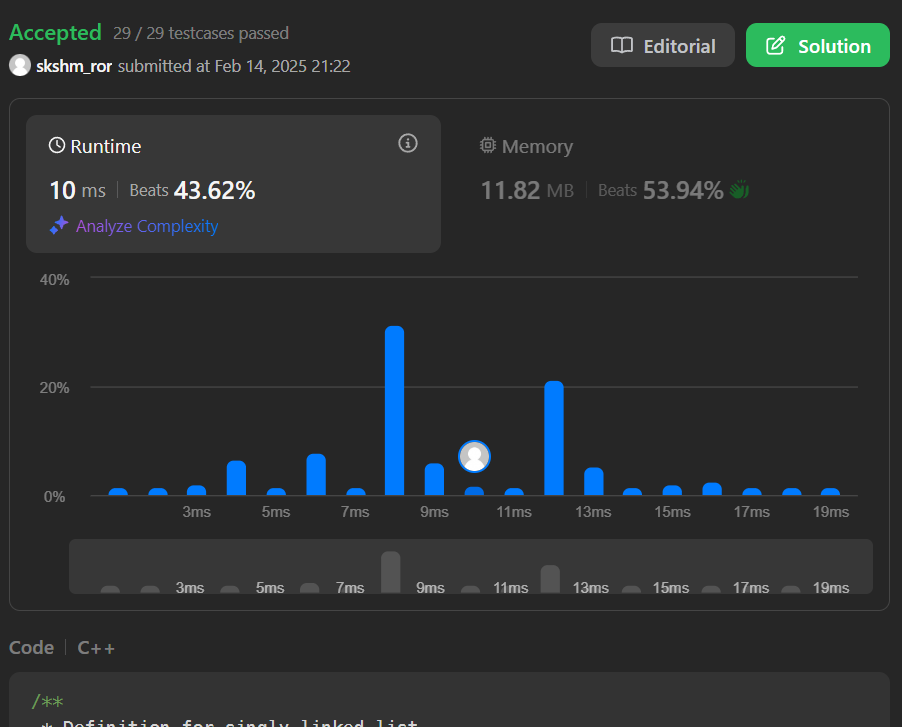
[**141. Linked List Cycle**](https://leetcode.com/problems/linked-list-cycle/)



Code:

class Solution {

 public:

  bool hasCycle(ListNode\* head) {

    ListNode\* slow = head;

    ListNode\* fast = head;

    while (fast != nullptr && fast->next != nullptr) {

      slow = slow->next;

      fast = fast->next->next;

      if (slow == fast)

        return true;

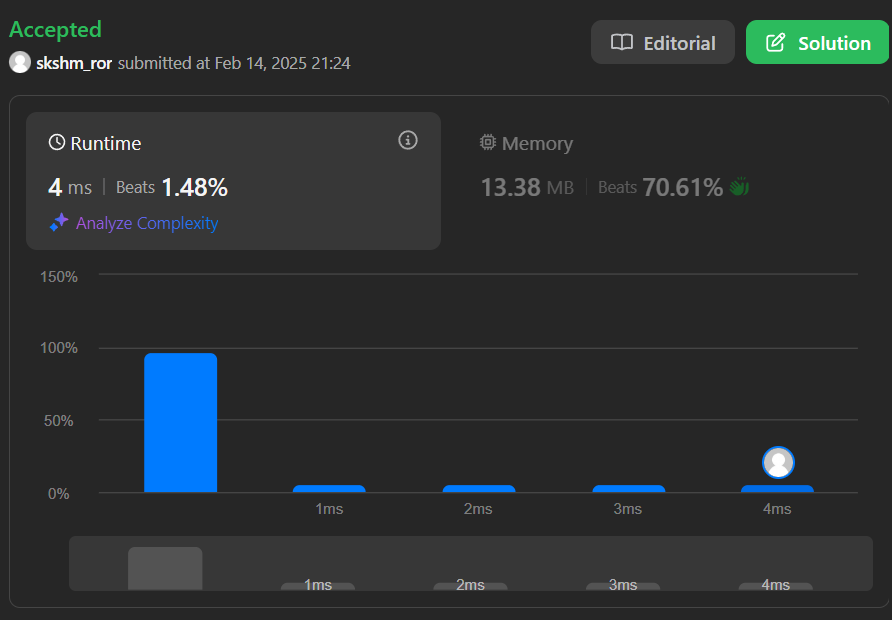
    }

    return false;

  }

};

[**206. Reverse Linked List**](https://leetcode.com/problems/reverse-linked-list/)



Code:

class Solution {

public:

    ListNode\* reverseList(ListNode\* head)

     {

        ListNode \*pre = new ListNode(0),

        \*cur = head;

        pre -> next = head;

        while (cur && cur -> next)

         {

            ListNode\* temp = pre -> next;

            pre -> next = cur -> next;

            cur -> next = cur -> next -> next;

            pre -> next -> next = temp;

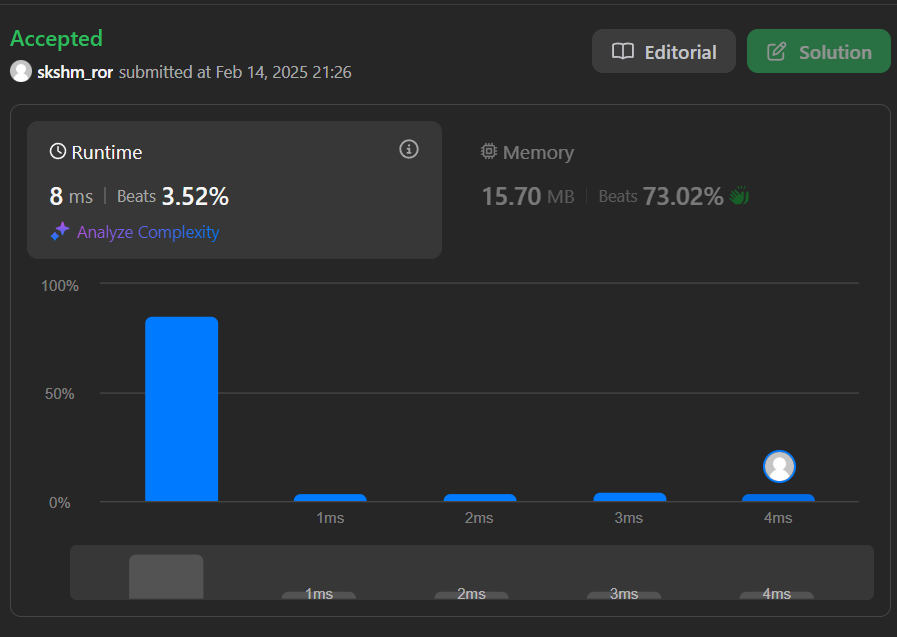
        }

        return pre -> next;

    }

};

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



Code:

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        if (head == nullptr || head->next == nullptr) {

            return head;

        }

        ListNode\* dummy = new ListNode(0, head);

        ListNode\* prev = dummy;

        ListNode\* curr = head;

        ListNode\* temp = head->next;

        bool flag = false;

        while (temp != nullptr) {

            if (curr->val != temp->val) {

                if (flag) {

                    prev->next = temp;

                    flag = false;

                } else {

                    prev = prev->next;

                }

            } else {

                flag = true;

            }

            temp = temp->next;

            curr = curr->next;

        }

        if (flag) {

            prev->next = temp;

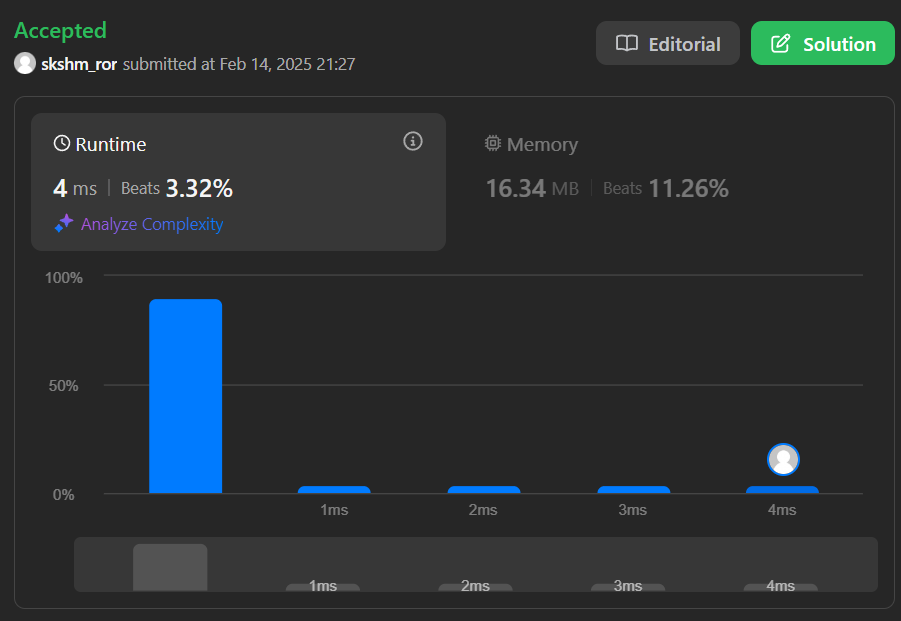
        }

        return dummy->next;

    }

};

[**83. Remove Duplicates from Sorted List**](https://leetcode.com/problems/remove-duplicates-from-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;

            }

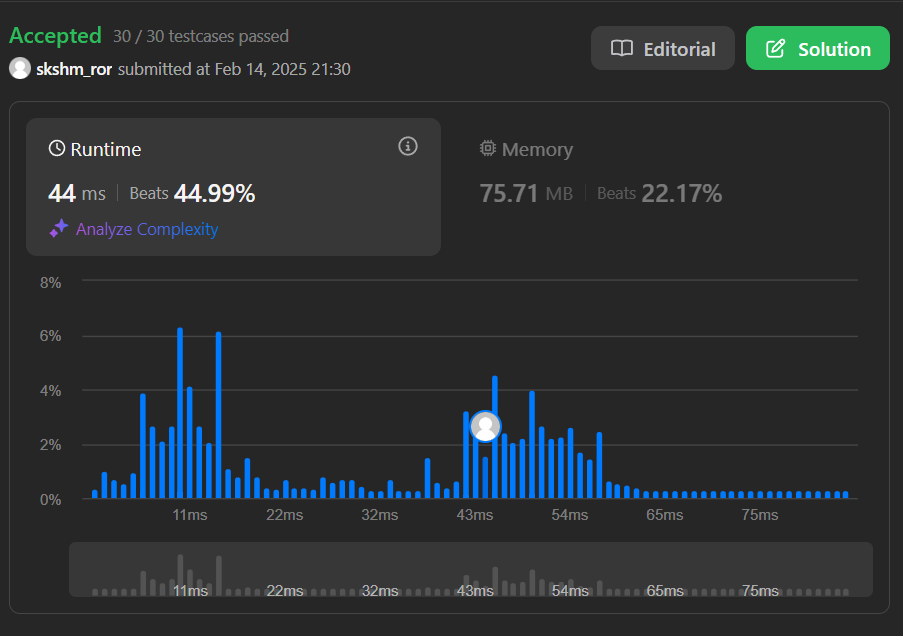
        }

        return head;

    }

};

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



Code:

class Solution {

public:

    ListNode\* getmid(ListNode\* head) {

        ListNode\* slow = head;

        ListNode\* fast = head->next;

        while (fast != NULL && fast->next != NULL) {

            slow = slow->next;

            fast = fast->next->next;

        }

        return slow;

    }

ListNode\* merge(ListNode\* left, ListNode\* right) {

        if (left == NULL)

            return right;

        if (right == NULL)

            return left;

        ListNode\* dummy = new ListNode(0);

        ListNode\* temp = dummy;

        while (left != NULL && right != NULL) {

            if (left->val < right->val) {

                temp->next = left;

                temp = left;

                left = left->next;

            } else {

                temp->next = right;

                temp = right;

                right = right->next;

            }

        }

        while (left != NULL) {

            temp->next = left;

            temp = left;

            left = left->next;

        }

        while (right != NULL) {

            temp->next = right;

            temp = right;

            right = right->next;

        }

        dummy = dummy->next;

        return dummy;

    }

    ListNode\* sortList(ListNode\* head) {

        // using merge sort

        // base case

        if (head == NULL || head->next == NULL)

            return head;

        ListNode\* mid = getmid(head);

        ListNode\* left = head;

        ListNode\* right = mid->next;

        mid->next = NULL;

        left = sortList(left);

        right = sortList(right);

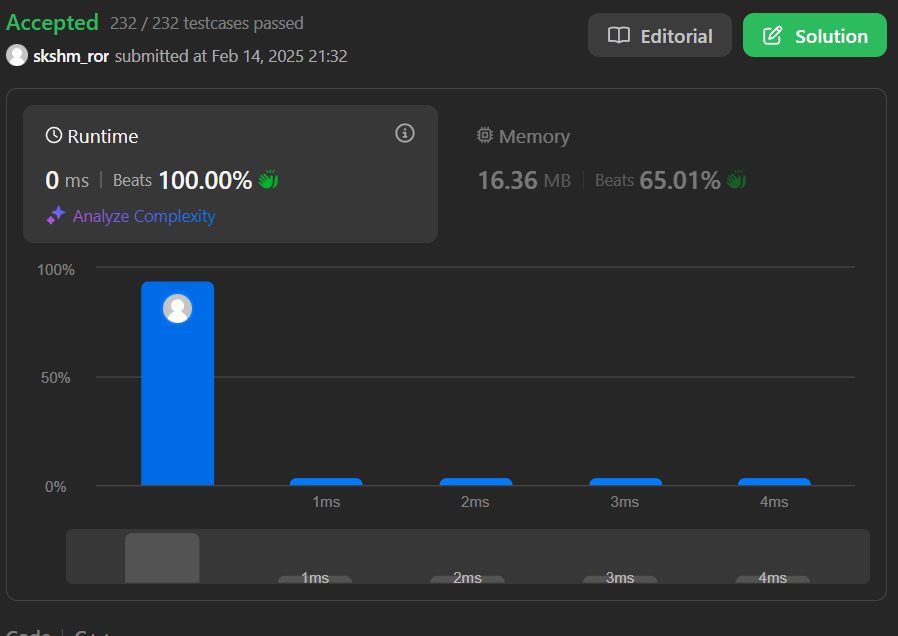
        ListNode\* result = merge(left, right);

        return result;

    }

};

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



Code:

class Solution {

 public:

  ListNode\* rotateRight(ListNode\* head, int k) {

    if (!head || !head->next || k == 0)

      return head;

    ListNode\* tail;

    int length = 1;

    for (tail = head; tail->next; tail = tail->next)

      ++length;

    tail->next = head;

    const int t = length - k % length;

    for (int i = 0; i < t; ++i)

      tail = tail->next;

    ListNode\* newHead = tail->next;

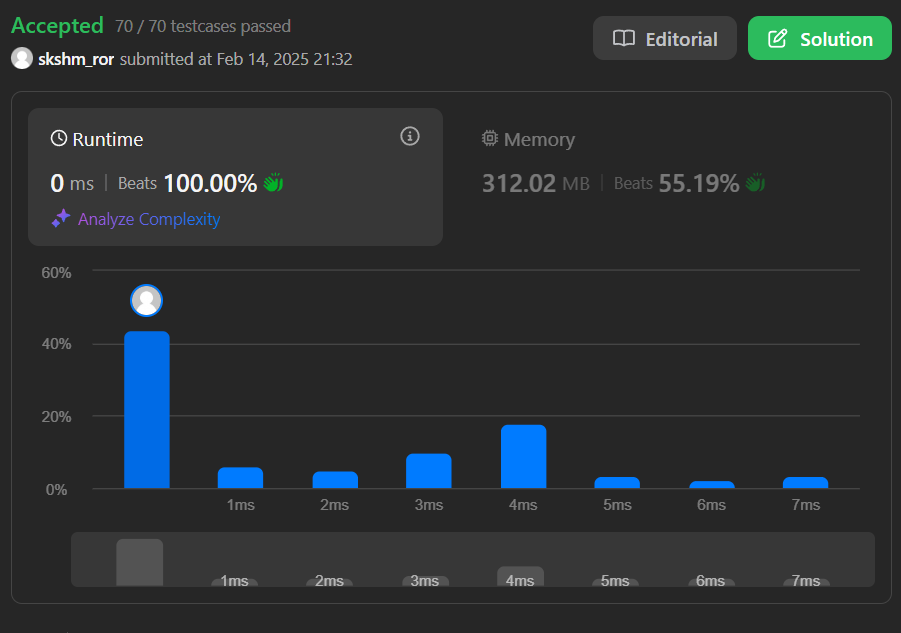
    tail->next = nullptr;

    return newHead;

  }

};

[**2095. Delete the Middle Node of a Linked List**](https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/)



Code:

class Solution {

public:

    ListNode\* deleteMiddle(ListNode\* head) {

        if (!head || !head->next) {

            return nullptr;

        }

        ListNode \*slow = head, \*fast = head;

        fast = fast->next->next;

        while (fast != nullptr && fast->next != nullptr) {

            slow = slow->next;

            fast = fast->next->next;

        }

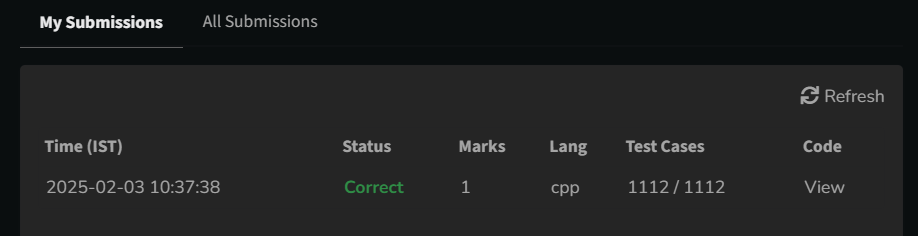
        slow->next = slow->next->next;

        return head;

    }

};

**Print Linked List**



Code:

class Solution {

public:

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

void printList(Node \*head) {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data;

if (temp->next != nullptr) {

cout << " "; // Print space only if there is a next element

}

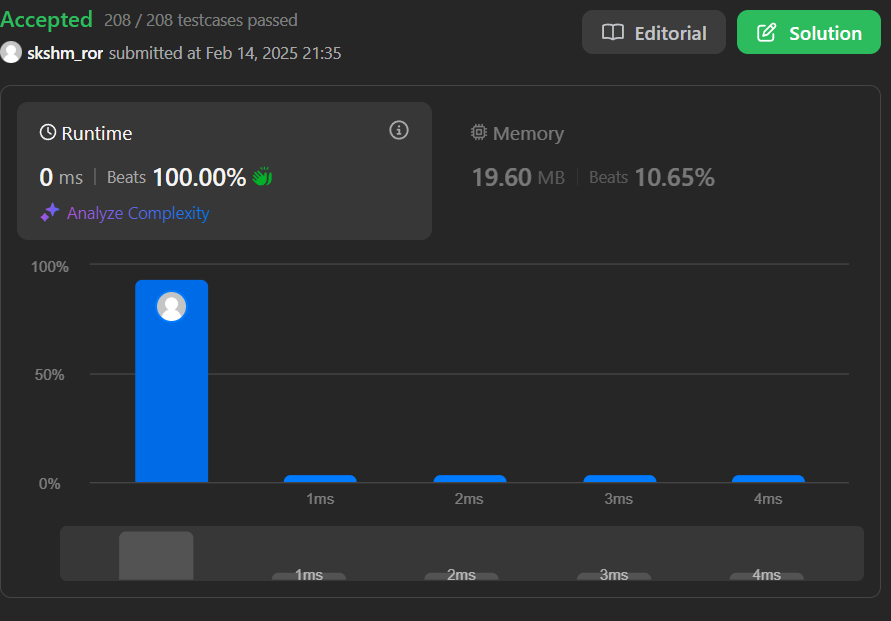
temp = temp->next;

}

}

};

[**21. Merge Two Sorted Lists**](https://leetcode.com/problems/merge-two-sorted-lists/)



Code:

class Solution {

public:

    ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

        if (!l1) return l2;  // If l1 is empty, return l2

        if (!l2) return l1;  // If l2 is empty, return l1

        if (l1->val <= l2->val) {

            l1->next = mergeTwoLists(l1->next, l2); // Merge rest of l1 with l2

            return l1;  // Return the smaller node

        } else {

            l2->next = mergeTwoLists(l1, l2->next); // Merge l1 with rest of l2

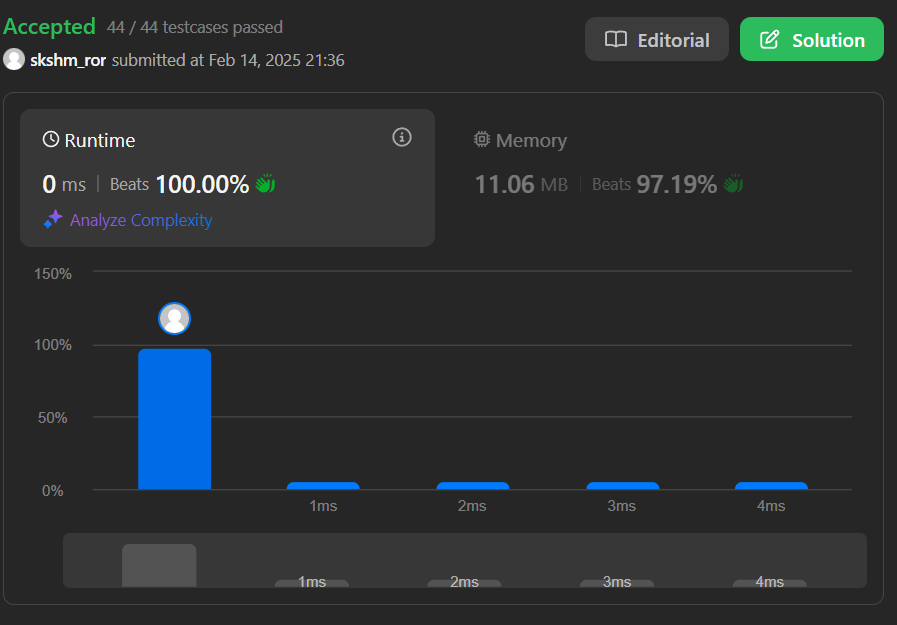
            return l2;  // Return the smaller node

        }

    }

};

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



/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode() : val(0), next(nullptr) {}

 \*     ListNode(int x) : val(x), next(nullptr) {}

 \*     ListNode(int x, ListNode \*next) : val(x), next(next) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* reverseBetween(ListNode\* head, int left, int right) {

        ListNode \*dummy = new ListNode(0); // created dummy node

        dummy->next = head;

        ListNode \*prev = dummy; // intialising prev pointer on dummy node

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

            prev = prev->next; // adjusting the prev pointer on it's actual index

        ListNode \*curr = prev->next; // curr pointer will be just after prev

        // reversing

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

            ListNode \*forw = curr->next; // forw pointer will be after curr

            curr->next = forw->next;

            forw->next = prev->next;

            prev->next = forw;

        }

        return dummy->next;

    }

};