**Name:** Mayank Singh

**UID:** 22BCS10205

**Section:** 22BCS\_IOT-612-B

**Subject:** Advanced Programming Lab-2

**Assignment**

1. **Code: (Print linked list)**

class Solution {

// Function to display the elements of a linked list in the same line

void printList(Node head) {

Node current = head;

while (current != null) {

System.out.print(current.data);

current = current.next;

if (current != null) {

System.out.print(" ");

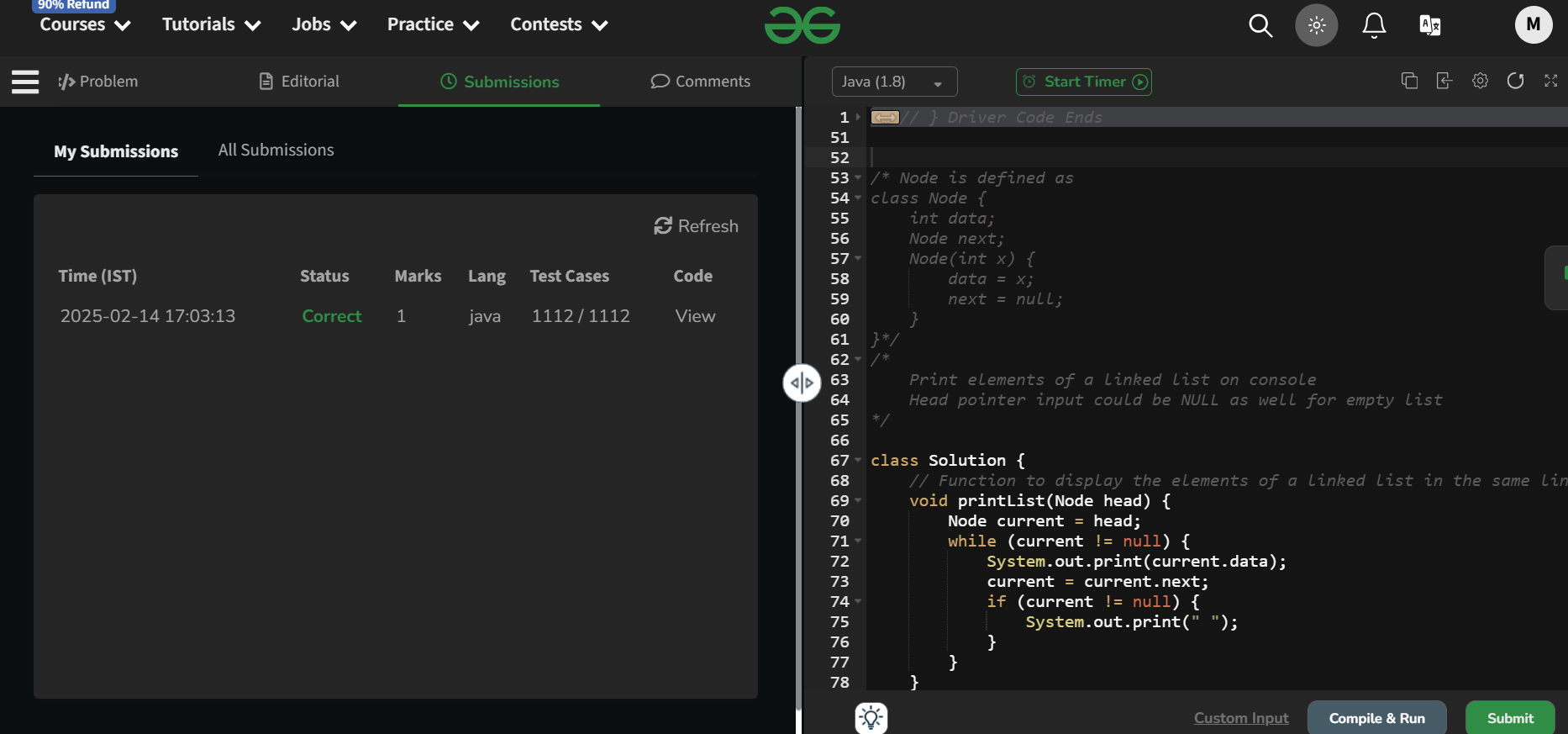
}

}

}

}

**Output:**

****

1. **Code: (Remove linked list from sorted list)**

class Solution {

    public ListNode deleteDuplicates(ListNode head) {

        if (head == null || head.next == null) {

            return head;

        }

        ListNode current = head;

        while (current != null && current.next != null) {

            if (current.val == current.next.val) {

                current.next = current.next.next;

            } else {

                current = current.next;

            }

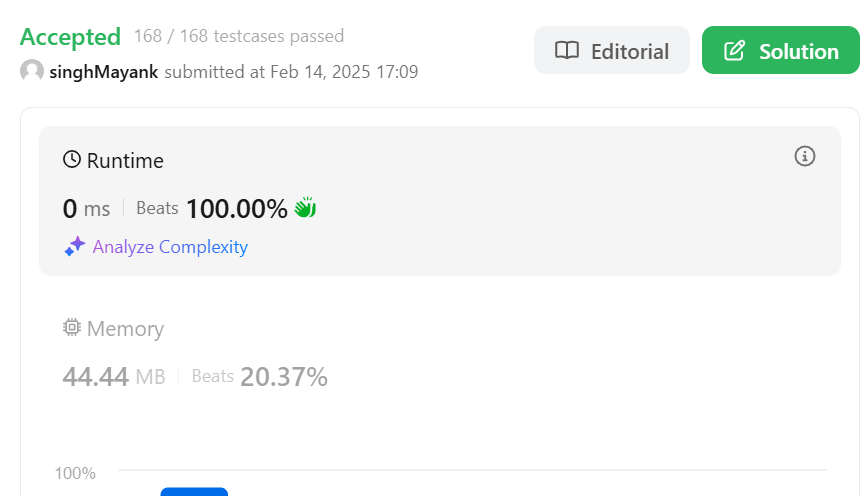
        }

        return head;

    }

}

**OUTPUT:**

****

1. **Code: (Reverse the linked list)**

class Solution {

    public ListNode reverseList(ListNode head) {

        ListNode prev = null;

        ListNode curr = head;

        ListNode next = null;

        while (curr != null) {

            next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

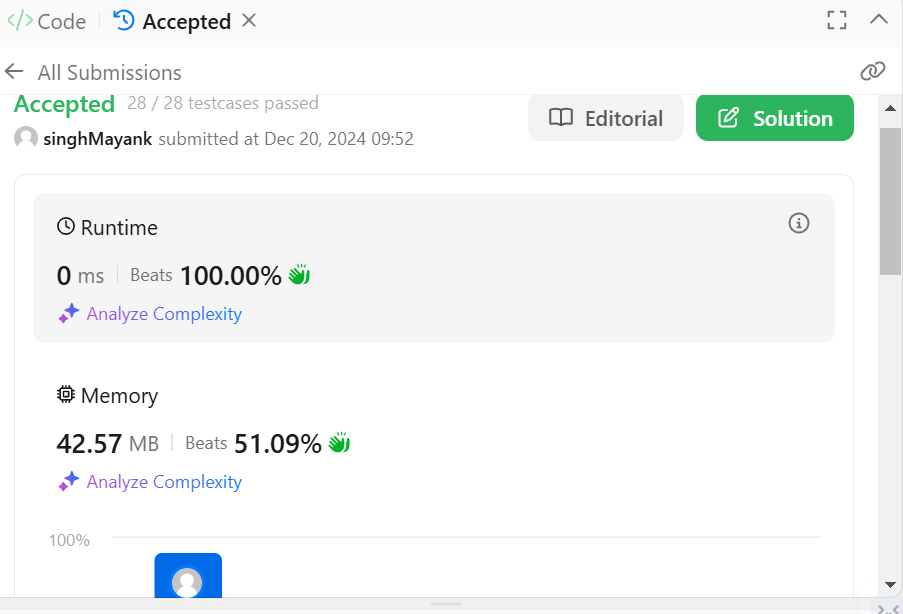
        }

        return prev;

    }

}

**Output:**

****

1. **Code: (Delete middle node of a list)**

class Solution {

    public ListNode deleteMiddle(ListNode head) {

        if (head == null || head.next == null) {

            return null;

        }

        ListNode current = head;

        int size = 0;

        while (current != null) {

            size++;

            current = current.next;

        }

        int mid = size / 2;

        current = head;

        ListNode prev = null;

        for (int i = 0; i < mid; i++) {

            prev = current;

            current = current.next;

        }

        if (prev != null) {

            prev.next = current.next;

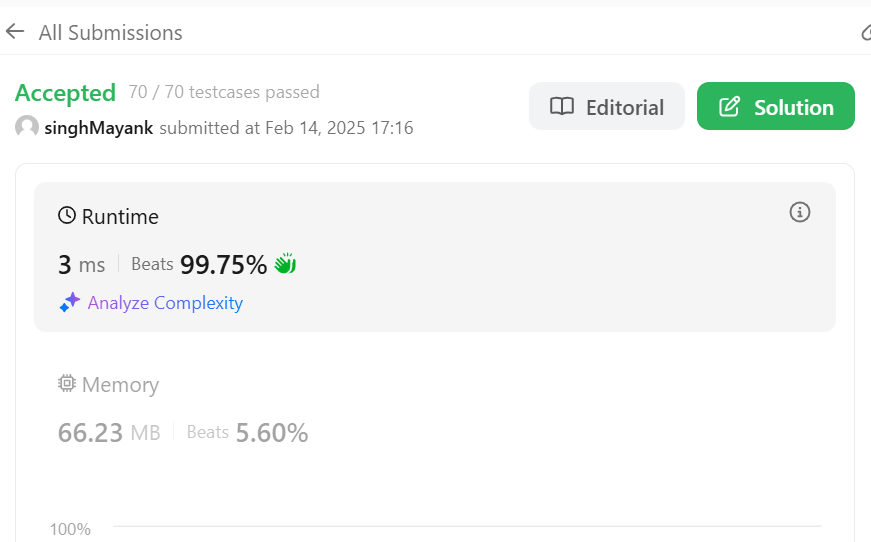
        }

        return head;

    }

}

**Output:**

****

1. **Code: (Merge two sorted list)**

class Solution {

    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

        ListNode dummy = new ListNode(0);

        ListNode current = dummy;

        while (list1 != null && list2 != null) {

            if (list1.val <= list2.val) {  current.next = list1;

                list1 = list1.next;

            } else {

                current.next = list2;

                list2 = list2.next;

            }

            current = current.next;

        }

        if (list1 != null) {

            current.next = list1;

        } else if (list2 != null) {

            current.next = list2;

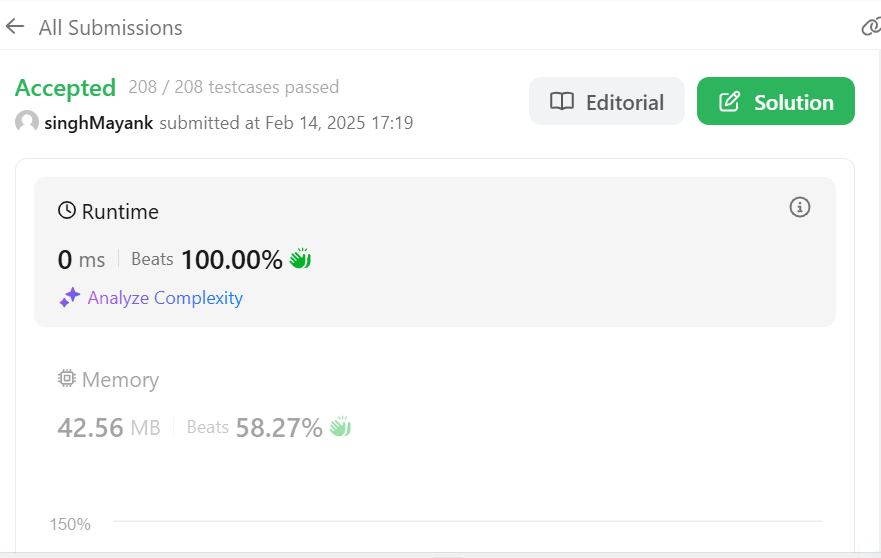
        }

        return dummy.next;

    }

}

**Output:**

****

1. **Code: (**[**Remove Duplicates from Sorted List II**](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/)**)**

class Solution {

    public ListNode deleteDuplicates(ListNode head) {

        ListNode dummy = new ListNode(0);

        dummy.next = head;

        ListNode prev = dummy;

        ListNode current = head;

        while (current != null) {

            while (current.next != null && current.val == current.next.val) {

                current = current.next;

            }

            if (prev.next == current) {

                prev = prev.next;

            } else {

                prev.next = current.next;

            }

            current = current.next;

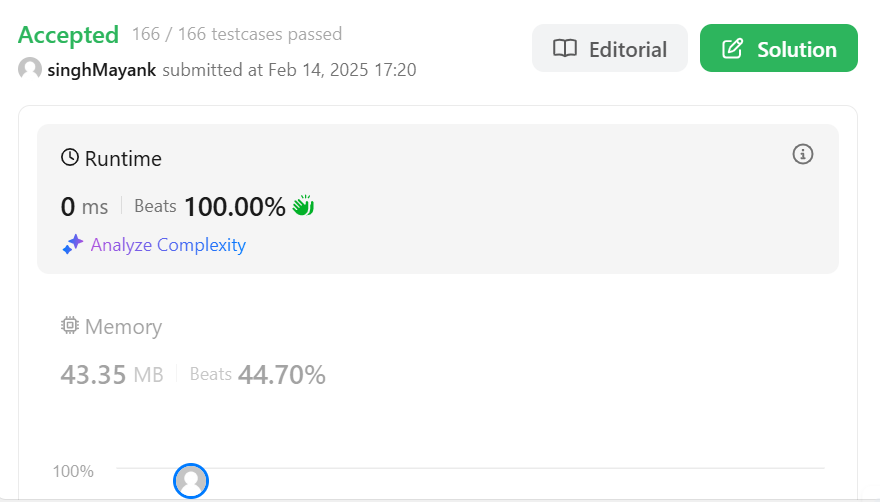
        }

        return dummy.next;

    }

}

**Output:**

****

1. **Code: (**[**Linked List Cycle**](https://leetcode.com/problems/linked-list-cycle/)**)**

public class Solution {

    public boolean hasCycle(ListNode head) {

        if (head == null) {

            return false;

        }

        ListNode slow = head;

        ListNode fast = head;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) {

                return true;

            }

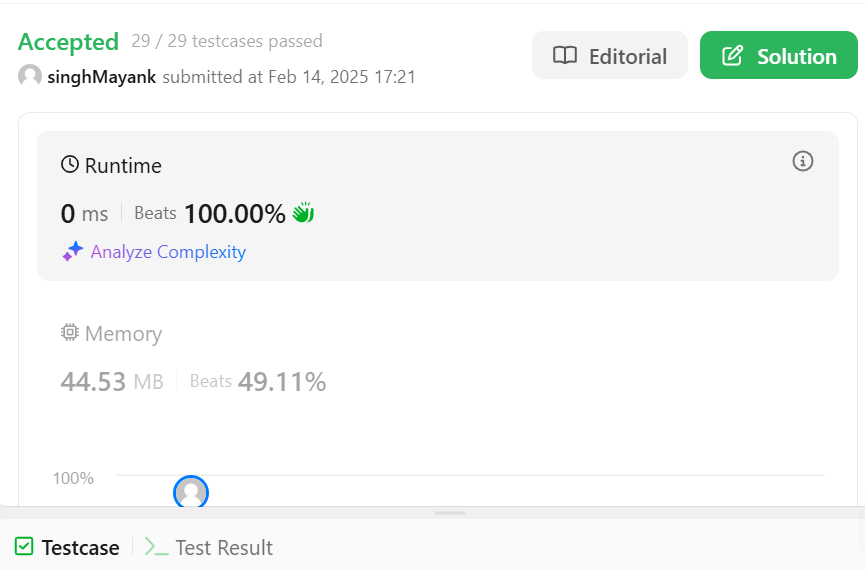
        }

        return false;

    }

}

**Output:**

****

1. **Code: (**[**Reverse Linked List II**](https://leetcode.com/problems/reverse-linked-list-ii/)**)**

class Solution {

    public ListNode reverseBetween(ListNode head, int left, int right) {

        if (head == null || left == right) return head;

        ListNode dummy = new ListNode(0);

        dummy.next = head;

        ListNode pre = dummy;

        for (int i = 1; i < left; i++) {

            pre = pre.next;

        }

        ListNode start = pre.next;

        ListNode then = start.next;

        for (int i = 0; i < right - left; i++) {

            start.next = then.next;

            then.next = pre.next;

            pre.next = then;

            then = start.next;

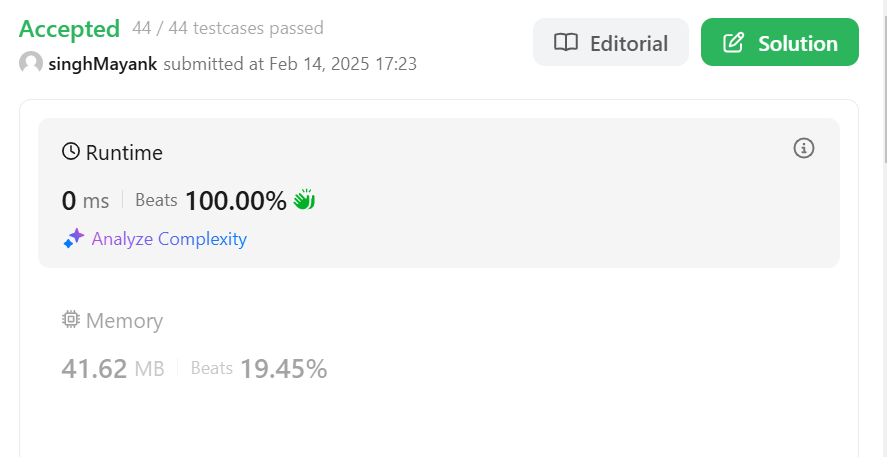
        }

        return dummy.next;

    }

}

**Output:**

****

1. **Code: (**[**Rotate List**](https://leetcode.com/problems/rotate-list/)**)**

class Solution {

    public ListNode rotateRight(ListNode head, int k) {

        if (head == null || head.next == null || k == 0) return head;

        ListNode current = head;

        int length = 1;

        while (current.next != null) {

            current = current.next;

            length++;

        }

        current.next = head;

        k = k % length;

        int stepsToNewHead = length - k;

        ListNode newTail = head;

        for (int i = 0; i < stepsToNewHead - 1; i++) {

            newTail = newTail.next;

        }

        ListNode newHead = newTail.next;

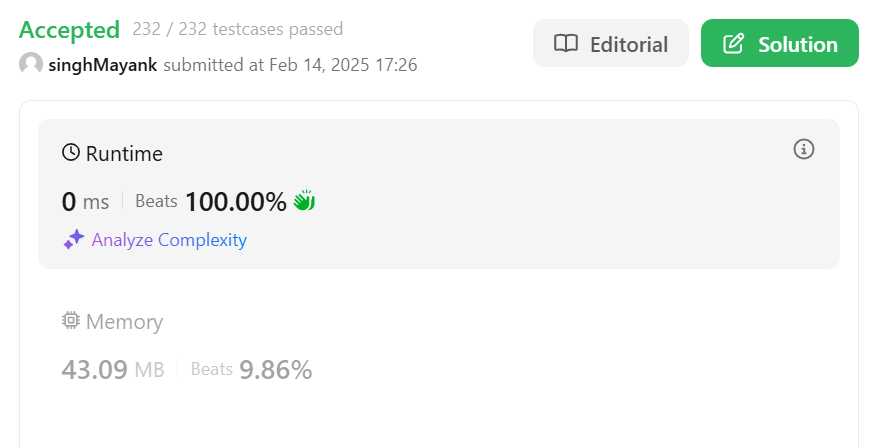
        newTail.next = null;

        return newHead;

    }

}

**Output:**

****

1. **Code: (**[**Sort List**](https://leetcode.com/problems/sort-list/)**)**

class Solution {

    public ListNode sortList(ListNode head) {

        if (head == null || head.next == null) {

            return head;

        }

        ListNode mid = getMid(head);

        ListNode left = sortList(head);

        ListNode right = sortList(mid);

        return merge(left, right);

    }

    private ListNode getMid(ListNode head) {

        ListNode slow = head;

        ListNode fast = head.next;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

        }

        ListNode mid = slow.next;

        slow.next = null;

        return mid;

    }

    private ListNode merge(ListNode l1, ListNode l2) {

        ListNode dummy = new ListNode(0);

        ListNode curr = dummy;

        while (l1 != null && l2 != null) {

            if (l1.val < l2.val) {

                curr.next = l1;

                l1 = l1.next;

            } else {

                curr.next = l2;

                l2 = l2.next;

            }

            curr = curr.next;

        }

        if (l1 != null) {

            curr.next = l1;

        } else {

            curr.next = l2;

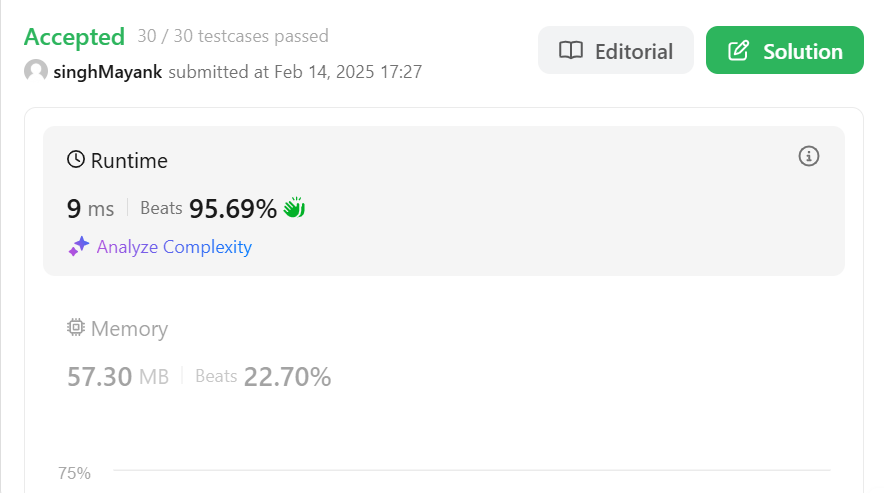
        }

        return dummy.next;

    }

}

**Output:**

****

1. **Code: (**[**Linked List Cycle II**](https://leetcode.com/problems/linked-list-cycle-ii/)**)**

public class Solution {

    public ListNode detectCycle(ListNode head) {

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

        ListNode slow = head;

        ListNode fast = head;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) {

                ListNode entry = head;

                while (entry != slow) {

                    entry = entry.next;

                    slow = slow.next;

                }

                return entry;

            }

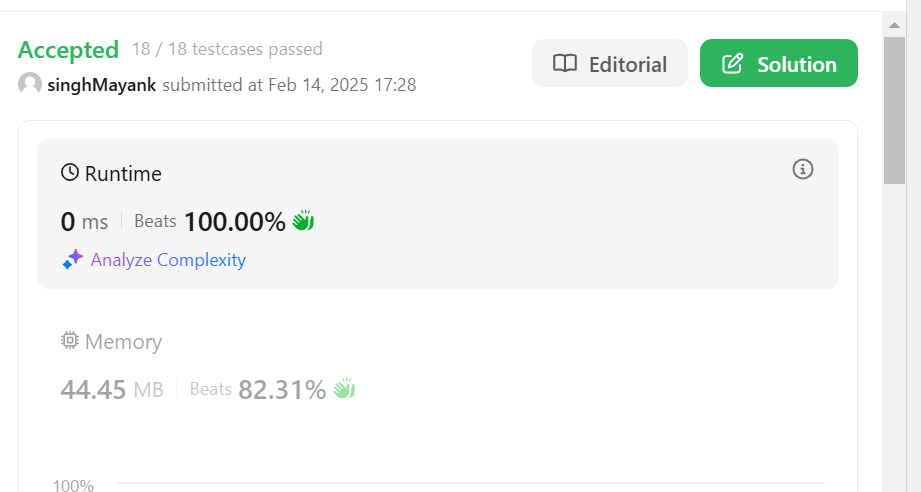
        }

        return null;

    }

}

**Output:**

****