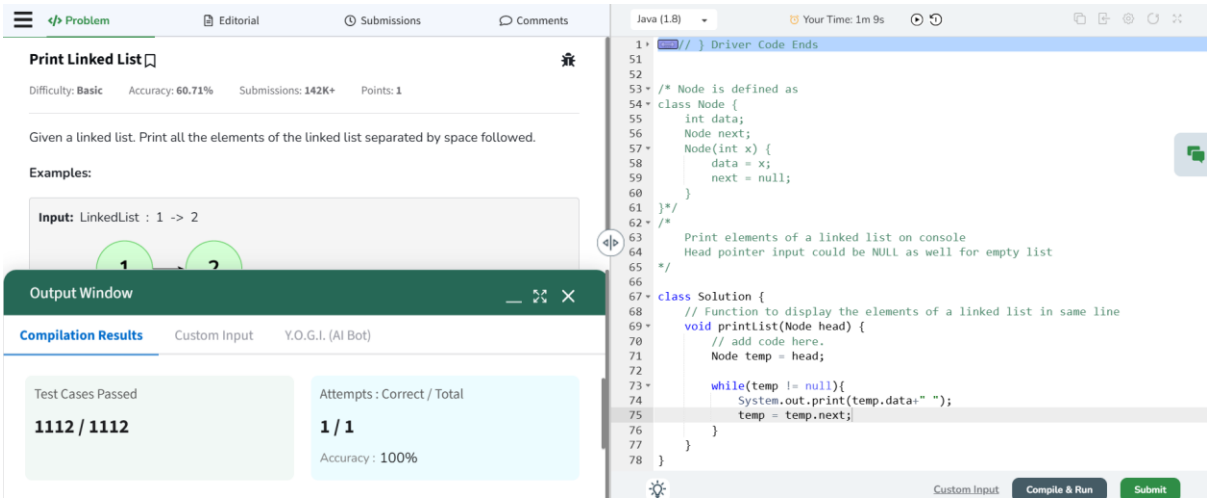
Assignment 3

1. Print Linked List

Code:

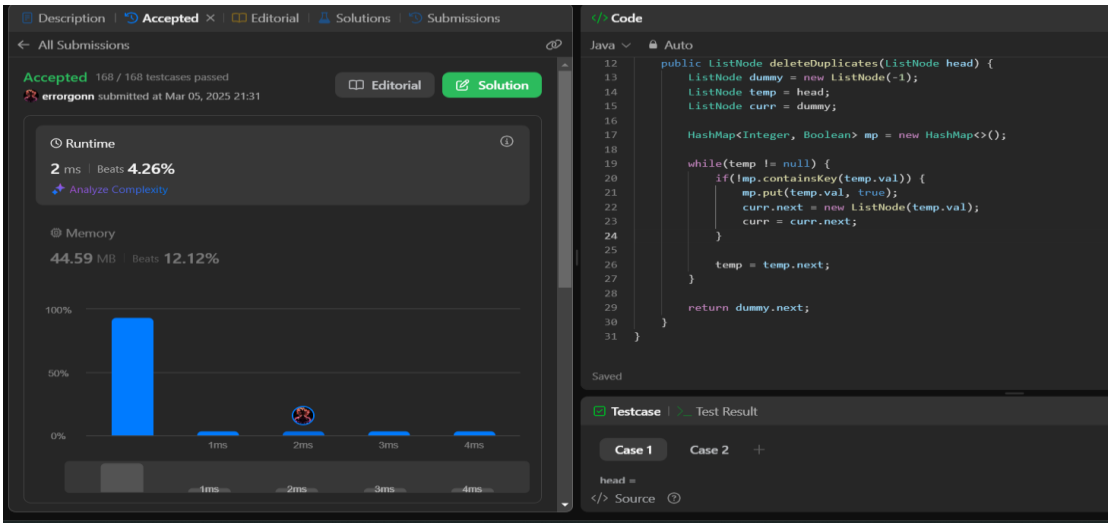
```
class Solution {
    // Function to display the elements of a linked list in same line
   void printList(Node head) {
      // add code here.
      Node temp = head;

      while (temp != null){
         System.out.print(temp.data+" ");
         temp = temp.next;
      }
   }
}
```



2. Remove duplicates from Sorted List

Code:

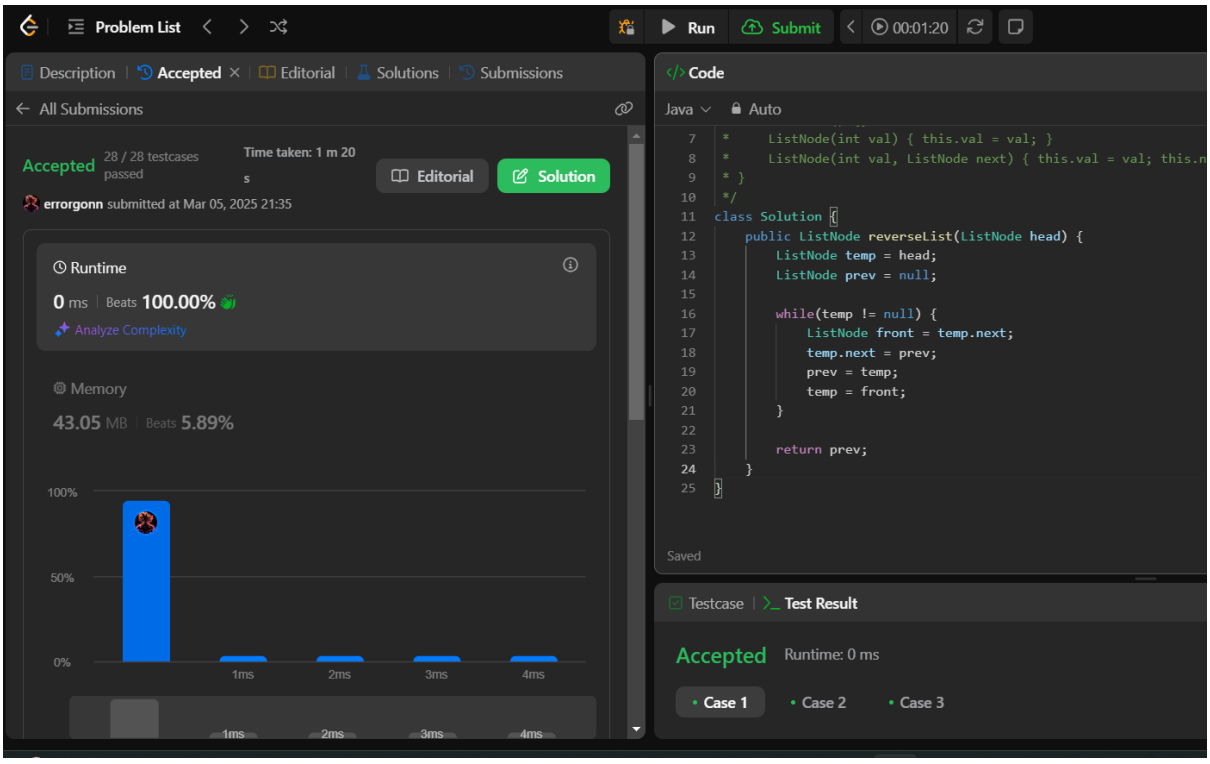
```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode dummy = new ListNode(-1);
        ListNode temp = head;
        ListNode curr = dummy;
        HashMap<Integer, Boolean> mp = new HashMap<>();
        while(temp != null) {
            if(!mp.containsKey(temp.val)) {
                mp.put(temp.val, true);
                curr.next = new ListNode(temp.val);
                curr = curr.next;
            }
            temp = temp.next;
        }
        return dummy.next;
    }
}
```



3.**Reverse a Linked List**

Code:

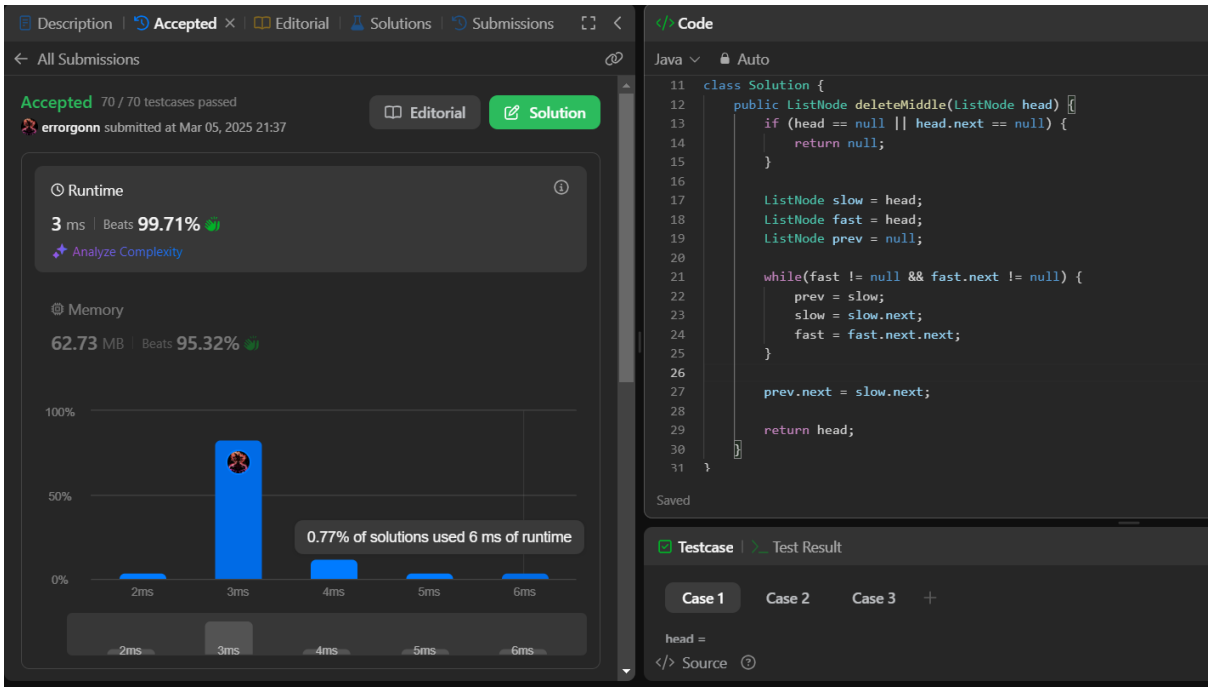
```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode temp = head;
        ListNode prev = null;
        while(temp != null) {
            ListNode front = temp.next;
            temp.next = prev;
            prev = temp;
            temp = front;
        }
        return prev;
    }
}
```



4. Delete middle of a Linked List

Code:

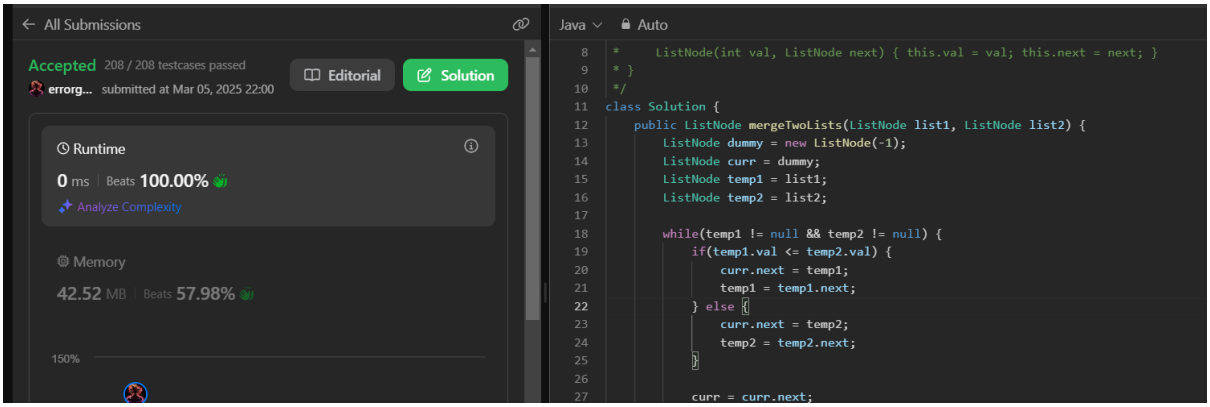
```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }
        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = null;

        while(fast != null && fast.next != null) {
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
        prev.next = slow.next;
        return head;
    }
}
```



5. Merge Two sorted Linked List

Code:

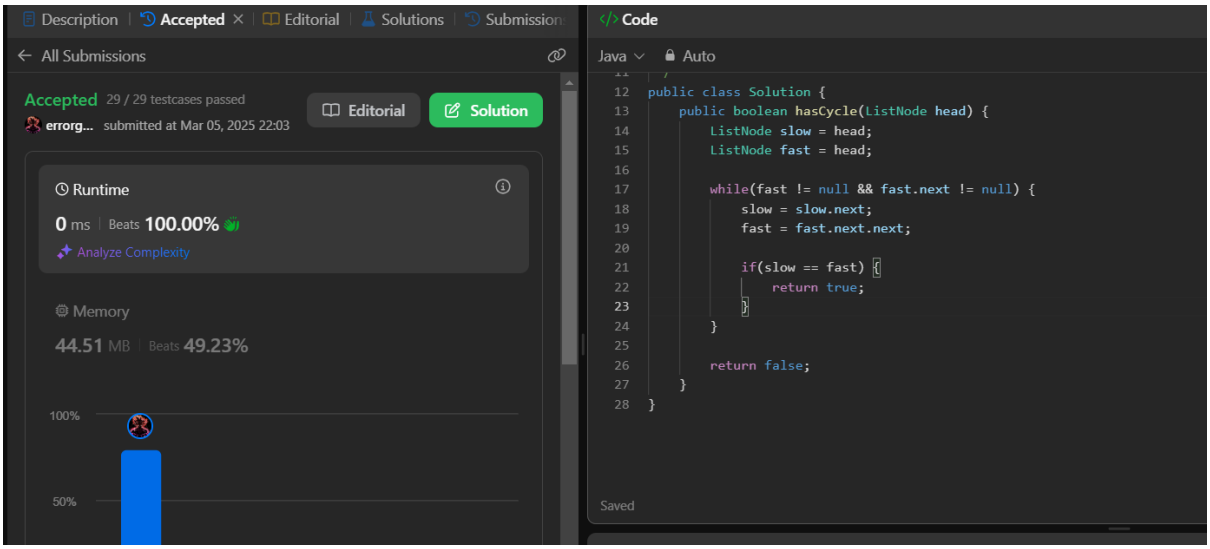
```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;
        ListNode temp1 = list1;
        ListNode temp2 = list2;

        while(temp1 != null && temp2 != null) {
            if(temp1.val <= temp2.val) {
                curr.next = temp1;
                temp1 = temp1.next;
            } else {
                curr.next = temp2;
                temp2 = temp2.next;
            }
            curr = curr.next;
        }
        if(temp1 != null) {
            curr.next = temp1;
        } else {
            curr.next = temp2;
        }
        return dummy.next;
    }
}
```



6. Detect a cycle in a Linked List

Code:

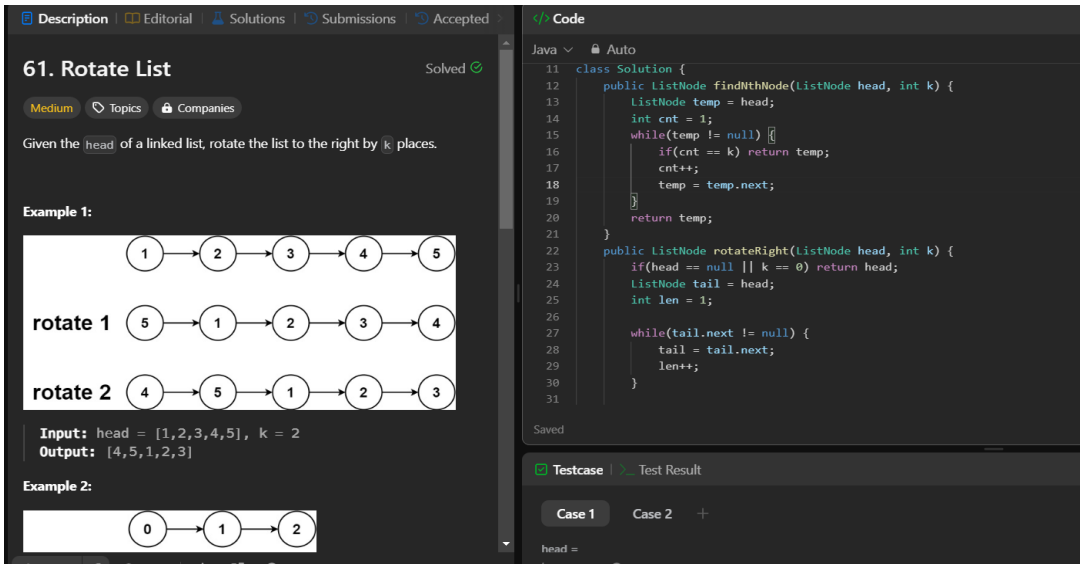
```
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if(slow == fast) {
                return true;
            }
        }
        return false;
    }
}
```



7. Rotate a Linked List

Code:

```
class Solution {
    public ListNode findNthNode(ListNode head, int k) {
        ListNode temp = head;
        int cnt = 1;
        while(temp != null) {
            if(cnt == k) return temp;
            cnt++;
            temp = temp.next;
        }
        return temp;
    }
    public ListNode rotateRight(ListNode head, int k) {
        if(head == null || k == 0) return head;
        ListNode tail = head;
        int len = 1;
        while(tail.next != null) {
            tail = tail.next;
            len++;
        }
        if(k % len == 0) return head;
        k = k % len;
        tail.next = head;
        ListNode newLastNode = findNthNode(head, len - k);
        head = newLastNode.next;
        newLastNode.next = null;
        return head;
    }
}
```

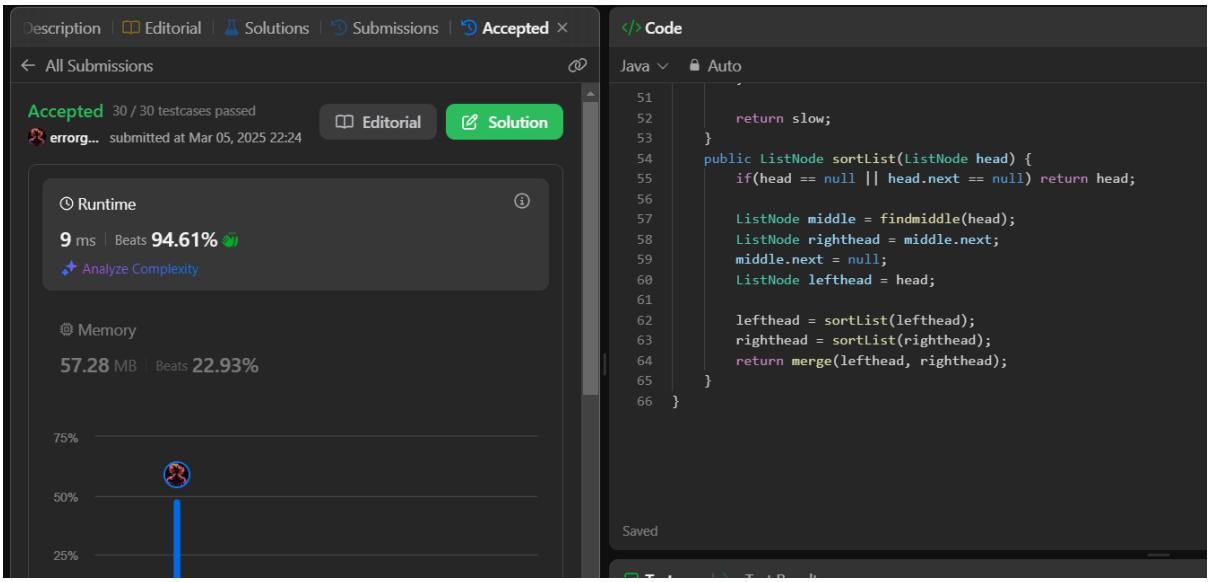


8. Sort List

Code:

```
class Solution {
   public ListNode merge(ListNode first, ListNode second) {
      ListNode t1 = first;
      ListNode t2 = second;
      ListNode dNode = new ListNode(-1);
      ListNode temp = dNode;
      while(t1 != null && t2 != null) {
         if(t1.val > t2.val) {
            temp.next = t2;
            temp = t2;
            t2 = t2.next;
         } else {
            temp.next = t1;
            temp = t1;
            t1 = t1.next;
         }
      }

      if(t1 != null) {
         temp.next = t1;
      } else {
         temp.next = t2;
      }

      return dNode.next;
          }

   public ListNode findmiddle(ListNode head) {
      ListNode slow = head;
      ListNode fast = head.next;

      if(head == null || head.next == null) return head;

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

      return slow;
   }
```

```
    public ListNode sortList(ListNode head) {
        if(head == null || head.next == null) return head;

        ListNode middle = findmiddle(head);
        ListNode righthead = middle.next;
        middle.next = null;
        ListNode lefthead = head;

        lefthead = sortList(lefthead);
        righthead = sortList(righthead);
        return merge(lefthead, righthead);
    }
}
```



9. Merge K Sorted List

Code:

```
import java.util.List;
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        if (l1.val < l2.val) {
            l1.next = mergeTwoLists(l1.next, l2);
            return l1;
        } else {
            l2.next = mergeTwoLists(l1, l2.next);
            return l2;
        }
    }

    public ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0) return null;
        return divideAndConquer(lists, 0, lists.length - 1);
    }

    private ListNode divideAndConquer(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);
    }
}
```