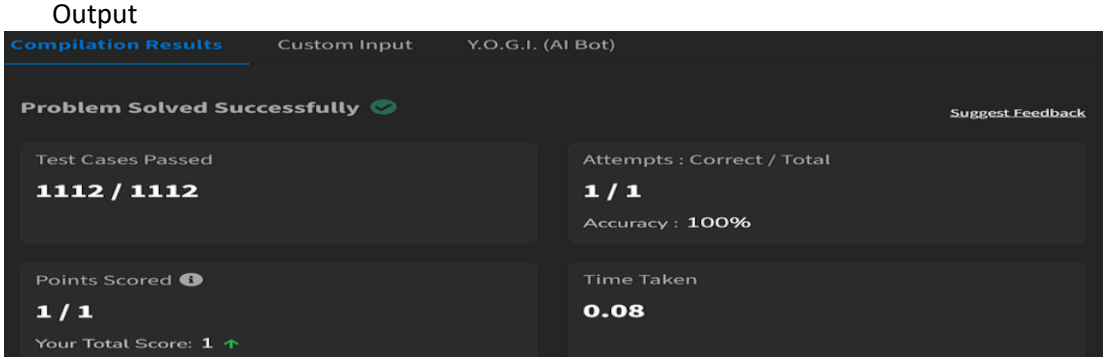
Assignment-3

1. Print Linked List:

- Code:-

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

Output

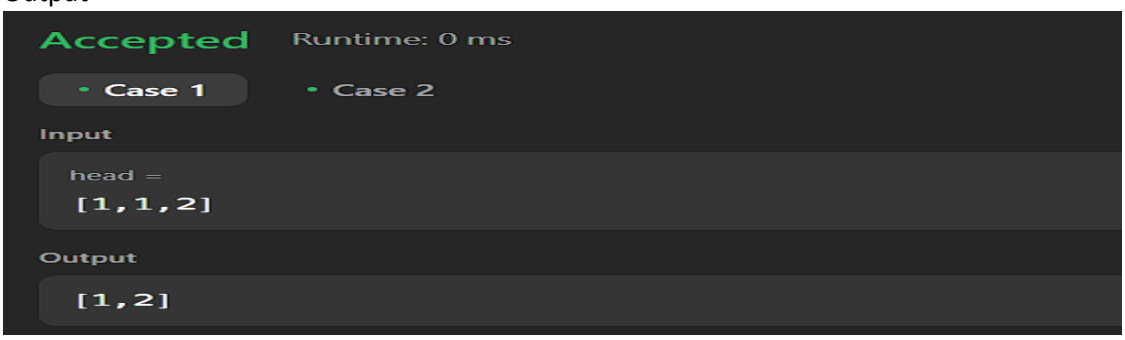


2. Remove duplicates from a sorted list:

- 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

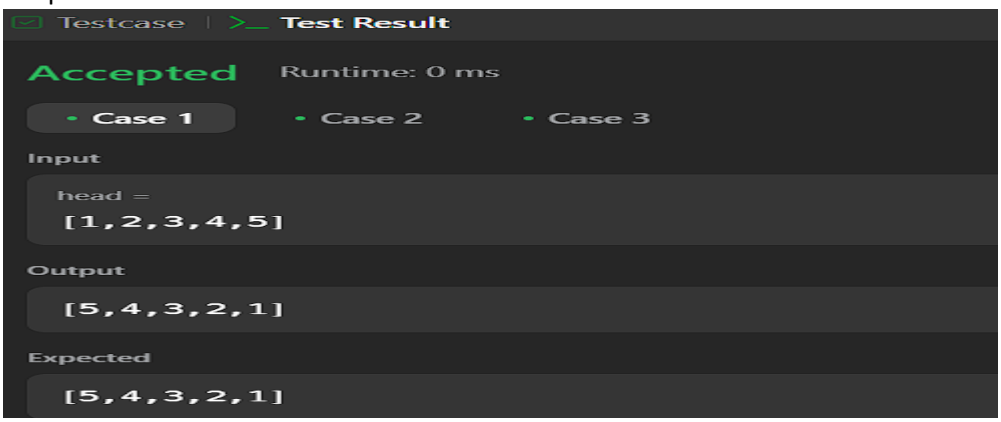


3. Reverse a linked list:

- Code:-

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* node = nullptr;
        while (head != nullptr) {
            ListNode* temp = head->next;
            head->next = node;
            node = head;
            head = temp;
        }
        return node;
    }
};
```

Output

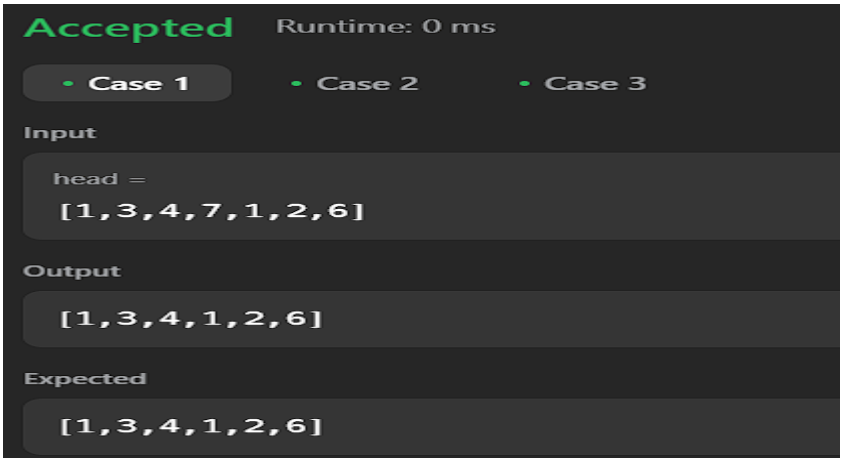


4. Delete middle node of a list:

- Code:-

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

Output:



5. Merge two sorted linked lists:

- Code:-

```
class Solution {
public:
   ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
      ListNode* dummy = new ListNode(0);
      ListNode* cur = dummy;
      while (list1 && list2) {
         if (list1->val > list2->val) {
            cur->next = list2;
            list2 = list2->next;
         } else {
            cur->next = list1;
            list1 = list1->next;
         }
         cur = cur->next;
      }
      cur->next = list1 ? list1 : list2;
      ListNode* head = dummy->next;
      delete dummy;
      return head;
};
```

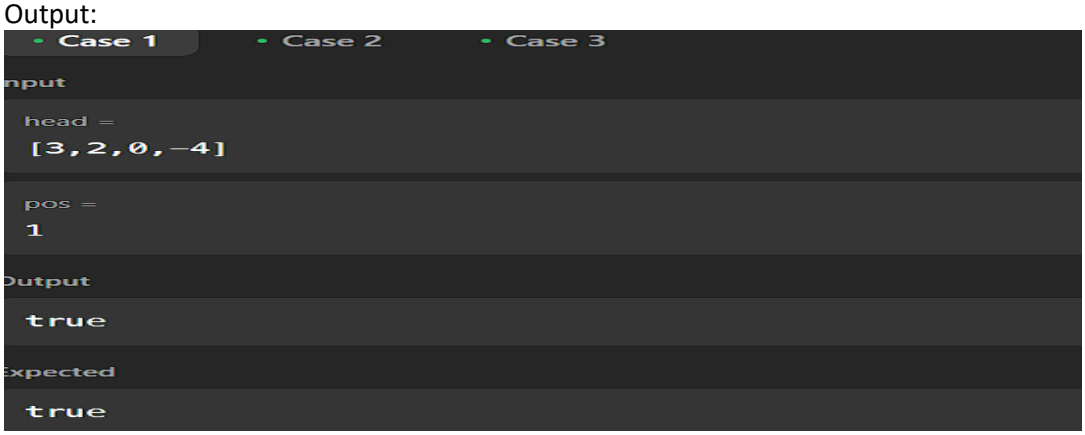
Output:

6. Detect a cycle in a linked list:

- Code:-

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

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:-

```
class Solution {
  public:
  ListNode* merge(ListNode* list1, ListNode* list2) {
      ListNode* C = new ListNode(100) ;
      ListNode* temp = C ;
      while(list1!= NULL && list2!=NULL){
        if(list1->val <= list2->val){
          temp->next = list1 ;
          list1 = list1->next ;
          temp = temp ->next ;
        }
        else{
          temp->next = list2 ;
          list2 = list2->next ;
          temp = temp->next ;
        }
      }
      if(list1 == NULL )temp->next = list2 ;
      if(list2==NULL) temp->next = list1 ;
    return C->next ;
    }
    ListNode* sortList(ListNode* head) {
      if(head==NULL || head->next == NULL)return head ;
      ListNode* slow = head ;
      ListNode* fast = head ;
      while(fast->next!=NULL && fast->next->next!=NULL){
        slow = slow->next ;
        fast = fast->next->next ;

      }
      ListNode* a = head ;
      ListNode* b = slow->next ;
      slow->next  = NULL ;
```

```
        a = sortList(a) ;
        b = sortList(b) ;
        ListNode* c = merge(a,b) ;
        return c ;
    }
};
```

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:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
lists =
[[1,4,5],[1,3,4],[2,6]]
Output
[1,1,2,3,4,4,5,6]
Expected
[1,1,2,3,4,4,5,6]

};