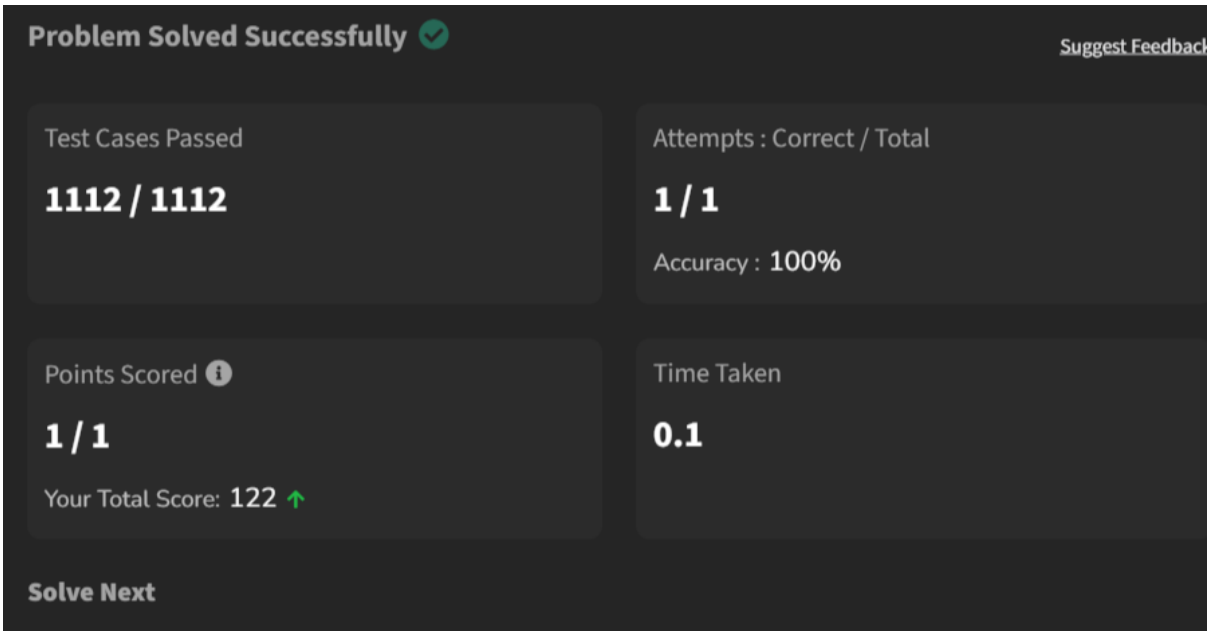
1) ***Print Linked List:***

Code:

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

Output:



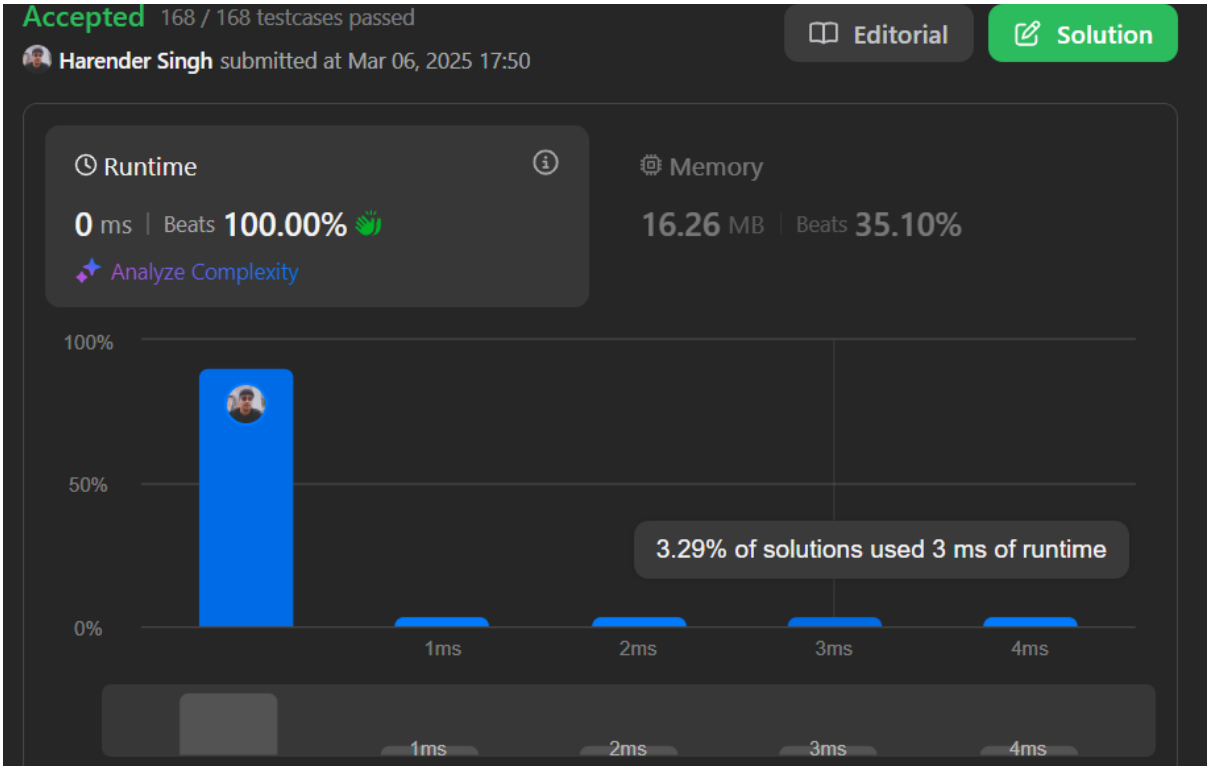
2) **Remove duplicates from a sorted list**

Code:

```
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;
  }
};
```

Output:



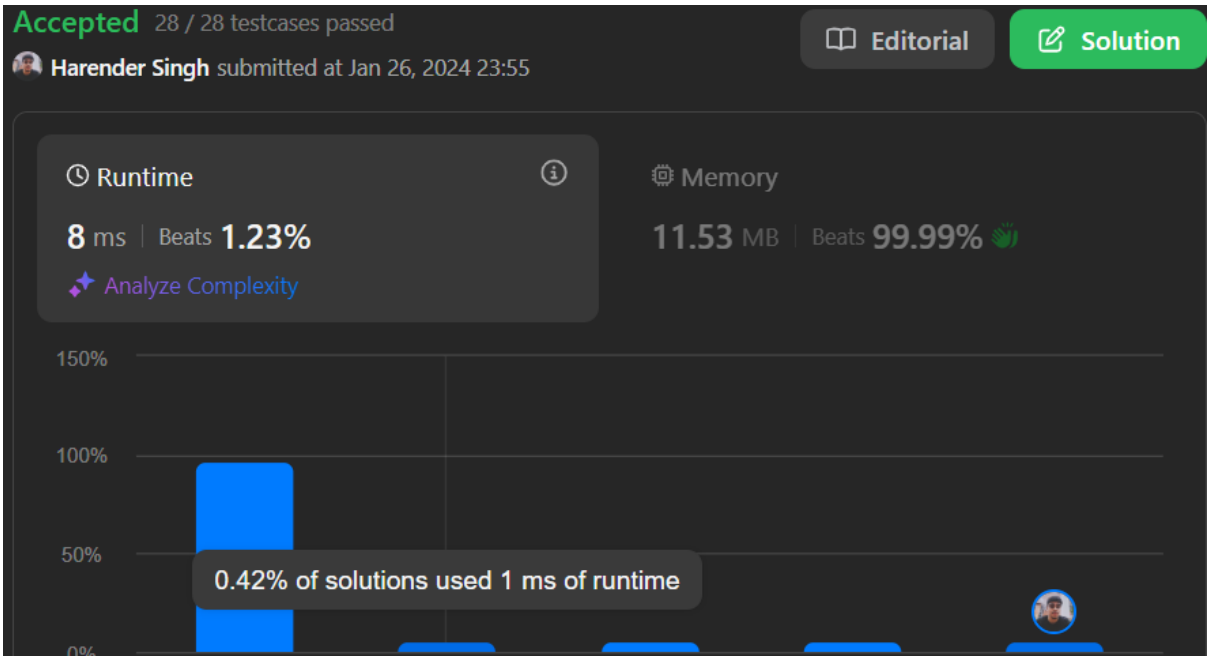
3) **Reverse a linked list:**

Code:

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

```
    }
};
```

Output:



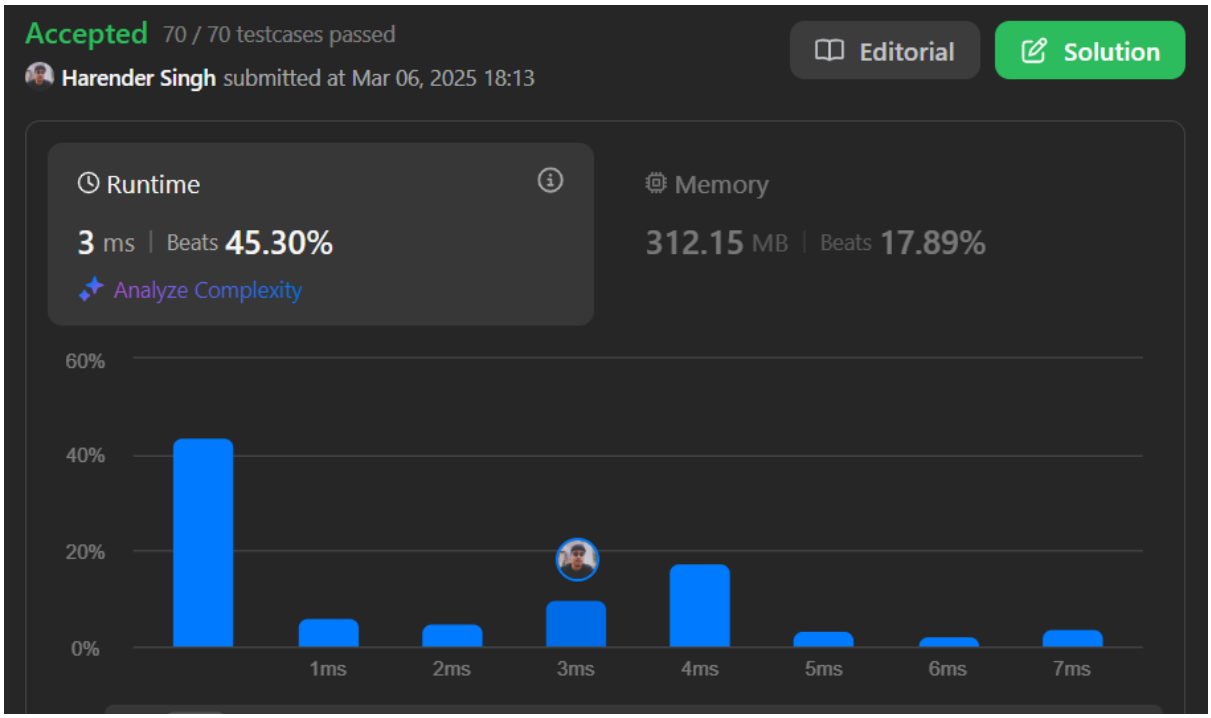
4) Delete the Middle Node of a Linked List

Code:

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

```
    return head;
  }
};
```

Output:

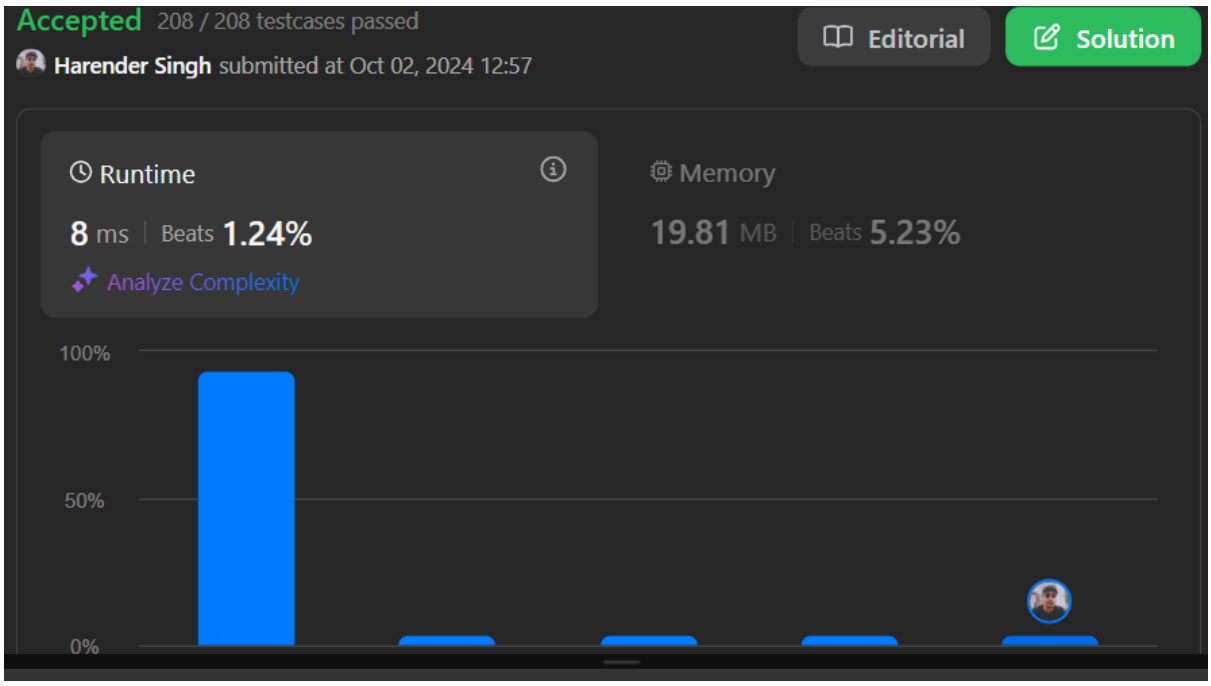


5) **Merge two sorted linked lists**

```
class Solution {
public:
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
  ListNode* ans=new ListNode(-1);
  ListNode*temp=ans;
 ListNode* head1=list1;
  ListNode*head2=list2;
  while(head1!=NULL&&head2!=NULL){
    if(head1->val<head2->val){
      temp->next=head1;
      temp=head1;
```

```
            head1=head1->next;
        }
        else{
            temp->next=head2;
            temp=head2;
            head2=head2->next;
        }
    }
    while(head1!=NULL){
            temp->next=head1;
            temp=head1;
            head1=head1->next;
    }
     while(head2!=NULL){
            temp->next=head2;
            temp=head2;
            head2=head2->next;
    }
    return ans->next;
    }
};
```

Output:

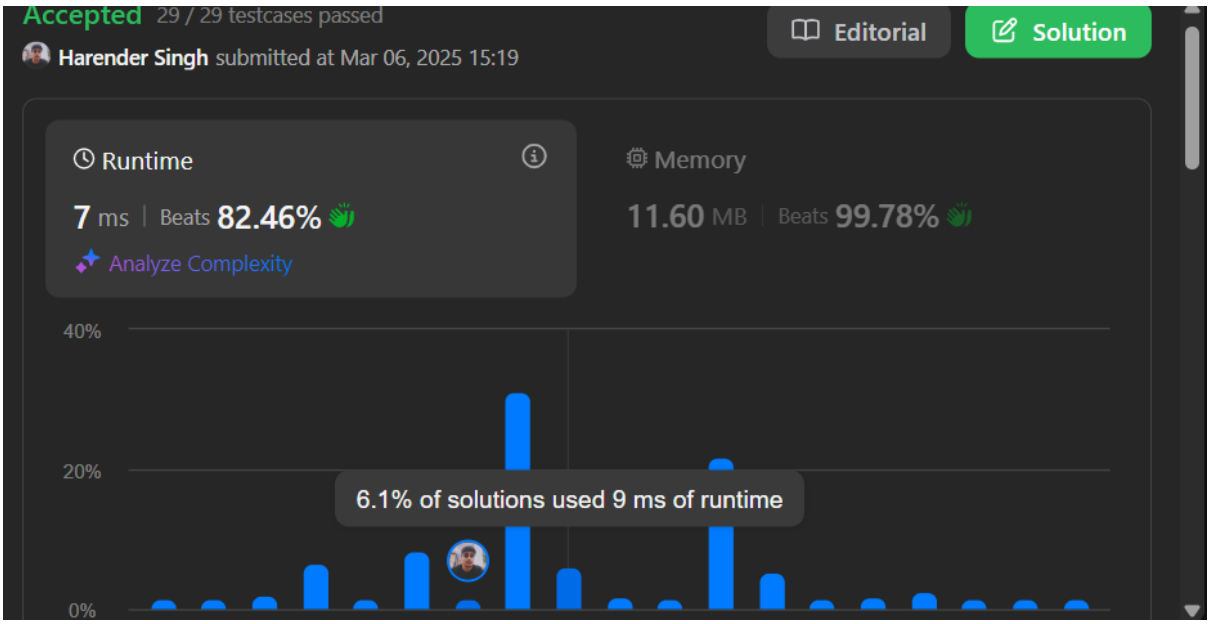


6) Linked List cycle

Code:

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

    }
};
```

Output:



7) **Rotate a list**

Code:

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        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;

        return head;

    }
};
```

Output:

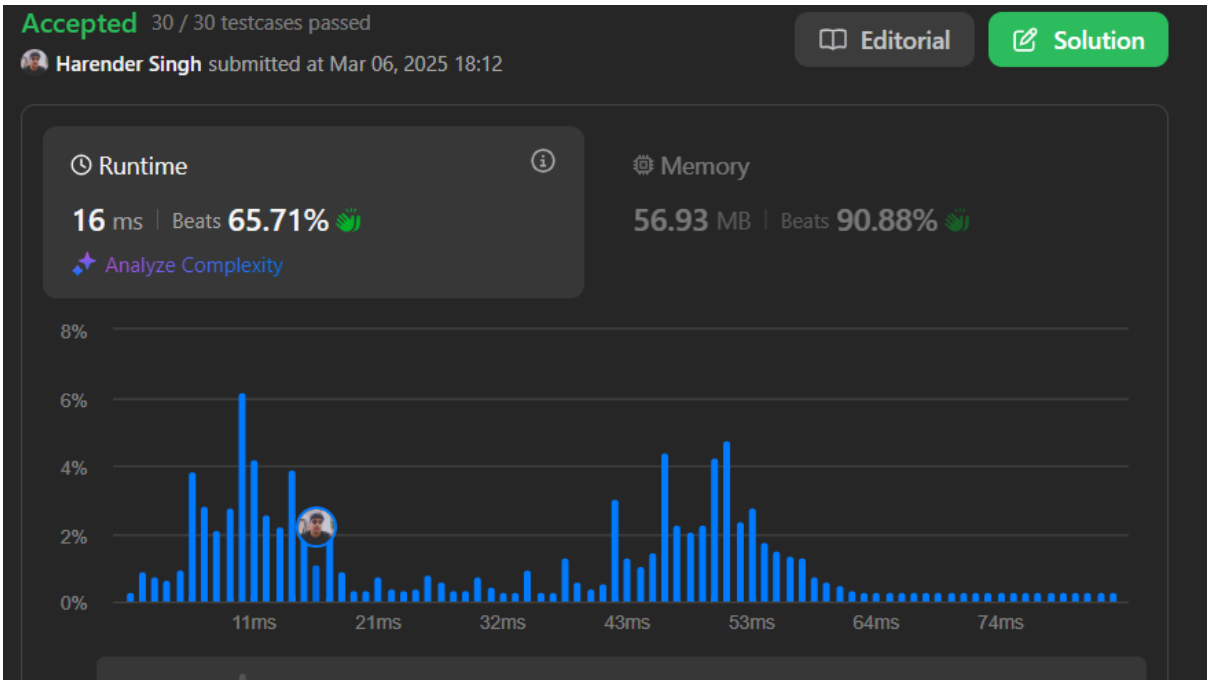
8) **Sort List:**

Code:

```
#include <iostream>
using namespace std;

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* mid = slow->next;
        slow->next = nullptr;


        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);

        return merge(left, right);
    }

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        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:



9) **Merge k sorted lists:**

Code:

```
#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);
    }
};
```

Output: