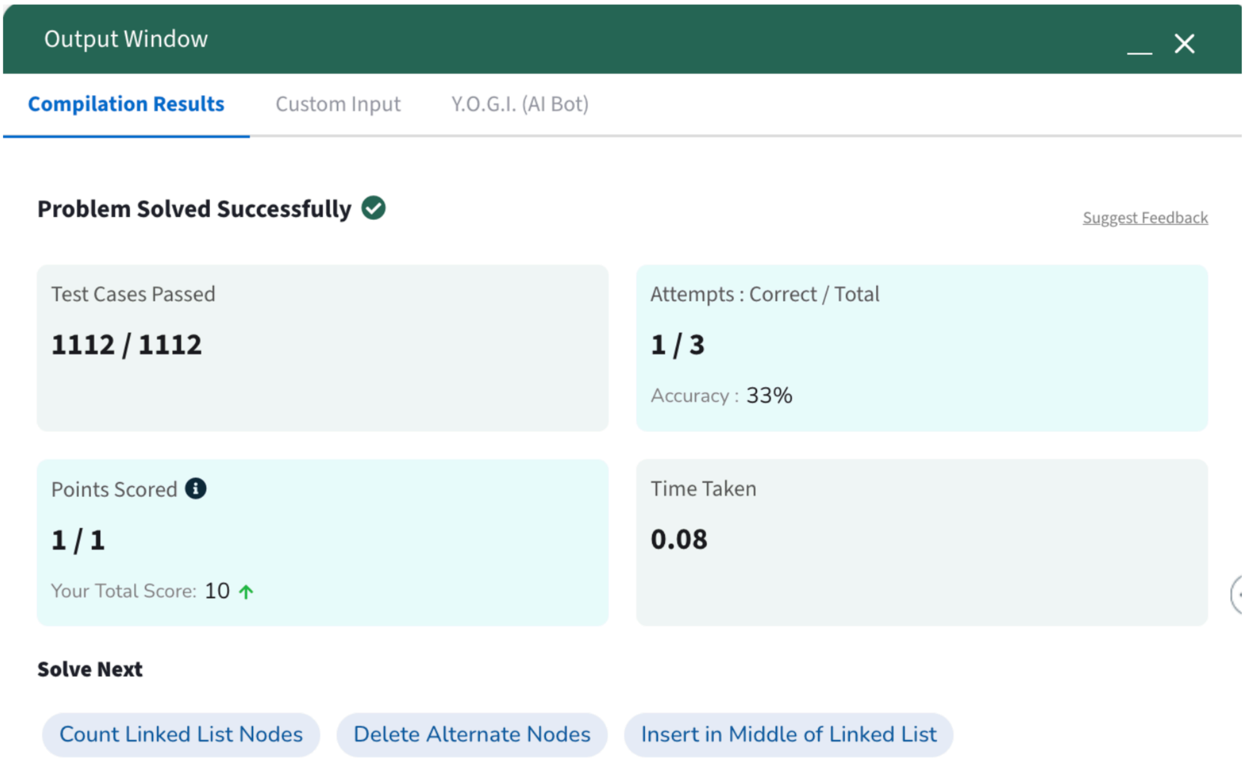
DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Name - Himanshi Gupta
Semester - 6
Subject - Advanced Programming - 2

UID - 22CBS15329
Date - 06-03-2025
Subject Code - 22CSP-351

1. Print Linked List

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



Problem Solved Successfully ✓

Test Cases Passed
1112 / 1112

Attempts : Correct / Total
1 / 3
Accuracy : 33%

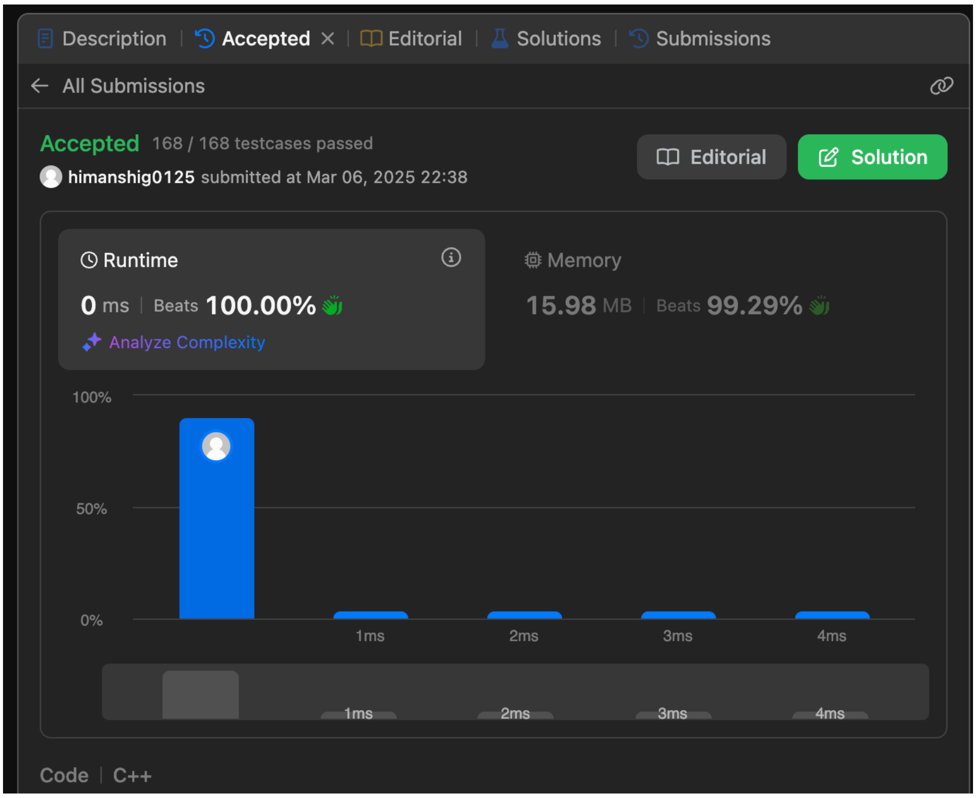
Points Scored
1 / 1
Your Total Score: 10 ↑

Time Taken
0.08

Solve Next

2. Remove duplicates from a sorted list

```
class Solution {
public:
ListNode* deleteDuplicates(ListNode* head) {
ListNode* current = head;
while (current != NULL && current->next != NULL) {
if (current->val == current->next->val) {
current->next = current->next->next; }
else {
Move to the next distinct node
current = current->next;
}}
return head;
}};
```



Accepted 168 / 168 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:38

Runtime

0 ms | Beats 100.00%

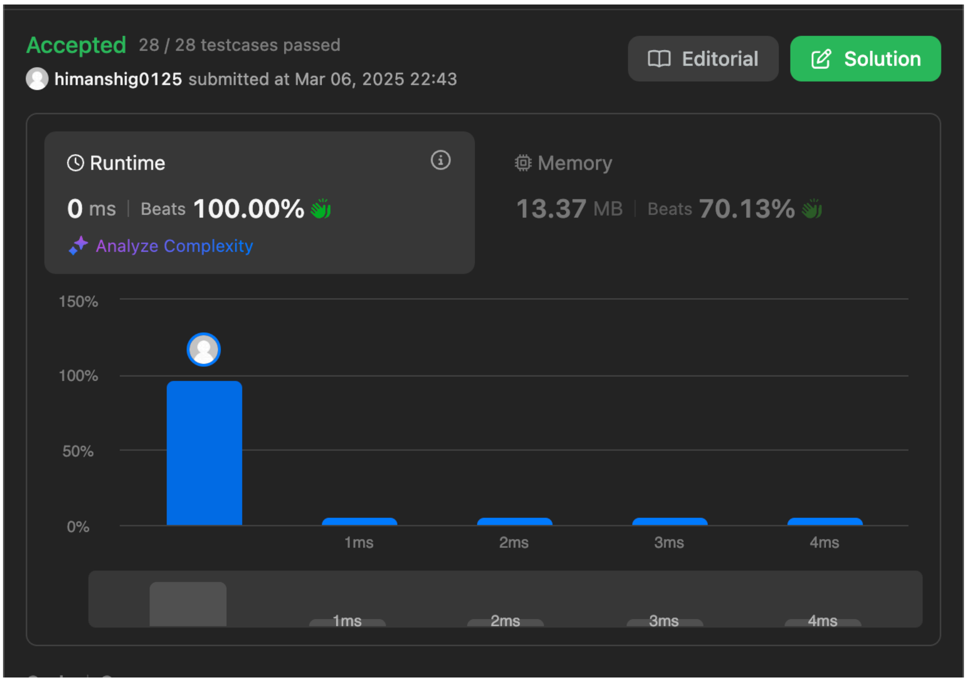
Memory

15.98 MB | Beats 99.29%

Code | C++

3. Reverse a linked list

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



Accepted 28 / 28 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:43

Runtime

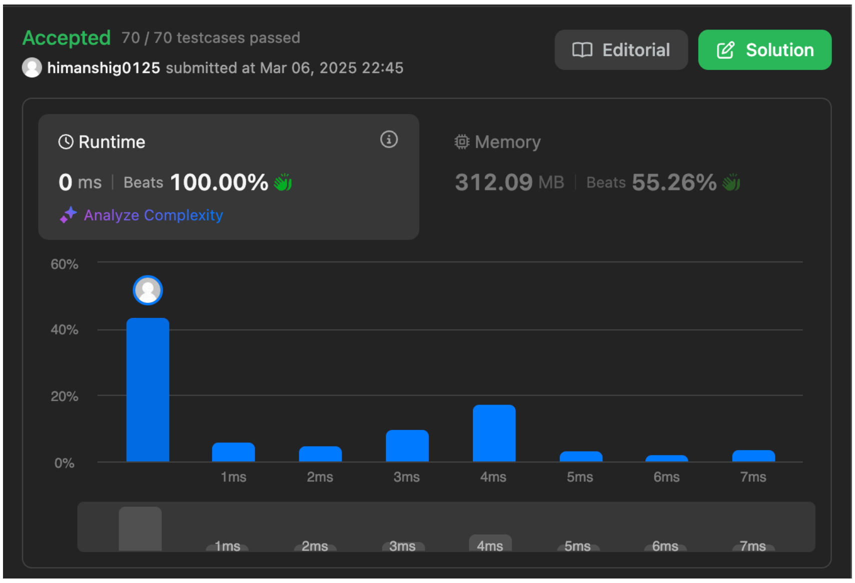
0 ms | Beats 100.00%

Memory

13.37 MB | Beats 70.13%

4. Delete middle node of a list

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



Accepted 70 / 70 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:45

Runtime

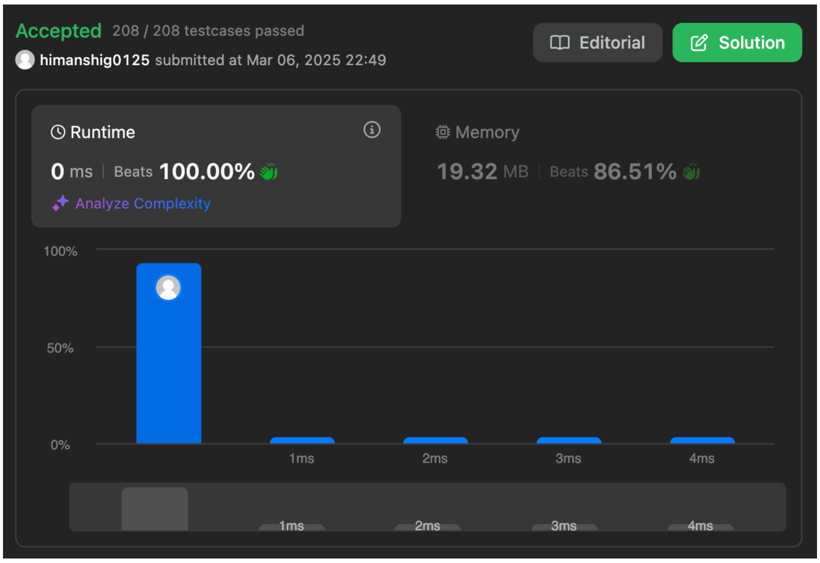
0 ms | Beats 100.00%

Memory

312.09 MB | Beats 55.26%

5. Merge two sorted linked list

```
class Solution {
public:
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
if (!list1) return list2;
if (!list2) return list1;
ListNode* dummy = new ListNode(-1);
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; }
if (list1) current->next = list1;
if (list2) current->next = list2;
return dummy->next;
}};
```



Accepted 208 / 208 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:49

Runtime

0 ms | Beats 100.00%

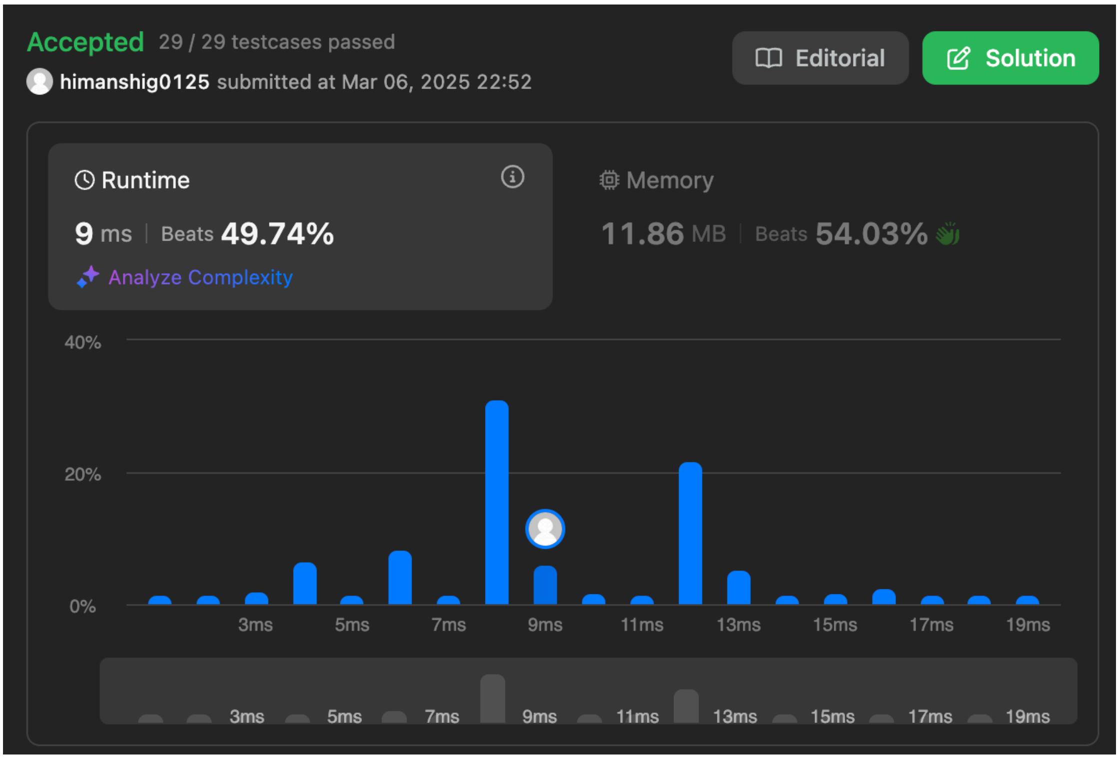
Analyze Complexity

Memory

19.32 MB | Beats 86.51%

6. Detect a cycle in Linked List

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

fast = fast->next->next;

if (slow == fast) return true;

}

return false;

}};
```



Accepted 29 / 29 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:52

Runtime

9 ms | Beats 49.74%

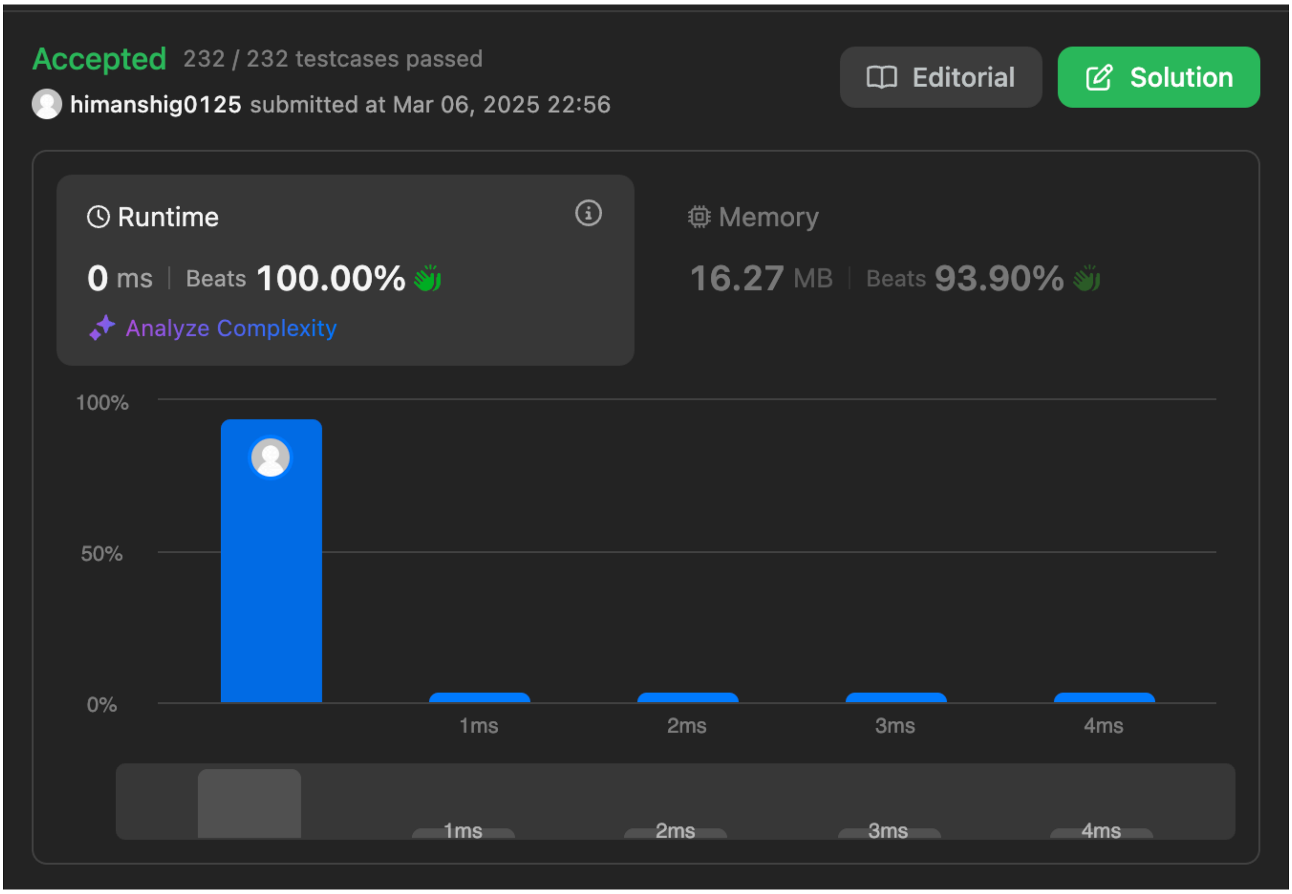
Analyze Complexity

Memory

11.86 MB | Beats 54.03%

7. Rotate a list

```
class Solution {
public:
ListNode* rotateRight(ListNode* head, int k) {
if (!head || !head->next || k == 0) return head;
int n = 1;
ListNode* tail = head;
while (tail->next) {
tail = tail->next;
n++;
}
k = k % n;
if (k == 0) return head;
ListNode* newTail = head;
for (int i = 0; i < n - k - 1; i++) {
newTail = newTail->next; }
ListNode* newHead = newTail->next;
newTail->next = nullptr;
tail->next = head;
return newHead;
} }
```



Accepted 232 / 232 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:56

Runtime

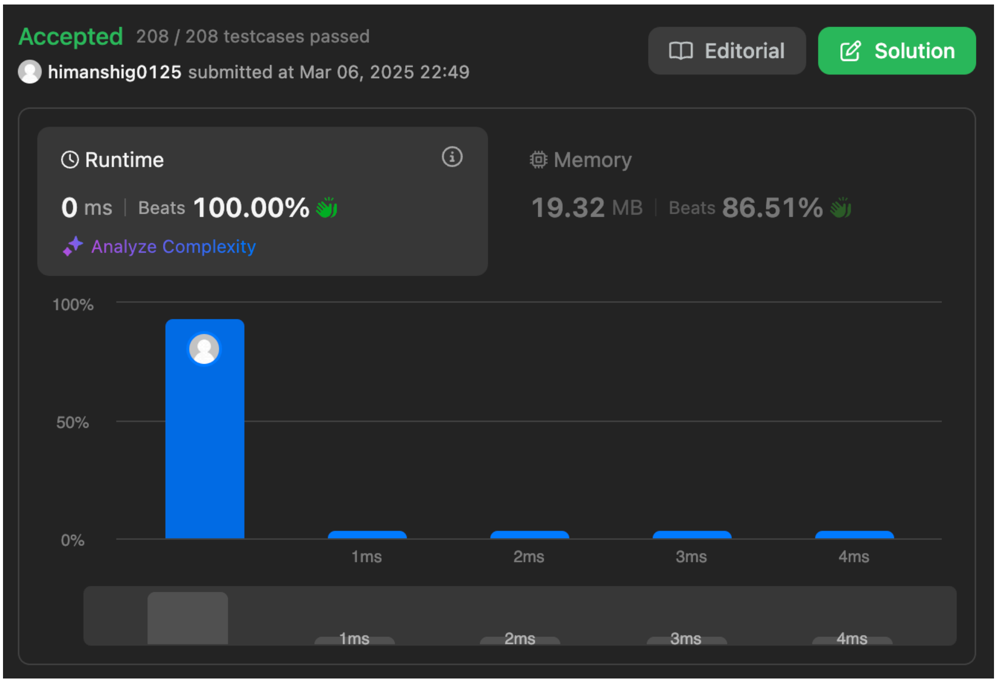
0 ms | Beats 100.00%

Memory

16.27 MB | Beats 93.90%

8. Sort List

```
class Solution { public:
ListNode* merge(ListNode* l1, ListNode* l2) {
ListNode* dummy = new ListNode(0);
ListNode* current = dummy;
while (l1 && l2) {
if (l1->val < l2->val) {
current->next = l1; l1 = l1->next;
} else { current->next = l2;
l2 = l2->next; }
current = current->next; }
if (l1) current->next = l1;
if (l2) current->next = l2; return dummy->next;
}
ListNode* getMid(ListNode* head) {
ListNode* slow = head;
ListNode* fast = head;
ListNode* prev = nullptr;
while (fast && fast->next) {
 prev = slow;
slow = slow->next;
fast = fast->next->next;
}
if (prev) prev->next = nullptr;
return slow;
}
ListNode* sortList(ListNode* head) {

if (!head || !head->next) return head;
ListNode* mid = getMid(head);
ListNode* left = sortList(head);
ListNode* right = sortList(mid);
return merge(left, right);} };
```



Accepted 208 / 208 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:49

Runtime

0 ms | Beats 100.00%

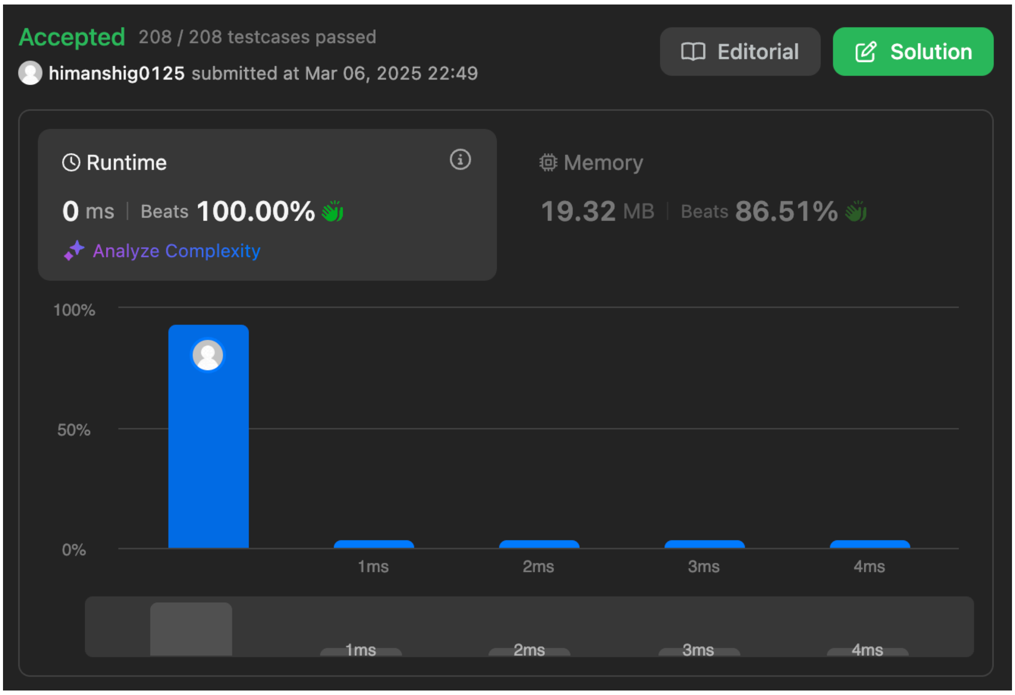
Analyze Complexity

Memory

19.32 MB | Beats 86.51%

3. Merge k sorted lists

```
#include <queue> class Solution { public:

struct Compare {
bool operator()(ListNode* a, ListNode* b) {

return a->val > b->val; // Min-heap based on node values }

};
ListNode* mergeKLists(vector<ListNode*>& lists) {

priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap; for (auto
list : lists) {

if (list) minHeap.push(list); }

ListNode dummy(0); // Dummy node for ease of handling ListNode* tail =
&dummy;
while (!minHeap.empty()) {

ListNode* smallest = minHeap.top(); minHeap.pop();
tail->next = smallest;
tail = tail->next;

if (smallest->next) {

minHeap.push(smallest->next); // Add the next node to the heap }
}
return dummy.next; // Return the merged list }
};
```



Accepted 208 / 208 testcases passed

himanshig0125 submitted at Mar 06, 2025 22:49

Runtime

0 ms | Beats 100.00%

Memory

19.32 MB | Beats 86.51%