AP ASSIGNMENT 3

Name – Rajeev Joshi

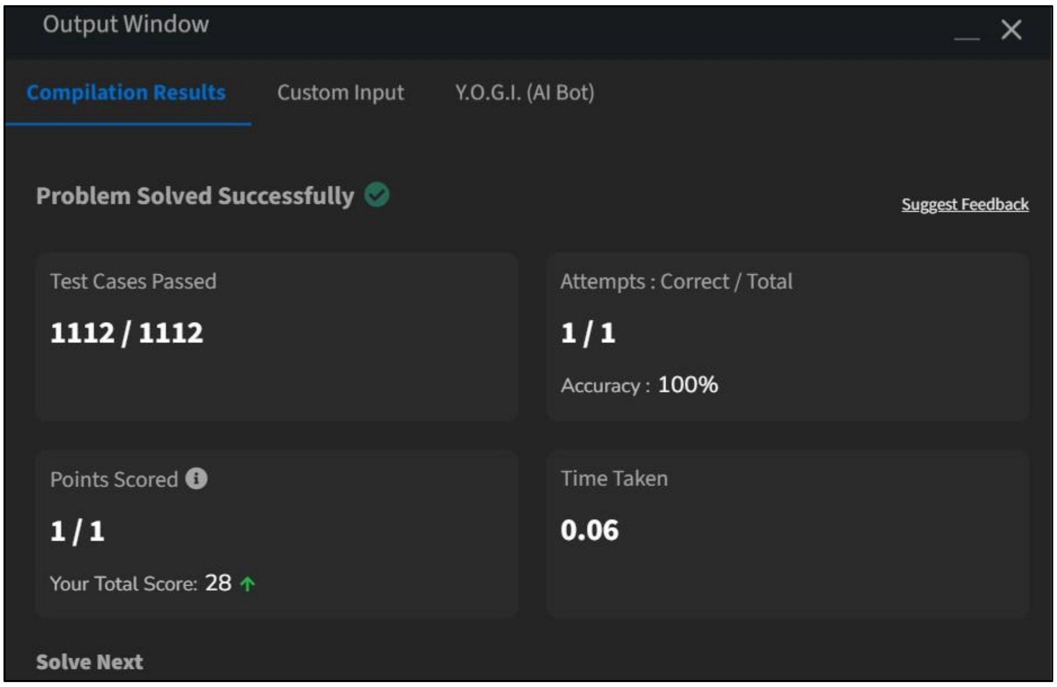
UID – 22BCS11920

Section – IOT_606-B

Date – 06/03/2025

1. Print Linked List:

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



Output Window

Compilation Results   Custom Input   Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓   Suggest Feedback

Test Cases Passed
1112 / 1112

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
1 / 1
Your Total Score: 28

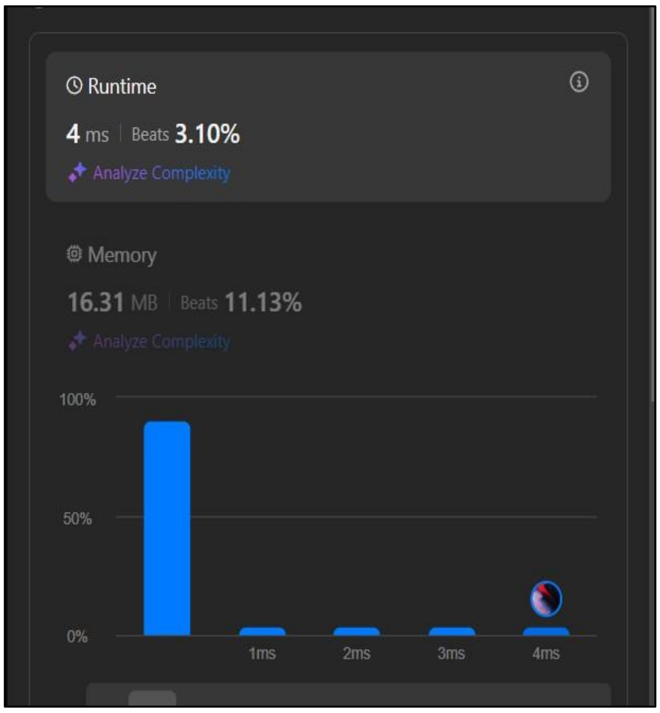
Time Taken
0.06

Solve Next

2. Remove duplicates from a sorted

```
list: class Solution {
public:
   ListNode* deleteDuplicates(ListNode*
      head) { if(!head || !head->next){
         return head;
      }
      ListNode* current = head;
```

```
        while(current && current->next){
          if(current->val    ==    current->next-
            >val){                          ListNode*
            duplicate=current->next;
            current->next=current->next-
            >next; delete duplicate;
          }
          else{
            current=current->next;
          }
        }
        return head;
      }
    };
```



Runtime

4 ms | Beats 3.10%

Analyze Complexity

Memory

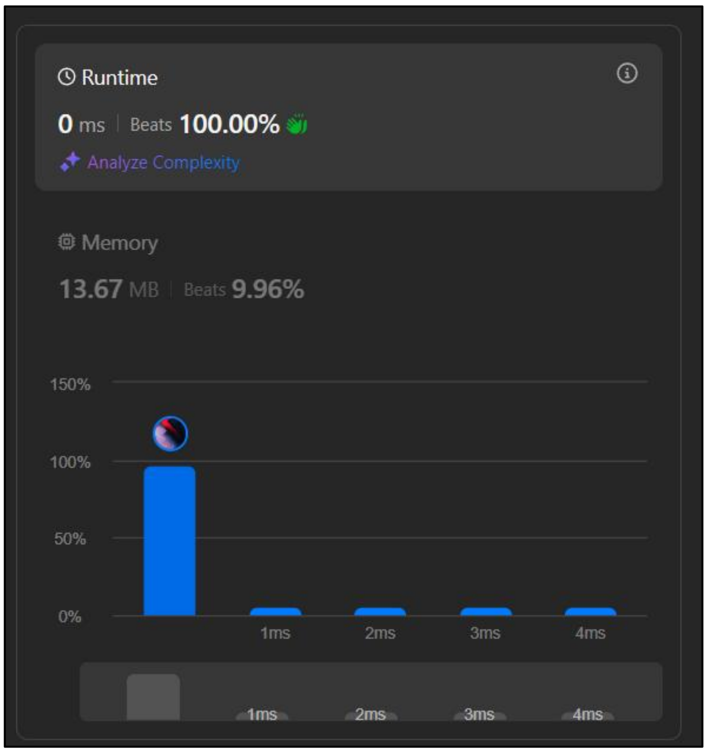
16.31 MB | Beats 11.13%

Analyze Complexity

3. Reverse a linked

list: class Solution

```
{
public:
  ListNode* reverseList(ListNode* head)
    { if(!head || !head->next){
      return head;
    }
    ListNode* newn=reverseList(head-
>next);
    head->next-
    >next=head; head-
    >next=nullptr; return
    newn;
  }
};
```



Runtime

0 ms | Beats 100.00%

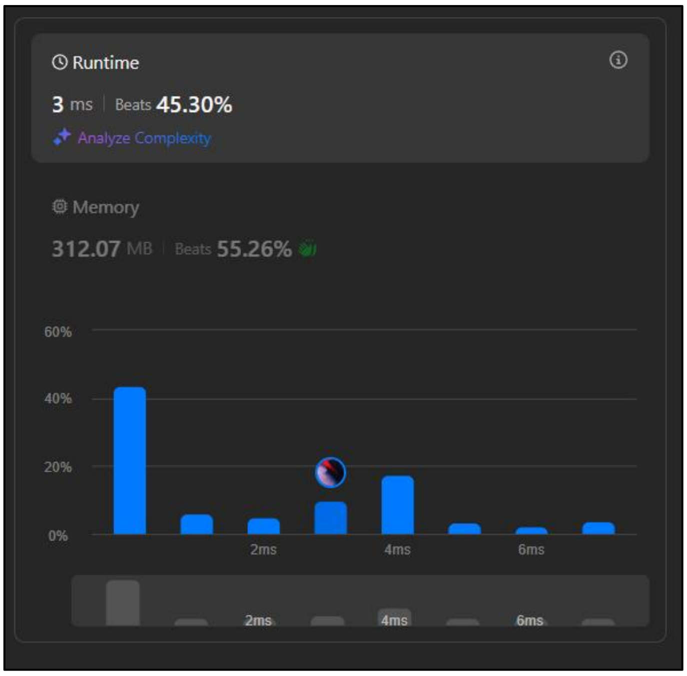
Analyze Complexity

Memory

13.67 MB | Beats 9.96%

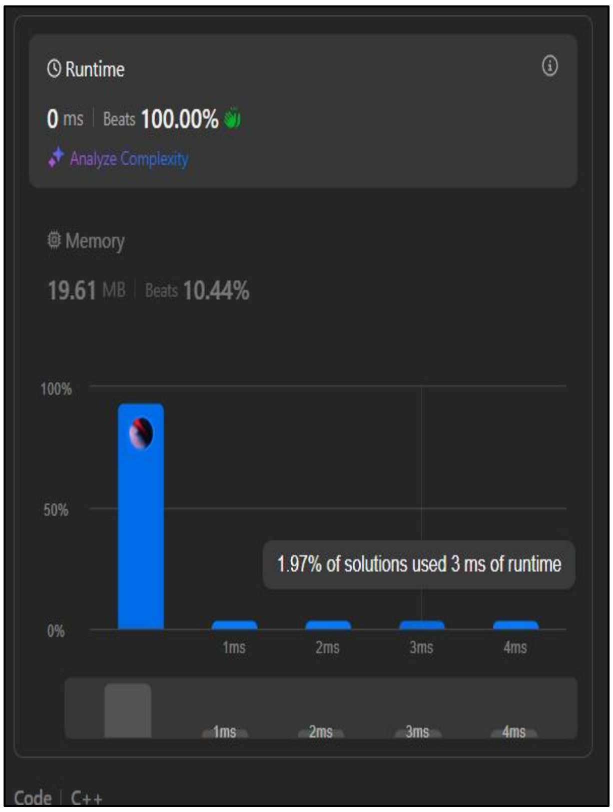
4. Delete middle node of a list:

```
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;
   }
};
```


Runtime
3 ms | Beats 45.30%
Analyze Complexity
Memory
312.07 MB | Beats 55.26%

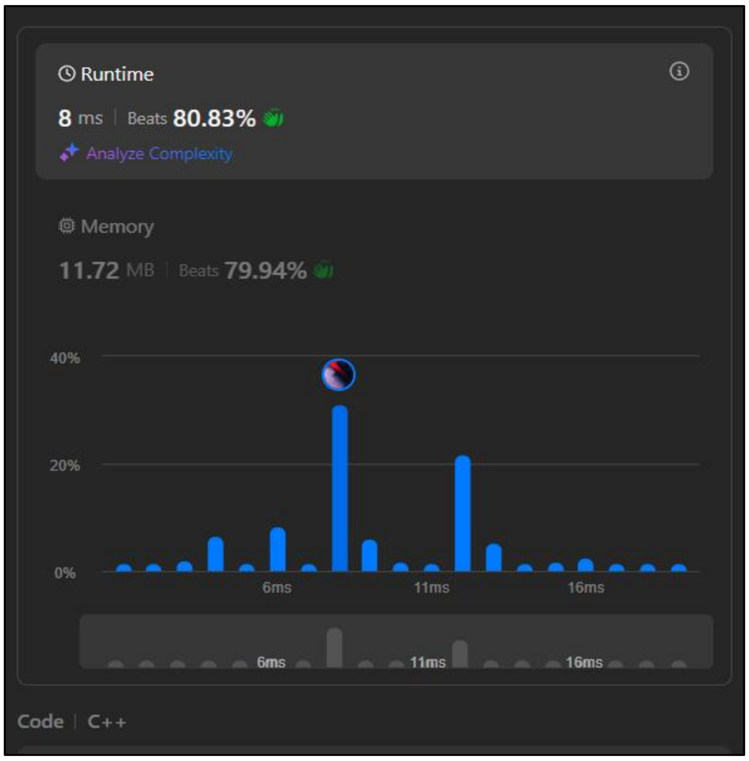
5. Merge two sorted linked

```
lists: class Solution {
public:
   ListNode* mergeTwoLists(ListNode* list1,
ListNode* list2) {
      if (!list1) return list2;
      if (!list2) return list1;

      if (list1->val <= list2->val) {
         list1->next = mergeTwoLists(list1->next,
         list2); return list1;
      } else {
         list2->next = mergeTwoLists(list1, list2-
         >next); return list2;
      }
   }

};
```


Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
19.61 MB | Beats 10.44%
1.97% of solutions used 3 ms of runtime
Code | C++

6. Detect a cycle in a linked list:

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


Runtime

8 ms | Beats 80.83%

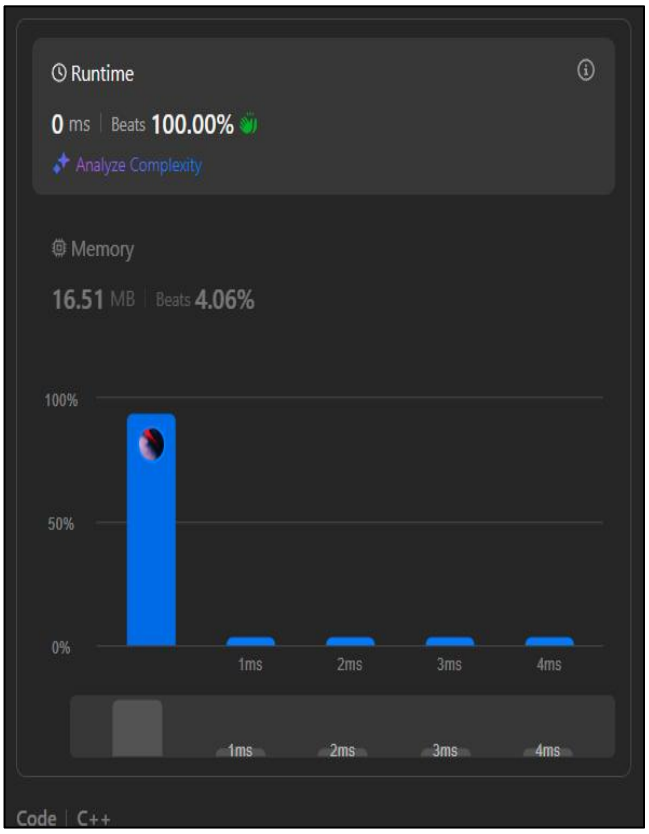
Analyze Complexity

Memory

11.72 MB | Beats 79.94%

7. Rotate a list:

```
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;
    }

    ListNode* newTail = head;
    for (int i = 0; i < length - k - 1;
      i++) { newTail = newTail-
      >next;
```


Runtime

0 ms | Beats 100.00%

Analyze Complexity

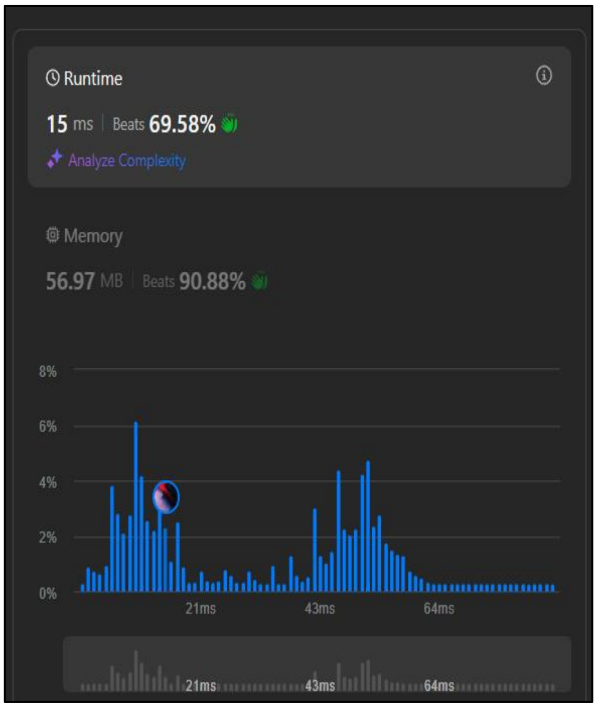
Memory

16.51 MB | Beats 4.06%

```
        }

        ListNode* newHead = newTail-
        >next; newTail->next = nullptr;
        tail->next = head;

        return newHead;
    }
};
```

8. Sort List:

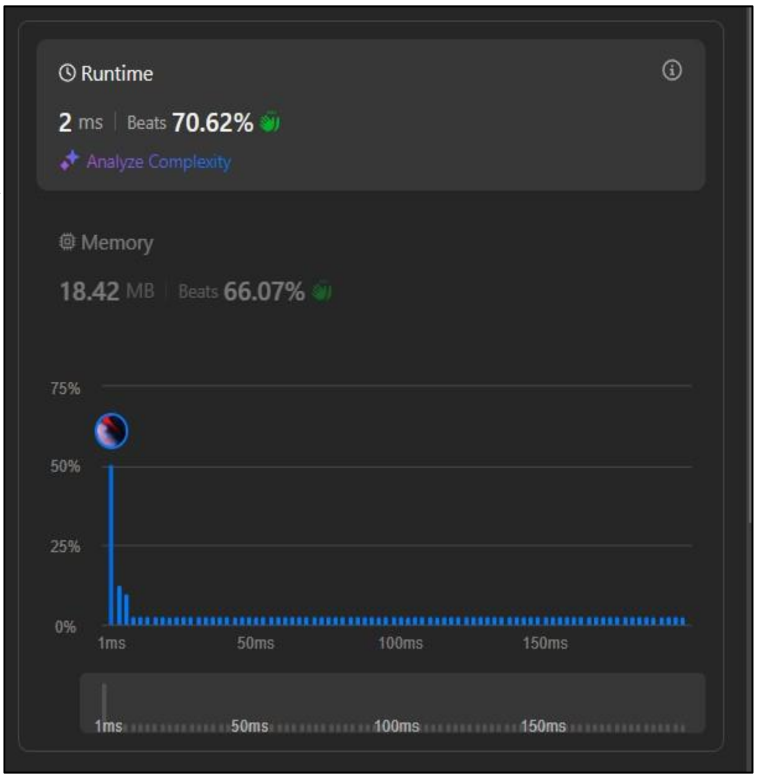
```
class Solution
{ public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* mid = getMiddle(head);
        ListNode* left = head;
        ListNode* right = mid-
        >next; mid->next =
        nullptr;
        left = sortList(left);
        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* 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;
            }
```


Runtime
15 ms | Beats 69.58%
Analyze Complexity
Memory
56.97 MB | Beats 90.88%

```
            tail = tail->next;
        }
        tail->next = l1 ? l1 : l2;
        return dummy.next;
    }
};
```

9. Merge K Sorted

```
List: 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> pq;

        for (auto list : lists) {
            if (list) pq.push(list);
        }

        ListNode dummy(0);
        ListNode* tail = &dummy;

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

            if (node->next) pq.push(node->next);
        }

        return dummy.next;
    }
};
```



Runtime

2 ms | Beats 70.62%

Analyze Complexity

Memory

18.42 MB | Beats 66.07%