Assignment:3

Name:Amul

UID:22BCS13712

Subject:AP

Section:IOT-607/B

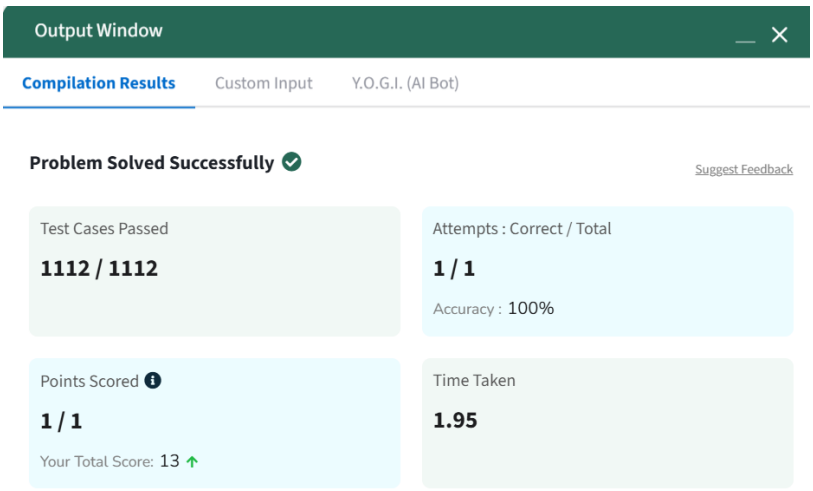
1. **Print Linked List:**

Code:

```
class Solution {

   void printList(Node head) {
      Node listElement = head;
  while (listElement != null) {
   System.out.print(listElement.data + " ");
   listElement = listElement.next;
     }
   }
}
```

Output:



2. **Remove duplicates from a sorted list:**

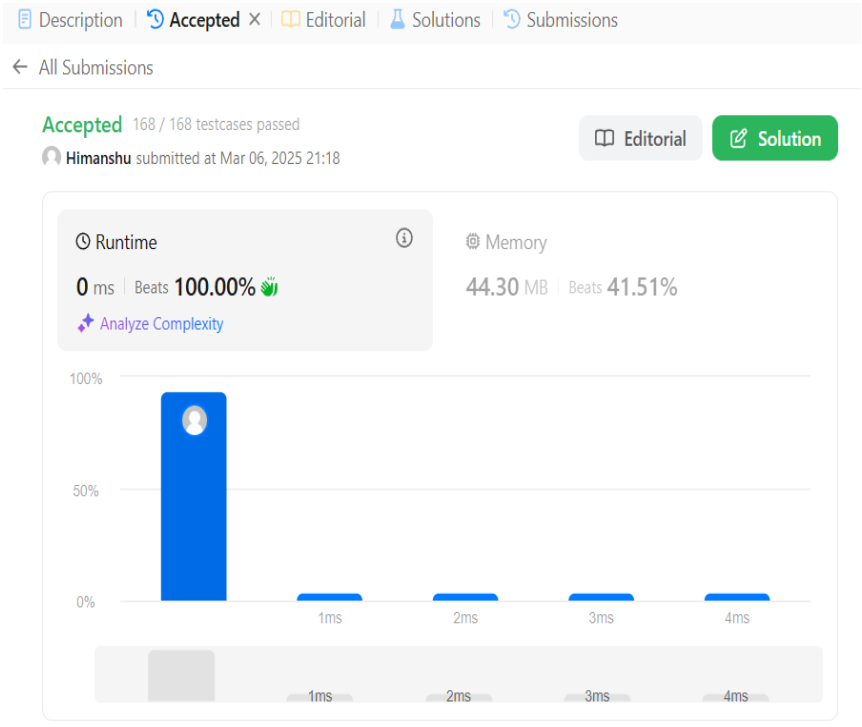
Code:

```
class Solution {
  public ListNode deleteDuplicates(ListNode head) {
    ListNode curr = head;

    while (curr != null) {
      while (curr.next != null && curr.val == curr.next.val)
        curr.next = curr.next.next;
      curr = curr.next;
    }

    return head;
  }
}
```

Output:



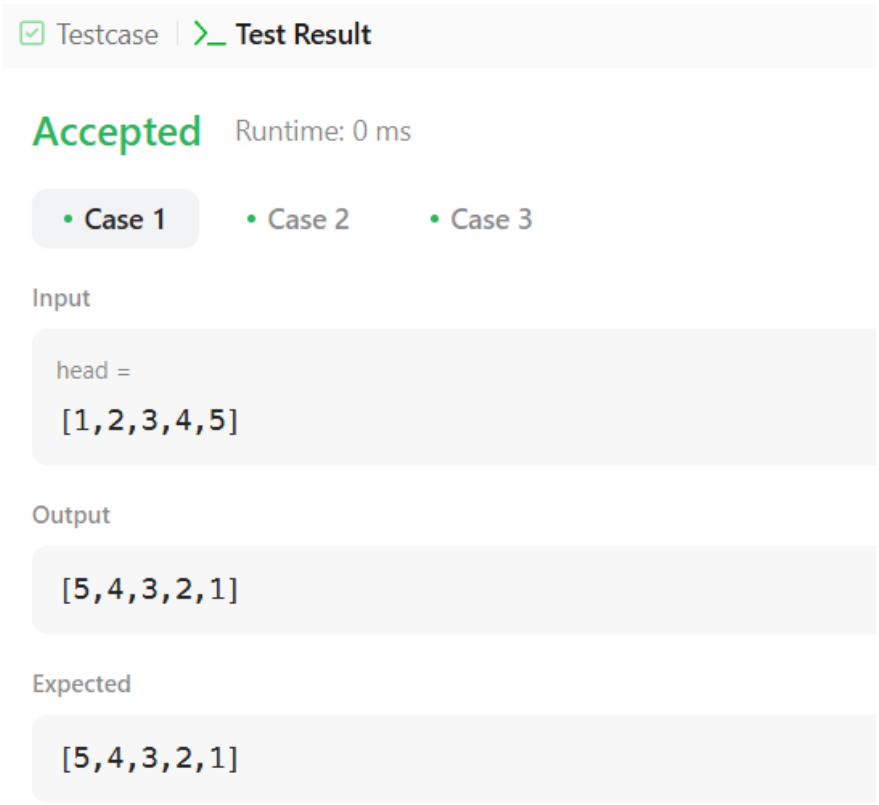
3. **Reverse a linked list:**

Code:

```
class Solution {
  public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null)
      return head;

    ListNode newHead = reverseList(head.next);
    head.next.next = head;
    head.next = null;
    return newHead;
  }
}
```

Output:



4. **Delete middle node of a list:**

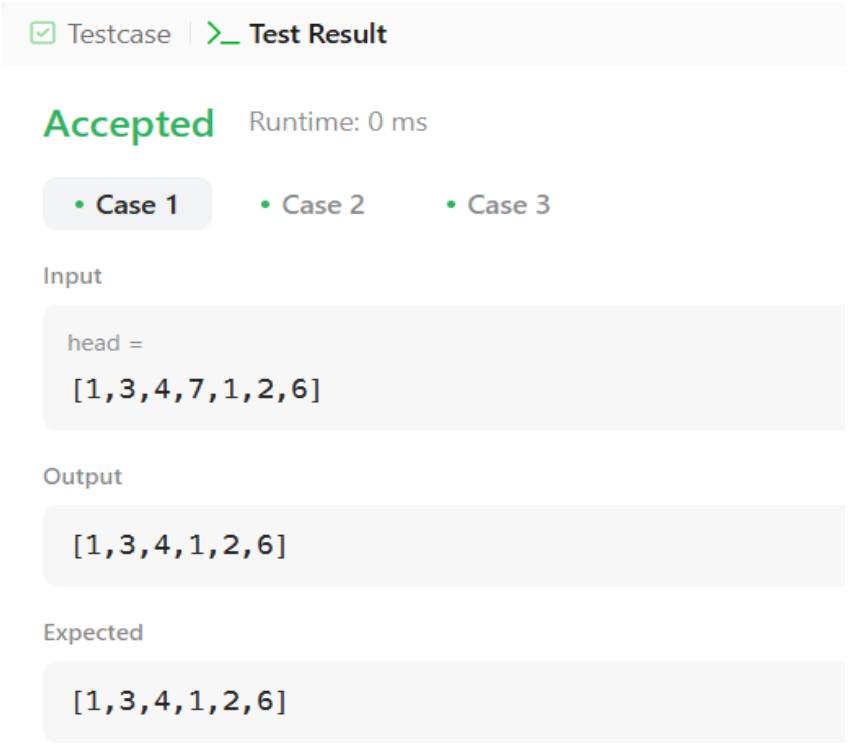
Code:

```
class Solution {
  public ListNode deleteMiddle(ListNode head) {
    ListNode dummy = new ListNode(0, head);
    ListNode slow = dummy;
    ListNode fast = dummy;

    while (fast.next != null && fast.next.next != null) {
      slow = slow.next;
      fast = fast.next.next;
    }

    // Delete the middle node.
    slow.next = slow.next.next;
    return dummy.next;
  }
}
```

Output:

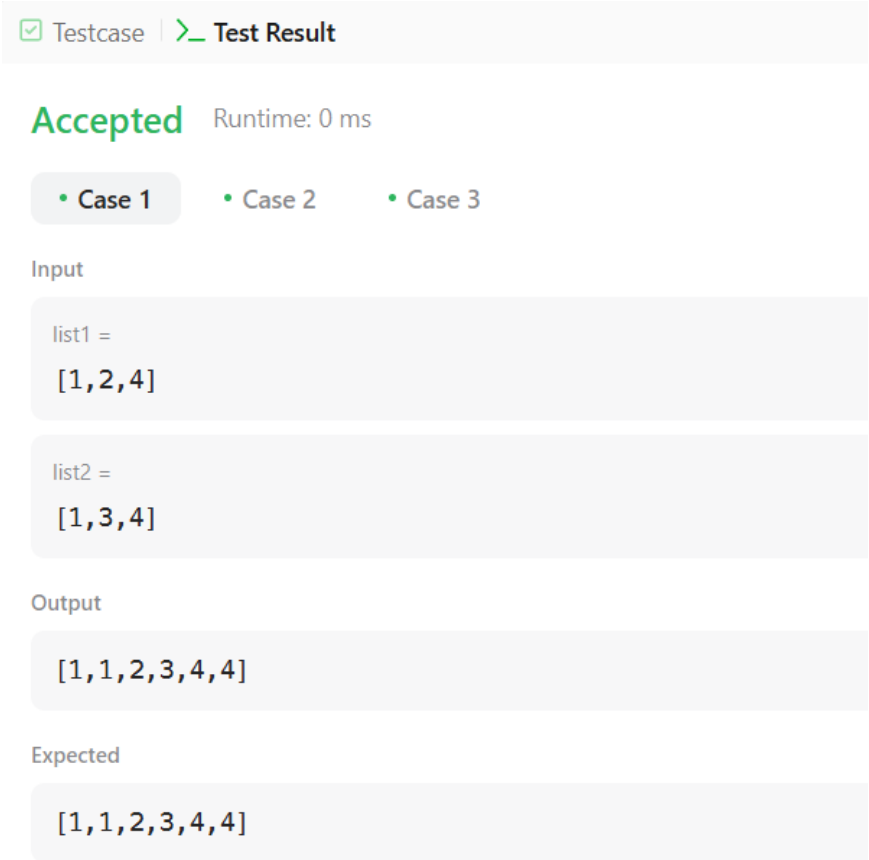


5. **Merge two sorted linked lists:**

Code:

```
class Solution {
  public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if (list1 == null || list2 == null)
      return list1 == null ? list2 : list1;
    if (list1.val > list2.val) {
      ListNode temp = list1;
      list1 = list2;
      list2 = temp;
    }
    list1.next = mergeTwoLists(list1.next, list2);
    return list1;
  }
}
```

Output:



6. **Detect a cycle in a linked list:**

Code:

```
class Solution {
  public boolean hasCycle(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;

    while (fast != null && fast.next != null) {
      slow = slow.next;
      fast = fast.next.next;
      if (slow == fast)
        return true;
    }

    return false;
  }
}
```

Output:

7. **Rotate a list:**

Code:

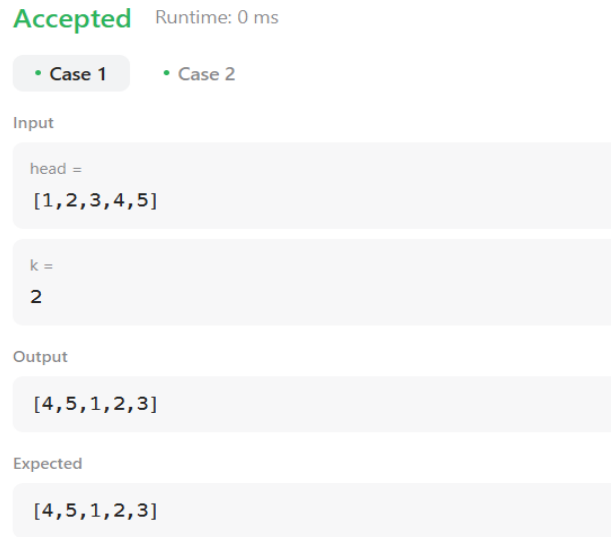
```
class Solution {
  public ListNode rotateRight(ListNode head, int k) {
    if (head == null || head.next == null || k == 0)
      return head;

    int length = 1;
    ListNode tail = head;
    for (; tail.next != null; tail = tail.next)
      ++length;
    tail.next = head;

    final int t = length - k % length;
    for (int i = 0; i < t; ++i)
      tail = tail.next;
    ListNode newHead = tail.next;
    tail.next = null;

    return newHead;
  }
}
```

Output:



8. **Sort List:**

Code:

```
class Solution {
 public ListNode sortList(ListNode head) {
  final int length = getLength(head);
  ListNode dummy = new ListNode(0, head);

  for (int k = 1; k < length; k *= 2) {
   ListNode curr = dummy.next;
   ListNode tail = dummy;
   while (curr != null) {
    ListNode l = curr;
    ListNode r = split(l, k);
    curr = split(r, k);
    ListNode[] merged = merge(l, r);
    tail.next = merged[0];
    tail = merged[1];
   }
  }

  return dummy.next;
 }

 private int getLength(ListNode head) {
  int length = 0;
  for (ListNode curr = head; curr != null; curr = curr.next)
   ++length;
  return length;
 }

 private ListNode split(ListNode head, int k) {
  while (--k > 0 && head != null)
   head = head.next;
  ListNode rest = head == null ? null : head.next;
  if (head != null)
```

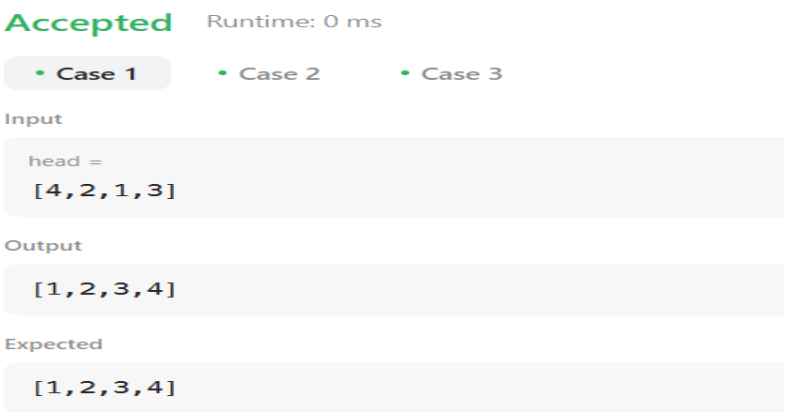
```
    head.next = null;
    return rest;
  }

  private ListNode[] merge(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode tail = dummy;

    while (l1 != null && l2 != null) {
      if (l1.val > l2.val) {
        ListNode temp = l1;
        l1 = l2;
        l2 = temp;
      }
      tail.next = l1;
      l1 = l1.next;
      tail = tail.next;
    }
    tail.next = l1 == null ? l2 : l1;
    while (tail.next != null)
      tail = tail.next;

    return new ListNode[] {dummy.next, tail};
  }
}
```

Output:



9. **Merge k sorted lists:**

Code:

```
class Solution {
  public ListNode mergeKLists(ListNode[] lists) {
    ListNode dummy = new ListNode(0);
    ListNode curr = dummy;
    Queue<ListNode> minHeap = new PriorityQueue<>((a, b) ->
Integer.compare(a.val, b.val));

    for (final ListNode list : lists)
      if (list != null)
        minHeap.offer(list);

    while (!minHeap.isEmpty()) {
      ListNode minNode = minHeap.poll();
      if (minNode.next != null)
        minHeap.offer(minNode.next);
      curr.next = minNode;
      curr = curr.next;
    }

    return dummy.next;
  }
}
```

Output: