode* head) {
4         ListNode* prev = nullptr;
5         ListNode* current = head;
6
7         while (current) {
8             ListNode* next = current->next;
9             current->next = prev;
10            prev = current;
11            current = next;
12        }
13
14        return prev;
15    }
16 };
17
```



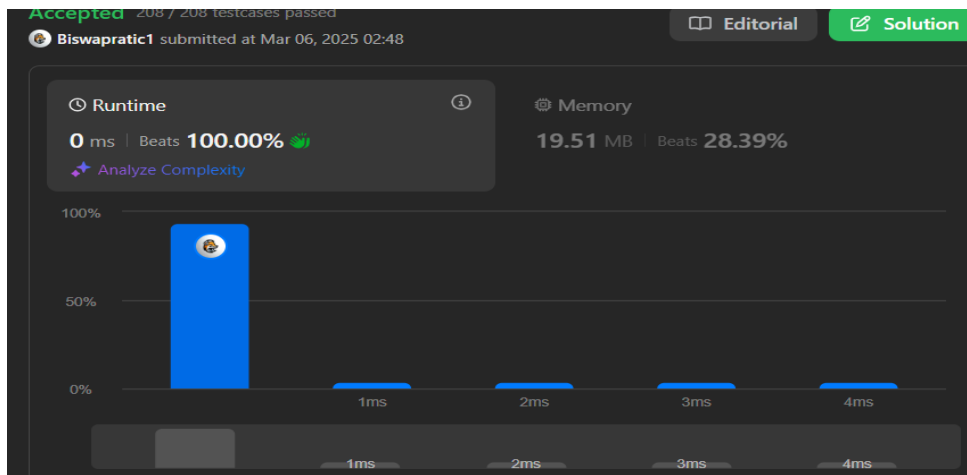
4. Delete middle node of a list

```
1  class Solution {
2  public:
3      ListNode* deleteMiddle(ListNode* head) {
4          if (!head || !head->next) return nullptr;
5
6          ListNode* slow = head;
7          ListNode* fast = head;
8          ListNode* prev = nullptr;
9
10         while (fast && fast->next) {
11             prev = slow;
12             slow = slow->next;
13             fast = fast->next->next;
14         }
15
16         prev->next = slow->next;
17         delete slow;
18         return head;
19     }
20 }
```



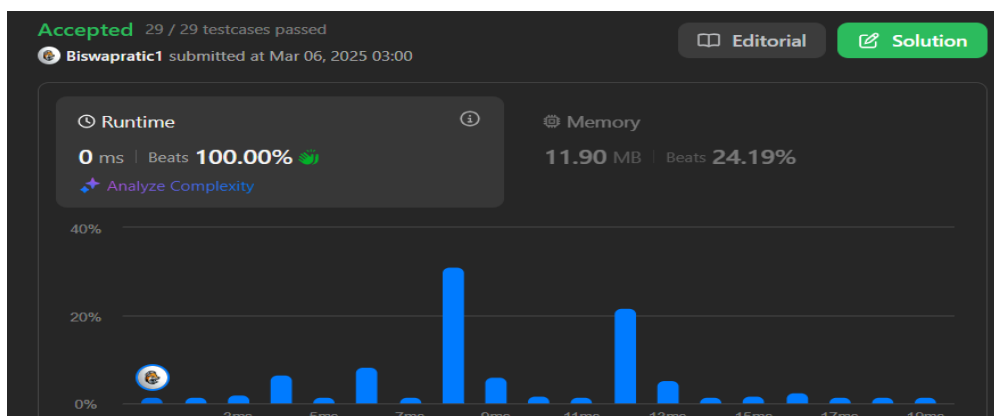
5. Merge two sorted linked lists

```
1  class Solution {
2  public:
3      ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
4          ListNode dummy(0);
5          ListNode* current = &dummy;
6
7          while (list1 && list2) {
8              if (list1->val < list2->val) {
9                  current->next = list1;
10                 list1 = list1->next;
11             } else {
12                 current->next = list2;
13                 list2 = list2->next;
14             }
15             current = current->next;
16         }
17
18         current->next = list1 ? list1 : list2;
19         return dummy.next;
20     }
21 }
```



6. Detect a cycle in a linked list


```
1 class Solution {  
2 public:  
3     bool hasCycle(ListNode *head) {  
4         ListNode *slow = head, *fast = head;  
5  
6         while (fast && fast->next) {  
7             slow = slow->next;  
8             fast = fast->next->next;  
9  
10            if (slow == fast) return true;  
11        }  
12        return false;  
13    }  
14 };  
15
```




7. Rotate a list


```
1  class Solution {
2  public:
3      ListNode* rotateRight(ListNode* head, int k) {
4          if (!head || !head->next || k == 0) return head;
5
6          int length = 1;
7          ListNode* tail = head;
8
9          while (tail->next) {
10             tail = tail->next;
11             length++;
12         }
13
14         k = k % length;
15         if (k == 0) return head;
16         tail->next = head;
17         int stepsToNewHead = length - k;
18         ListNode* newTail = head;
19
20         for (int i = 1; i < stepsToNewHead; i++) {
21             newTail = newTail->next;
22         }
23         ListNode* newHead = newTail->next;
24         newTail->next = nullptr;
25         return newHead;
26     }
27 }
```

Accepted 232 / 232 testcases passed

 Biswapratic1 submitted at Mar 06, 2025 03:06


 Editorial


 Solution


 Runtime

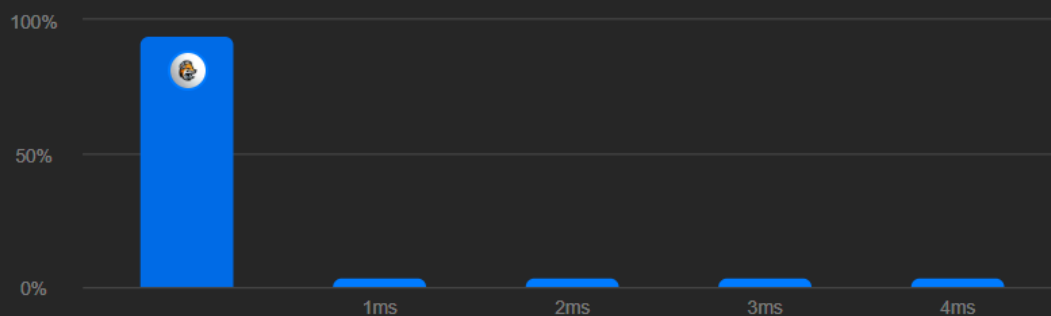


0 ms | Beats 100.00% 

 [Analyze Complexity](#)

 Memory


16.31 MB | Beats 64.71% 





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;
    }
};
```

Accepted 30 / 30 testcases passed

 Biswapratric1 submitted at Mar 06, 2025 03:11

 Editorial

 Solution

⌚ Runtime

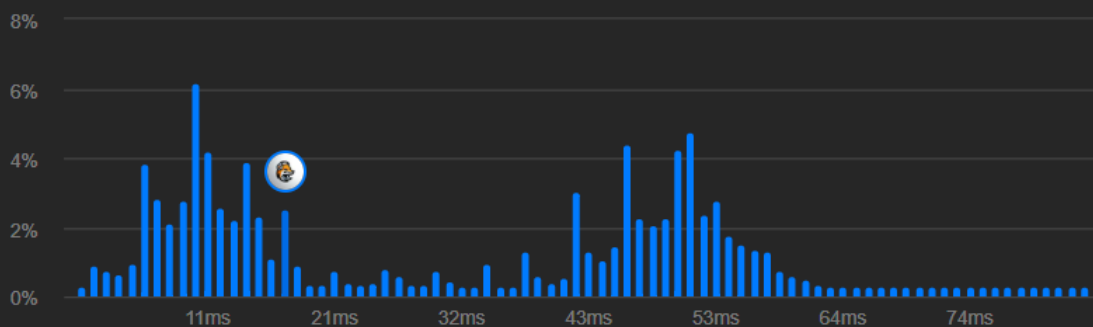


17 ms | Beats 63.38% 🌱

🔮 Analyze Complexity

💾 Memory


57.21 MB | Beats 79.99% 🌱





9. Merge k sorted lists


```
2 class Solution {
3 public:
4     struct Compare {
5         bool operator()(ListNode* a, ListNode* b) {
6             return a->val > b->val;
7         }
8     };
9     ListNode* mergeKLists(vector<ListNode*>& lists) {
10         priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
11         for (auto list : lists) {
12             if (list) minHeap.push(list);
13         }
14         ListNode dummy(0);
15         ListNode* tail = &dummy;
16
17         while (!minHeap.empty()) {
18             ListNode* node = minHeap.top();
19             minHeap.pop();
20             tail->next = node;
21             tail = tail->next;
22
23             if (node->next) minHeap.push(node->next);
24         }
25         return dummy.next;
26     }
27 };
```

Accepted 134 / 134 testcases passed

 Biswaprat1 submitted at Mar 06, 2025 03:14


 Editorial


 Solution

 Runtime



0 ms | Beats 100.00% 

 Analyze Complexity

 Memory

18.35 MB | Beats 80.73% 