Experiment 3

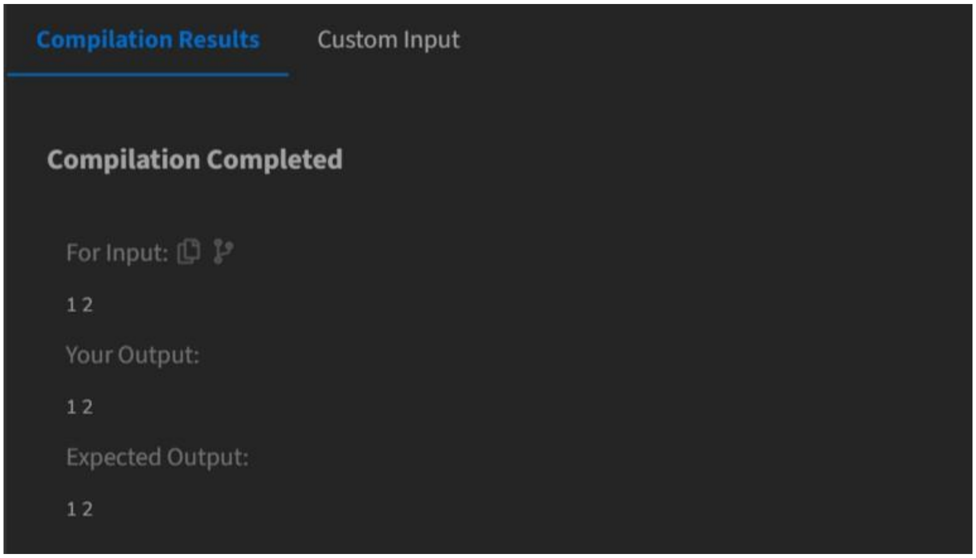
Name: Rohan **UID: 22BET10169**
Branch: BE-IT **Section/Group: 22BET_701-B**
Semester: 6 **Date of Performance: 05-02-25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22ITP-351**

Problem 1. Given a linked list. Print all the elements of the linked list separated by space followed.

Code:

```
class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        Node * ptr = head;

        while(ptr != NULL){
            cout << ptr -> data << " ";
            ptr = ptr -> next;
        }
    }
};
```

Output:



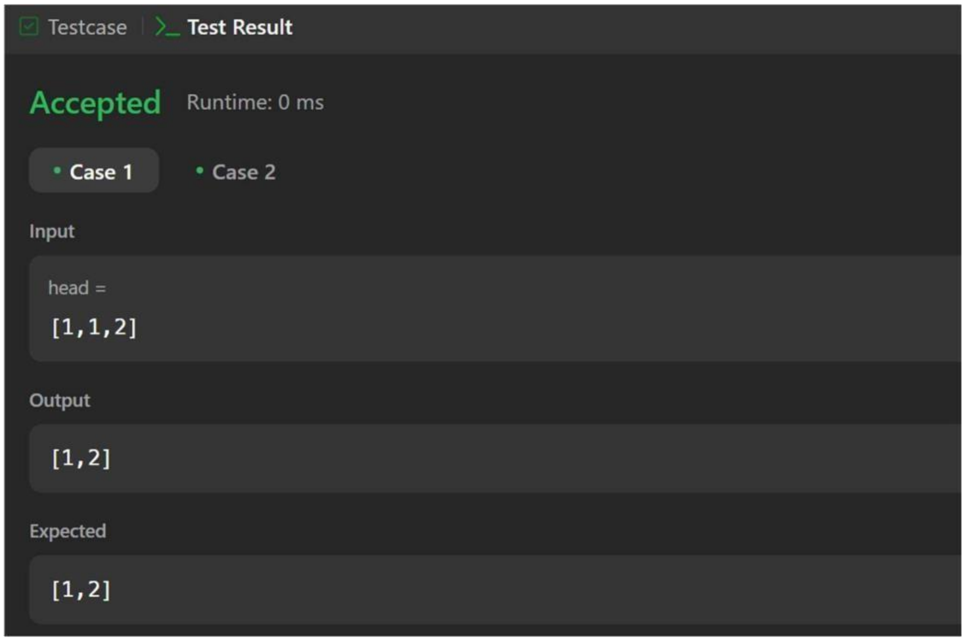
```
Compilation Completed

For Input:
1 2
Your Output:
1 2
Expected Output:
1 2
```

Problem 2. Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list sorted as well*.

Code:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;
        while(current && current-> next){
            if(current->val == current->next->val){
                ListNode* temp = current->next;
                current->next = current->next->next;
                delete temp;
            }
            else
            current = current->next;
        }
        return head;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

```
head =
[1,1,2]
```

Output

```
[1,2]
```

Expected

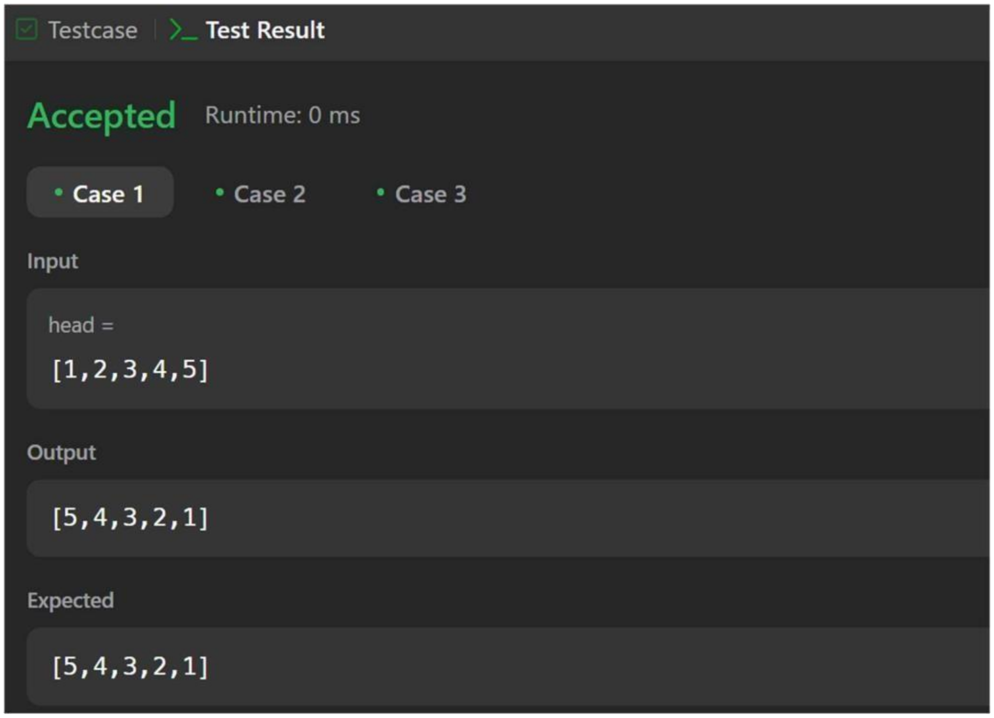
```
[1,2]
```

Problem 3. Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Code:

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

Output:



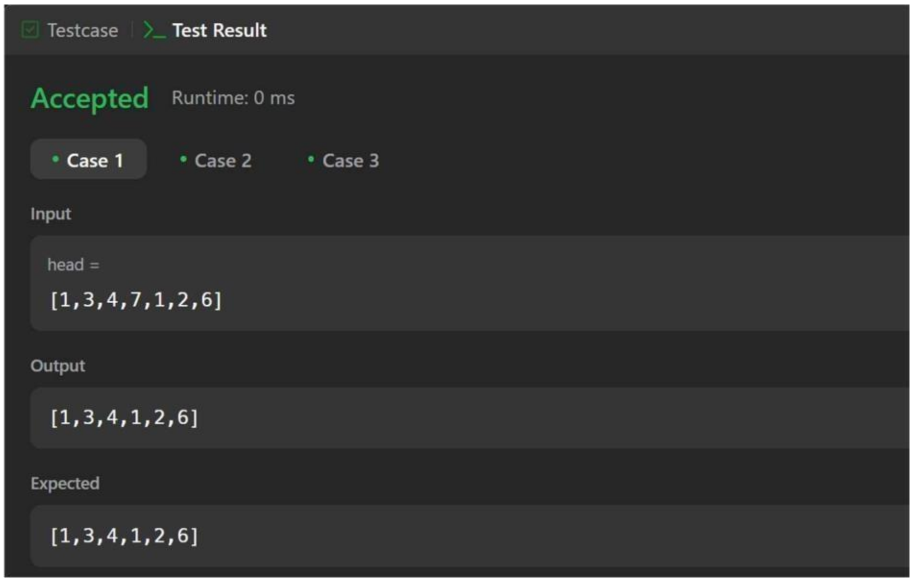
```
Accepted   Runtime: 0 ms

Input
head =
[1,2,3,4,5]

Output
[5,4,3,2,1]

Expected
[5,4,3,2,1]
```

Problem 4. You are given the head of a linked list. Delete the middle node, and return *the* head *of the modified linked list*.

Code:

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

Output:



Accepted Runtime: 0 ms

Input

head =

```
[1,3,4,7,1,2,6]
```

Output

```
[1,3,4,1,2,6]
```

Expected

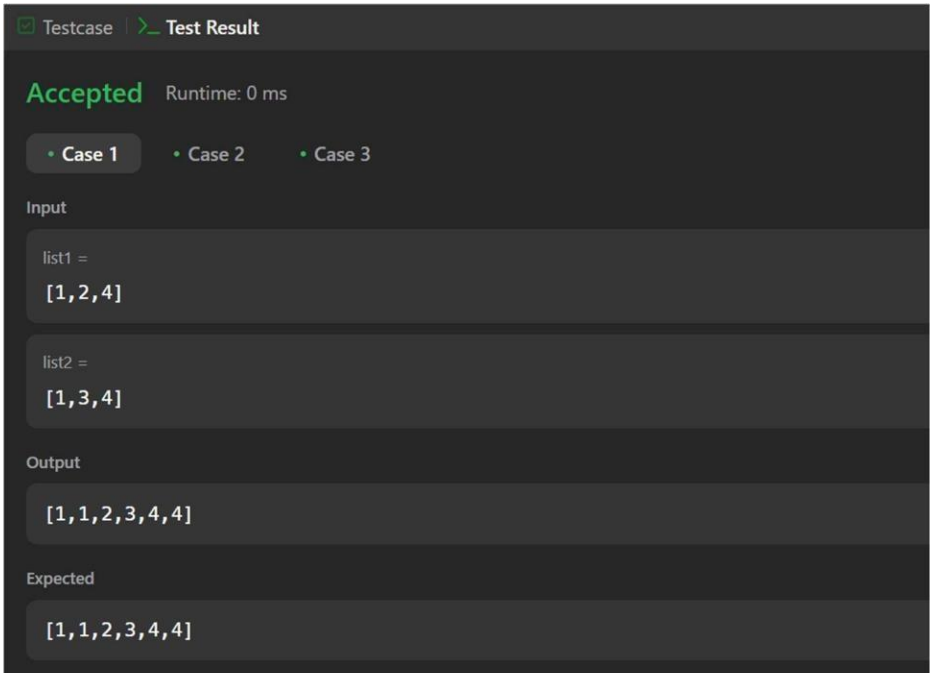
```
[1,3,4,1,2,6]
```

Problem 5. You are given the heads of two sorted linked lists list1 and list2.

Code:

```
class Solution {
 public:
   ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
       ListNode dummy(0);
       ListNode* tail = &dummy;

       while (list1 != nullptr && list2 != nullptr) {
           if (list1->val <= list2->val) {
               tail->next = list1; list1
               = list1->next;
           } else { tail->next =
               list2; list2 = list2-
               >next;
           }
           tail = tail->next;
       }
tail->next = (list1 != nullptr) ? list1 : list2;
       return dummy.next;
   }
};
```

Output:



Accepted Runtime: 0 ms

Input

list1 =

```
[1,2,4]
```

list2 =

```
[1,3,4]
```

Output

```
[1,1,2,3,4,4]
```

Expected

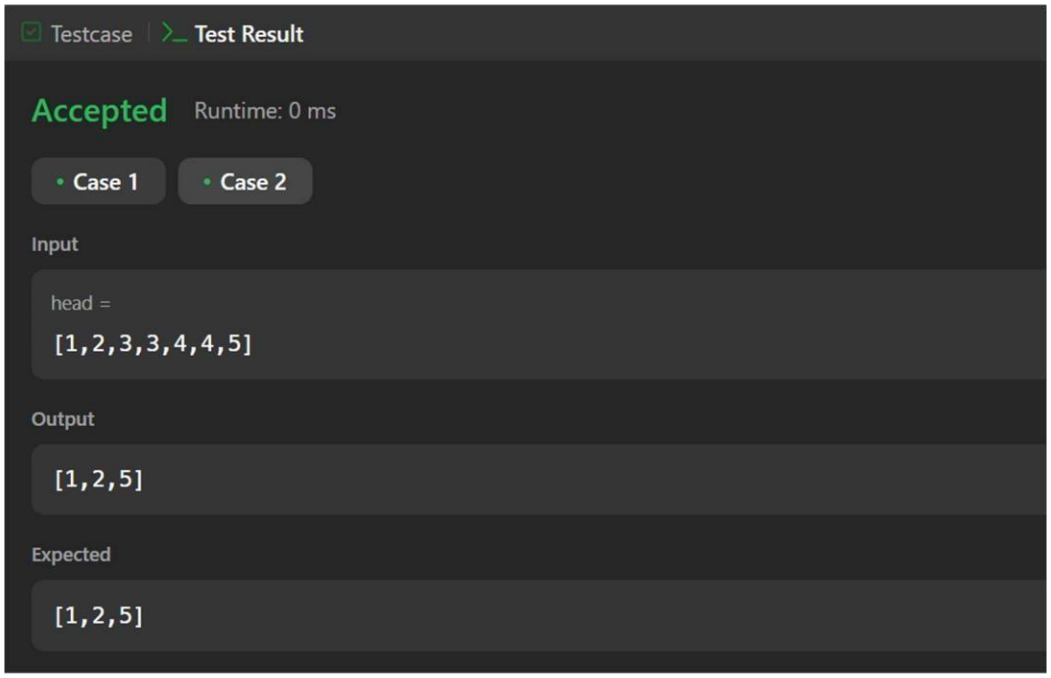
```
[1,1,2,3,4,4]
```

Problem 6. Given the head of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return *the linked list sorted as well*.

Code:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
       ListNode *dummy=new ListNode(0,head);
       ListNode *prev=dummy;
       while(head!=NULL){
        if(head->next!=NULL && head->val==head->next->val){
while(head->next!=NULL && head->val==head->next->val)head=head->next;
                    prev->next=head->next;
                }
            else prev=prev->next;
            head=head->next;
        }
    return dummy->next;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

```
head =
[1,2,3,3,4,4,5]
```

Output

```
[1,2,5]
```

Expected

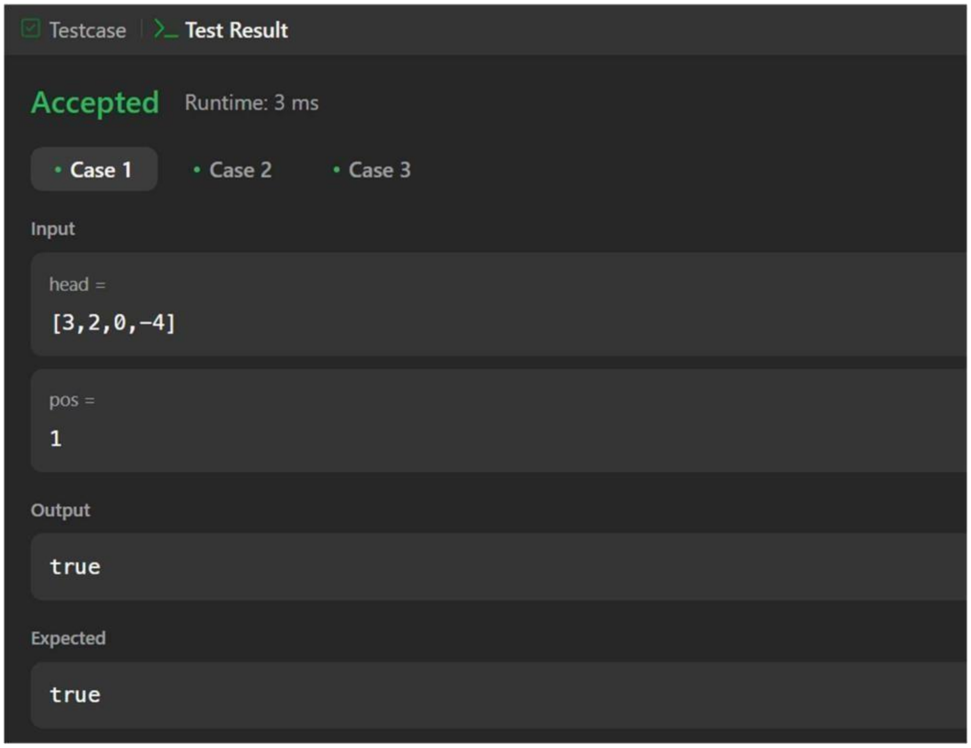
```
[1,2,5]
```

Problem 7. Given head, the head of a linked list, determine if the linked list has a cycle in it.

Code:

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

Output:



Accepted Runtime: 3 ms

Input

head =

```
[3,2,0,-4]
```

pos =

```
1
```

Output

```
true
```

Expected

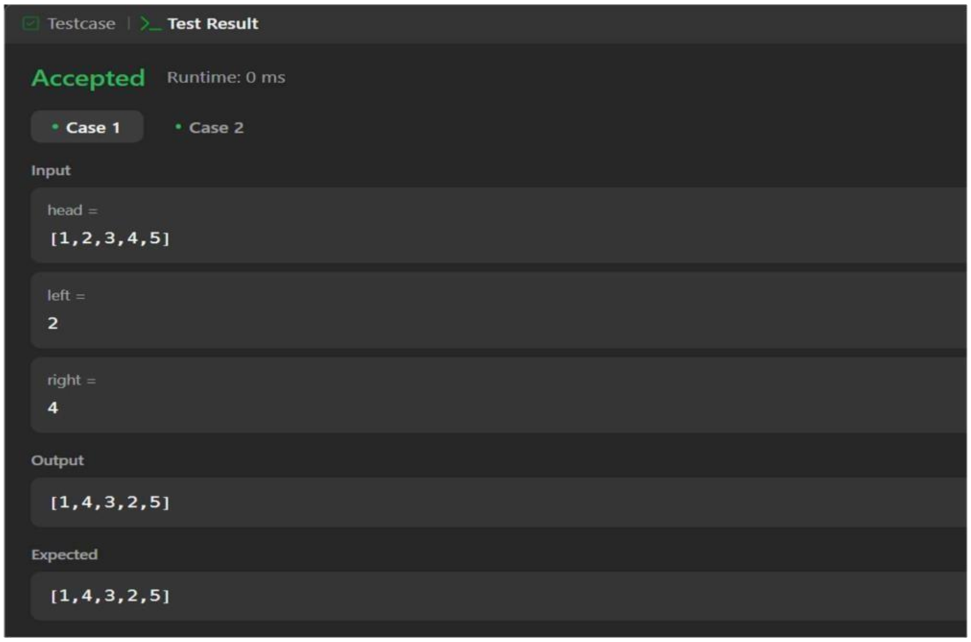
```
true
```

Problem 8. Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.

Code:

```
class Solution { public:
   ListNode* reverseBetween(ListNode* head, int left, int right) {
     if (head == nullptr || left == right) { return head;
      }
      ListNode* dummy = new ListNode(0);
      dummy->next = head; ListNode* prev
      = dummy;
      for (int i = 1; i < left; ++i) { prev
         = prev->next;
      }
      ListNode* curr = prev->next;
      ListNode* next = nullptr;
      for (int i = 0; i < right - left; ++i) {
         next = curr->next; curr->next =
         next->next; next->next = prev-
         >next; prev->next = next;
      }
      return dummy->next;
   }
};
```

Output:



Accepted Runtime: 0 ms

Input

head =
[1,2,3,4,5]

left =
2

right =
4

Output

[1,4,3,2,5]

Expected

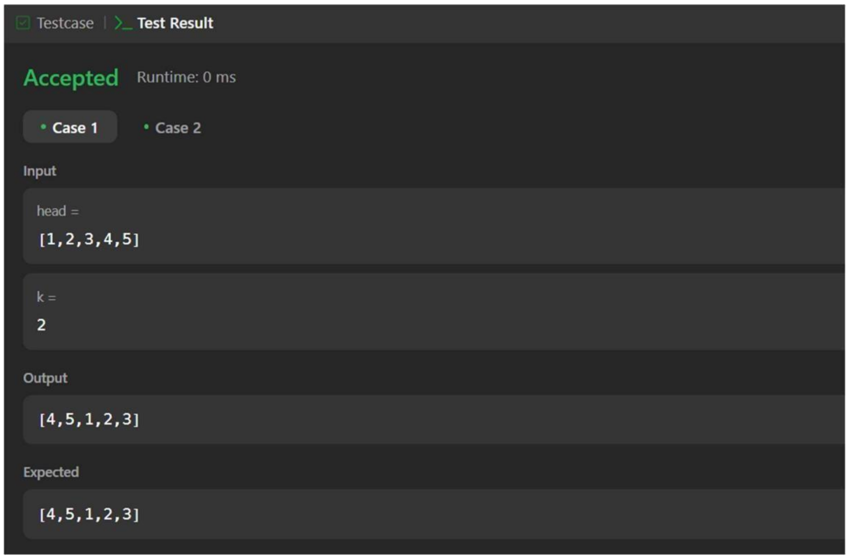
[1,4,3,2,5]

Problem 9. Given the head of a linked list, rotate the list to the right by k places.

Code:

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

            return newHead;
        }
    };
```

Output:



Accepted Runtime: 0 ms

Input

head =
[1,2,3,4,5]

k =
2

Output

[4,5,1,2,3]

Expected

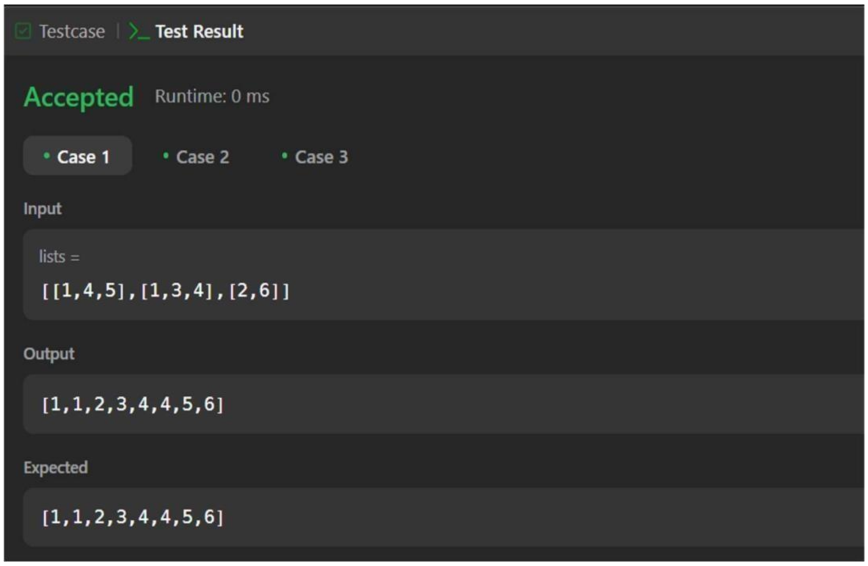
[4,5,1,2,3]

Problem 10. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Code:

```
         class Solution { public:
            ListNode* mergeKLists(vector<ListNode*>& lists) { if
               (lists.empty()) return nullptr;
               auto compare = [](ListNode* a, ListNode* b) { return
                  a->val > b->val;
               };
 priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> minHeap(compare);
               for (ListNode* list : lists) { if
                  (list) {
                     minHeap.push(list);
                  }
               }
               ListNode* dummy = new ListNode(0);
               ListNode* current = dummy;
               while       (!minHeap.empty())       {
                  ListNode* node = minHeap.top();
                  minHeap.pop();  current->next  =
                  node;  current  =  current->next;  if
                  (node->next)                       {
                  minHeap.push(node->next);
                  }
               }
               return dummy->next;
            }
         };
```

Output:



Accepted Runtime: 0 ms

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]
```