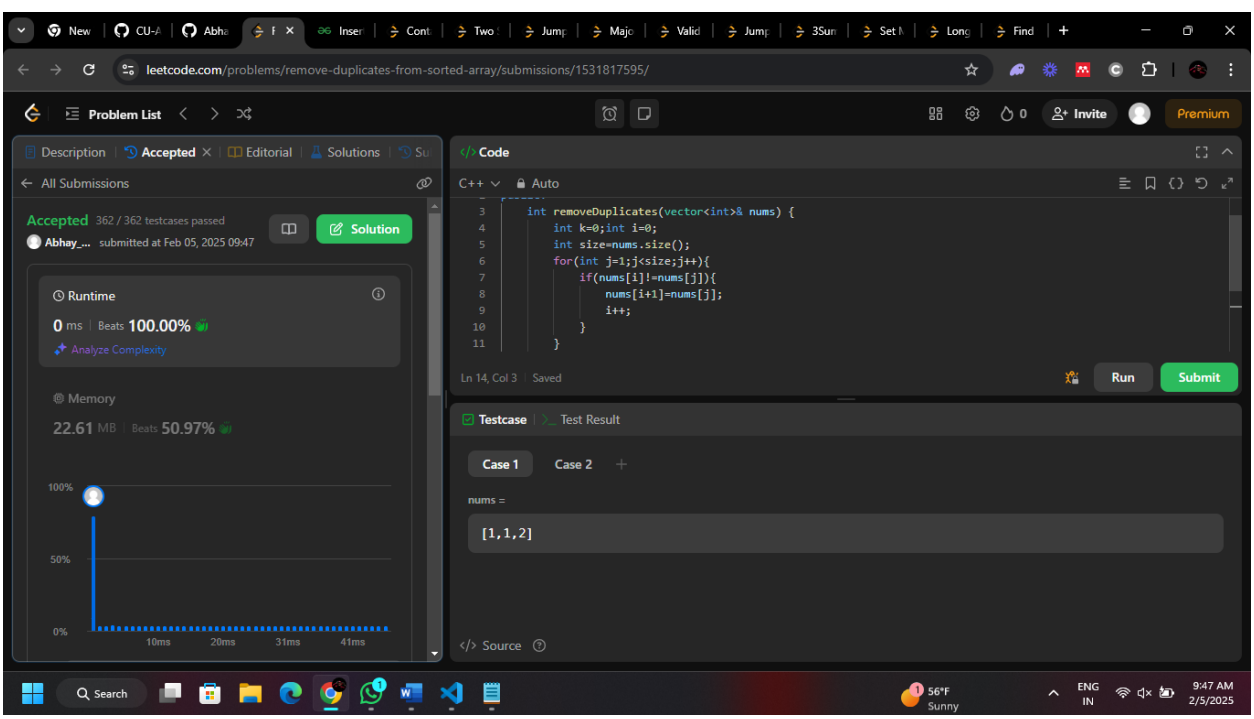
Assignment 1

Student Name: Abhay Kejriwal
UID: 22BCS14663
Branch: BE-CSE
Section/Group: IOT-601-B
Semester: 6th
Date of Performance: 22/07/24
Subject Name: Advanced Programming
Subject Code: 22CSP – 351

DAY-1:

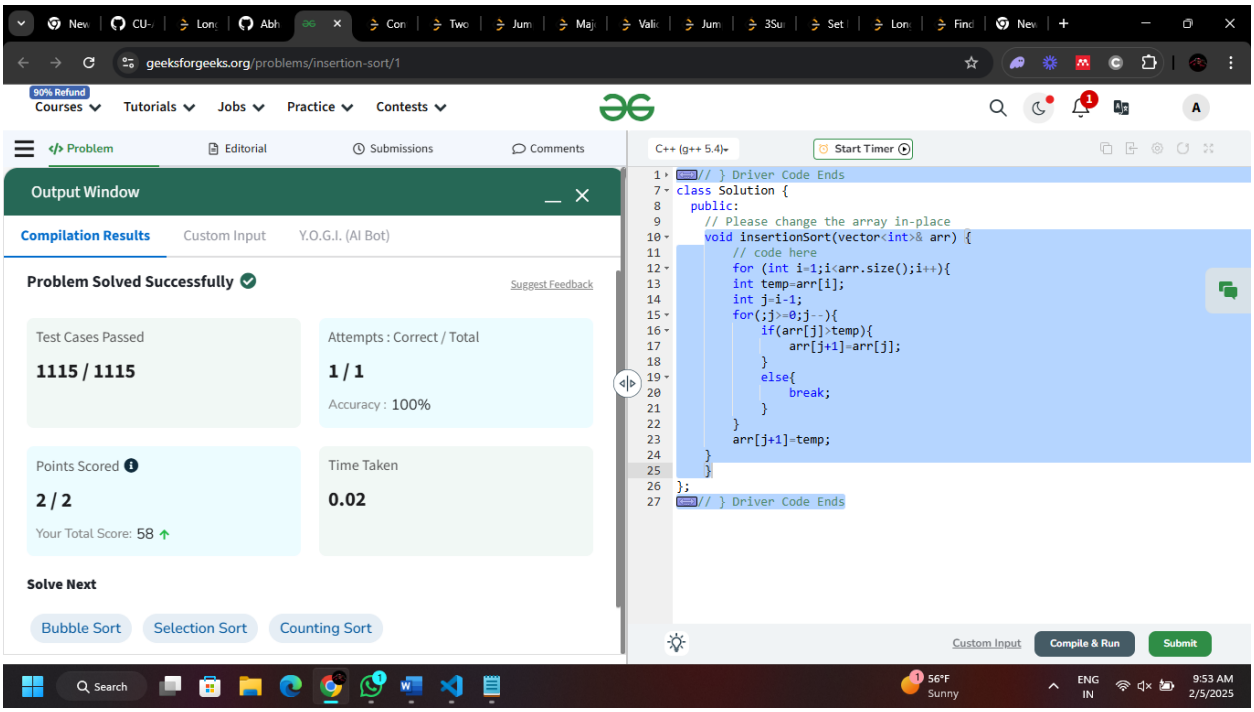
1. Remove Duplicates From The Sorted Array-

```
int removeDuplicates(vector<int>& nums) {
    int k=0;int i=0;
    int size=nums.size();
    for(int j=1;j<size;j++){
        if(nums[i]!=nums[j]){
            nums[i+1]=nums[j];
            i++;
        }
    }
    return i+1;
}
```



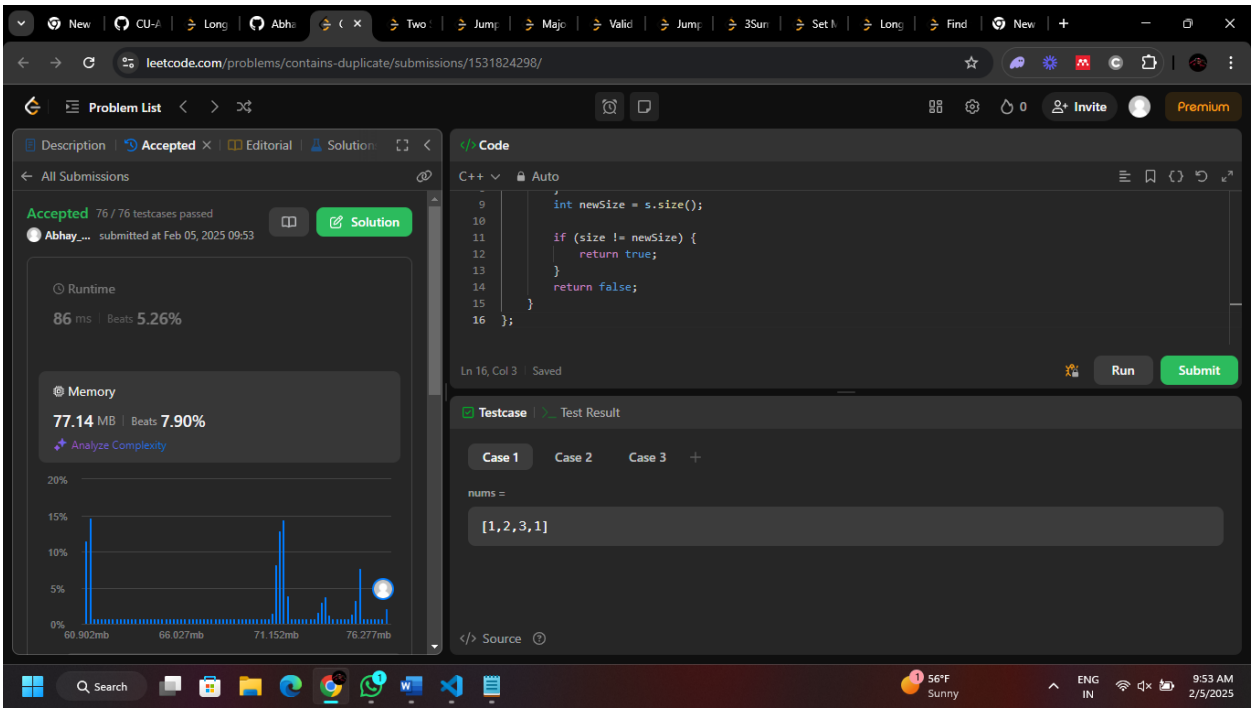
2. Implementing Insertion Sort-

```
void insertionSort(vector<int>& arr) {
    // code here
    for (int i=1;i<arr.size();i++){
    int temp=arr[i];
    int j=i-1;
    for(;j>=0;j--){
      if(arr[j]>temp){
        arr[j+1]=arr[j];
      }
      else{
        break;
      }
    }
    arr[j+1]=temp;
  }
}
```



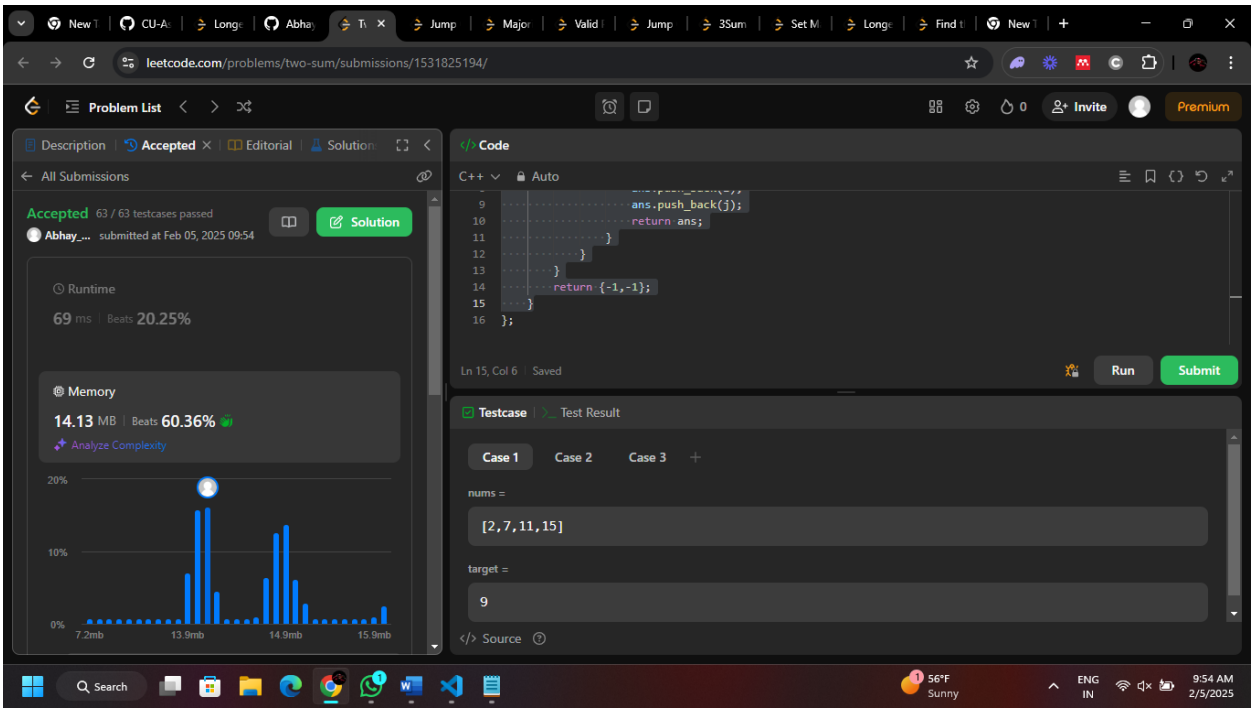
3. Contains Duplicate-

```
bool containsDuplicate(vector<int>& nums) {
    int size = nums.size();
    set<int> s;
    for (auto it : nums) {
        s.insert(it);
    }
    int newSize = s.size();

    if (size != newSize) {
        return true;
    }
    return false;
}
```



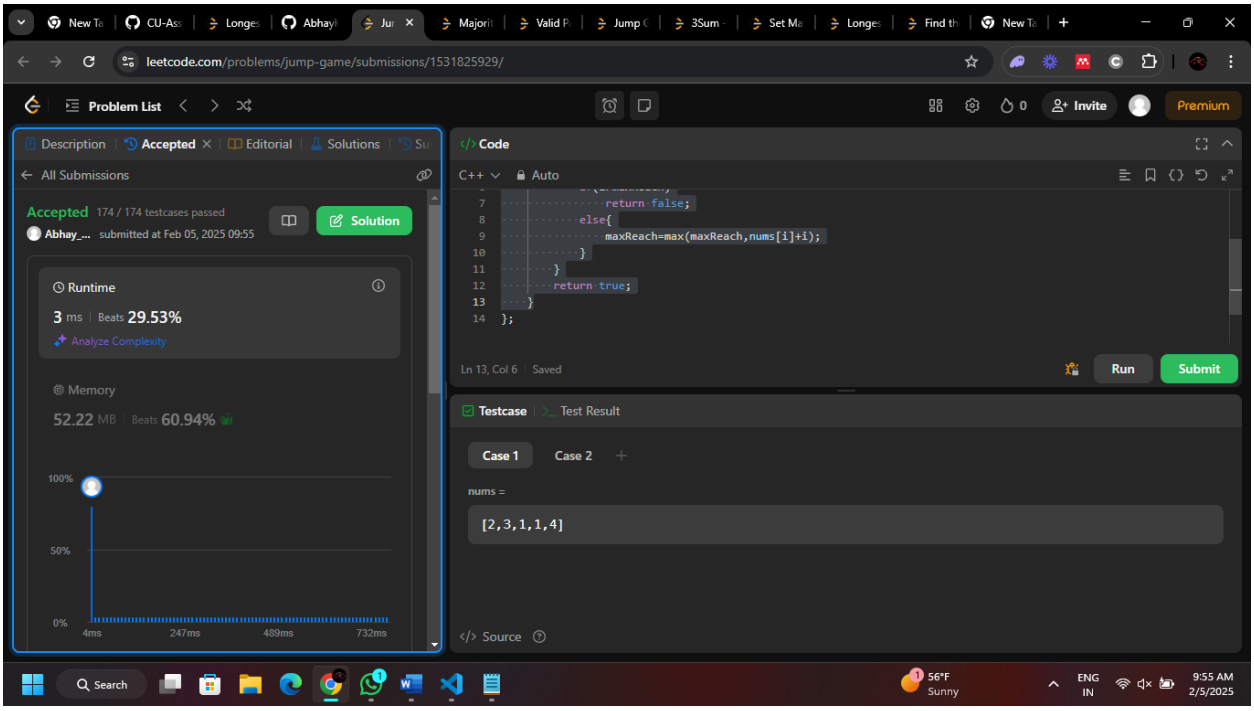
4. Two Sum –

```
vector<int> twoSum(vector<int>& nums, int target) {
    vector<int> ans;
    for(int i=0;i<nums.size();i++){
        for(int j=i+1;j<nums.size();j++){
            if(nums[i]+nums[j]==target){
                ans.push_back(i);
                ans.push_back(j);
                return ans;
            }
        }
    }
    return {-1,-1};
}
```



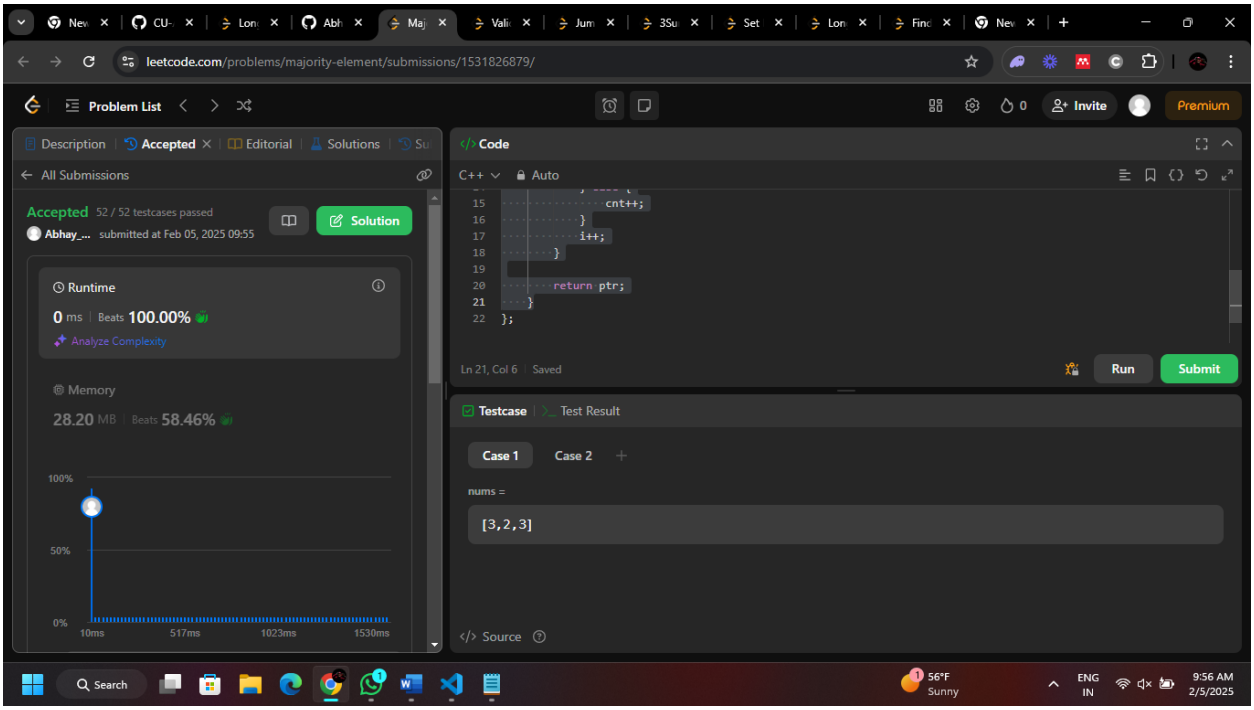
5. Jump Game-

```
bool canJump(vector<int>& nums) {
    int maxReach=0;
    for(int i=0;i<nums.size();i++){
        if(i>maxReach)
            return false;
        else{
            maxReach=max(maxReach,nums[i]+i);
        }
    }
    return true;
}
```



6. Majority Element-

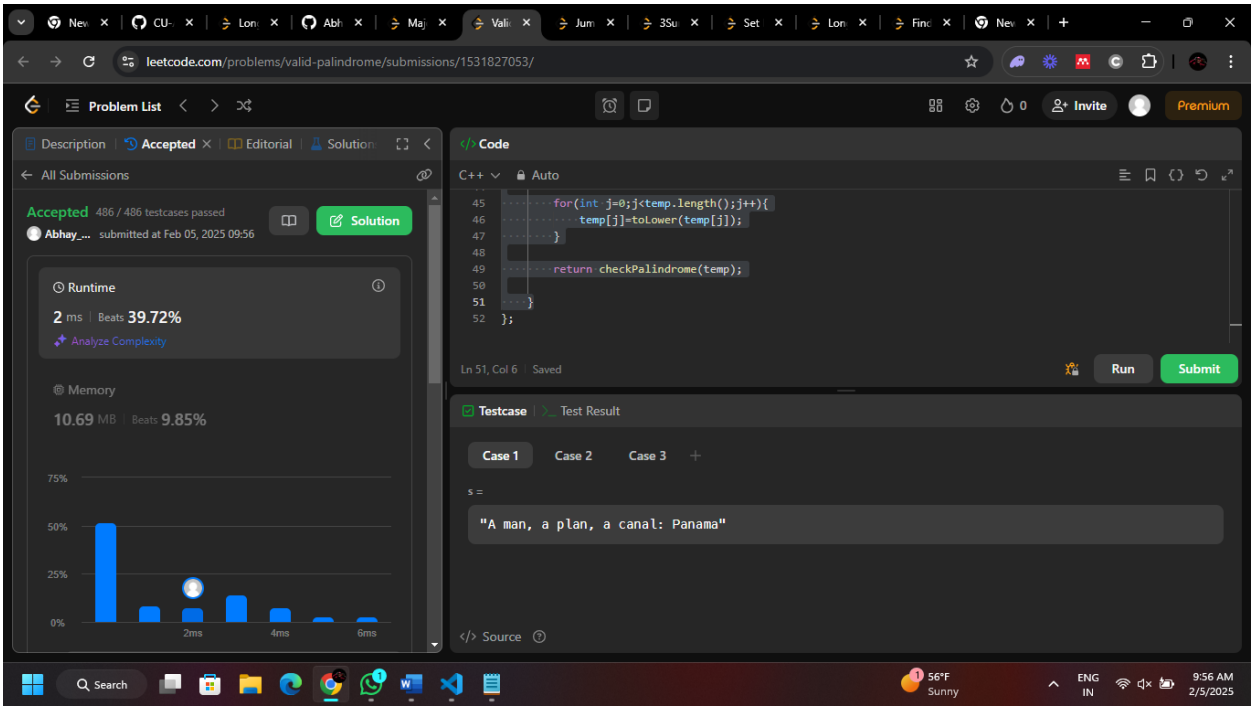
```
int majorityElement(vector<int>& nums) {
    int cnt = 0;
    int i = 0;
    int ptr = 0;

    while (i < nums.size()) {
        if (cnt == 0) {
            ptr = nums[i];
        }
        if (ptr != nums[i]) {
            cnt--;
        } else {
            cnt++;
        }
        i++;
    }
    return ptr;
}
```



7. Valid Palindrome –

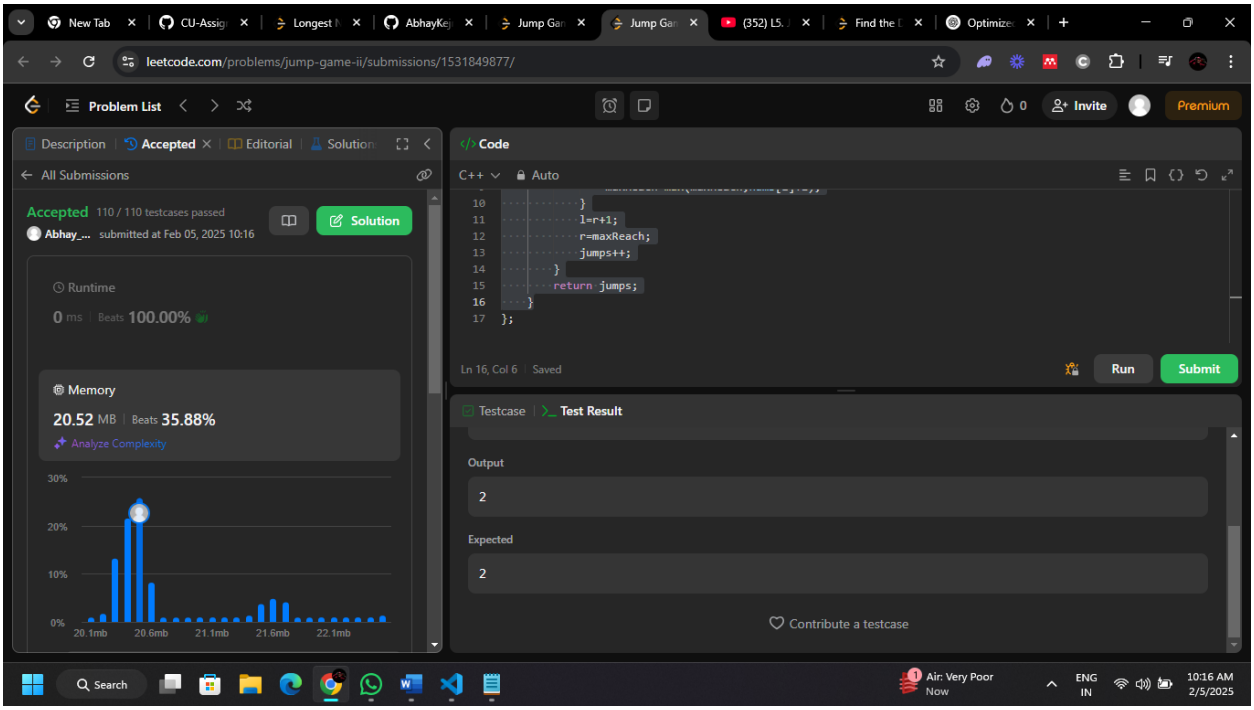
```
bool valid(char ch){
    if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || (ch>='0' && ch<='9')){
        return 1;
    }
    return 0;
}

char toLower(char ch){
    if((ch>='a' && ch<='z') || (ch>='0' && ch<='9')){
        return ch;
    }
    else{
        char temp= ch-'A'+'a';
        return temp;
    }
}

bool checkPalindrome(string a){
    int s=0;
    int e=a.length()-1;

    while(s<=e){
        if(a[s] != a[e]){
            return 0;
        }
        else{
            s++;
            e--;
        }
    }
    return 1;
}

public:
    bool isPalindrome(string s) {
        string temp="";
        for(int j=0;j<s.length();j++){
```

```
            if(valid(s[j])){
                temp.push_back(s[j]);
            }
        }

        for(int j=0;j<temp.length();j++){
            temp[j]=toLower(temp[j]);
        }

        return checkPalindrome(temp);
    }
}
```



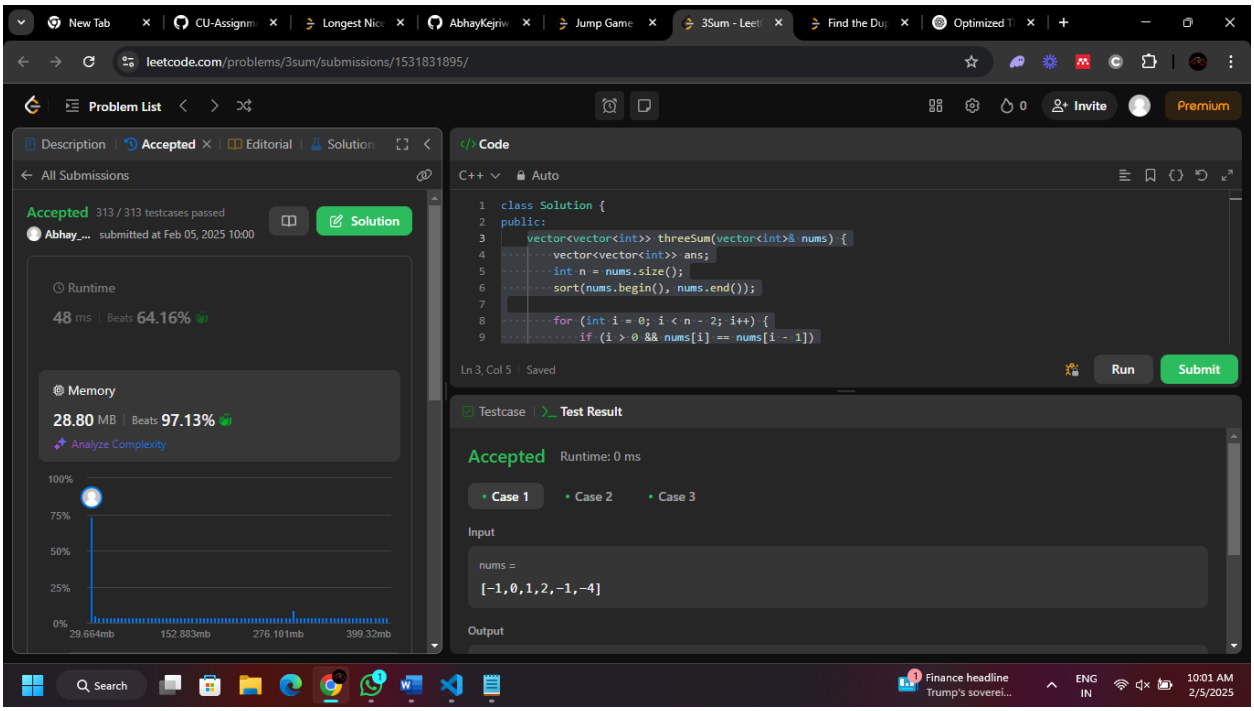
8. Jump Game 2-

```
int jump(vector<int>& nums) {
    int jumps=0;
    int l=0,r=0;
    while(r<nums.size()-1){
        int maxReach=0;
        for(int i=l;i<=r;i++){
            maxReach=max(maxReach,nums[i]+i);
        }
        l=r+1;
        r=maxReach;
        jumps++;
    }
    return jumps;
}
```



9. 3 Sum-

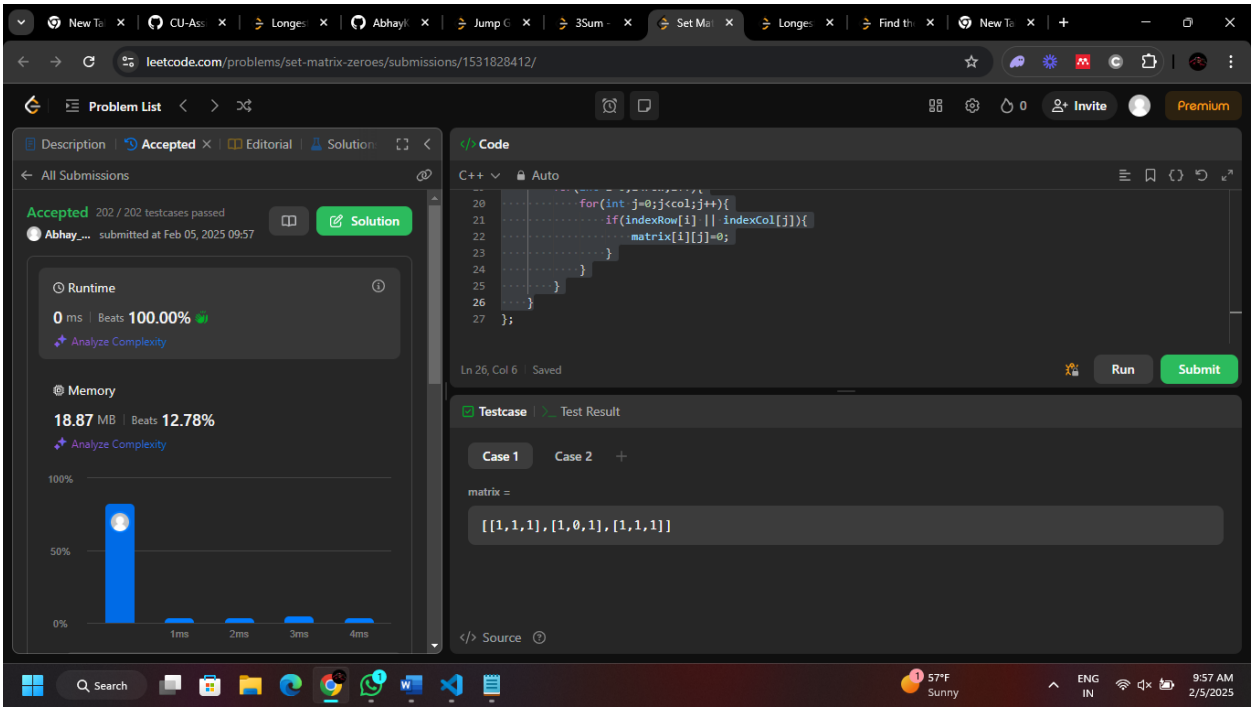
```
vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> ans;
    int n = nums.size();
    sort(nums.begin(), nums.end());

    for (int i = 0; i < n - 2; i++) {
        if (i > 0 && nums[i] == nums[i - 1])
            continue;
        int left = i + 1, right = n - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];

            if (sum == 0) {
                ans.push_back({nums[i], nums[left], nums[right]});

                while (left < right && nums[left] == nums[left + 1])
                    left++;
                while (left < right && nums[right] == nums[right - 1])
                    right--;

                left++, right--;
            } else if (sum < 0) {
                left++; // Increase the sum by moving left pointer
            } else {
                right--; // Decrease the sum by moving right pointer
            }
        }
    }
    return ans;
}
```



10. Set Matrix Zeros-

```
void setZeroes(vector<vector<int>>& matrix) {
    int row=matrix.size();
    int col=matrix[0].size();

    vector<int> indexRow(row,0);
    vector<int> indexCol(col,0);

    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(matrix[i][j]==0){
                indexRow[i] =1;
                indexCol[j]=1;
            }
        }
    }

    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(indexRow[i] || indexCol[j]){
```

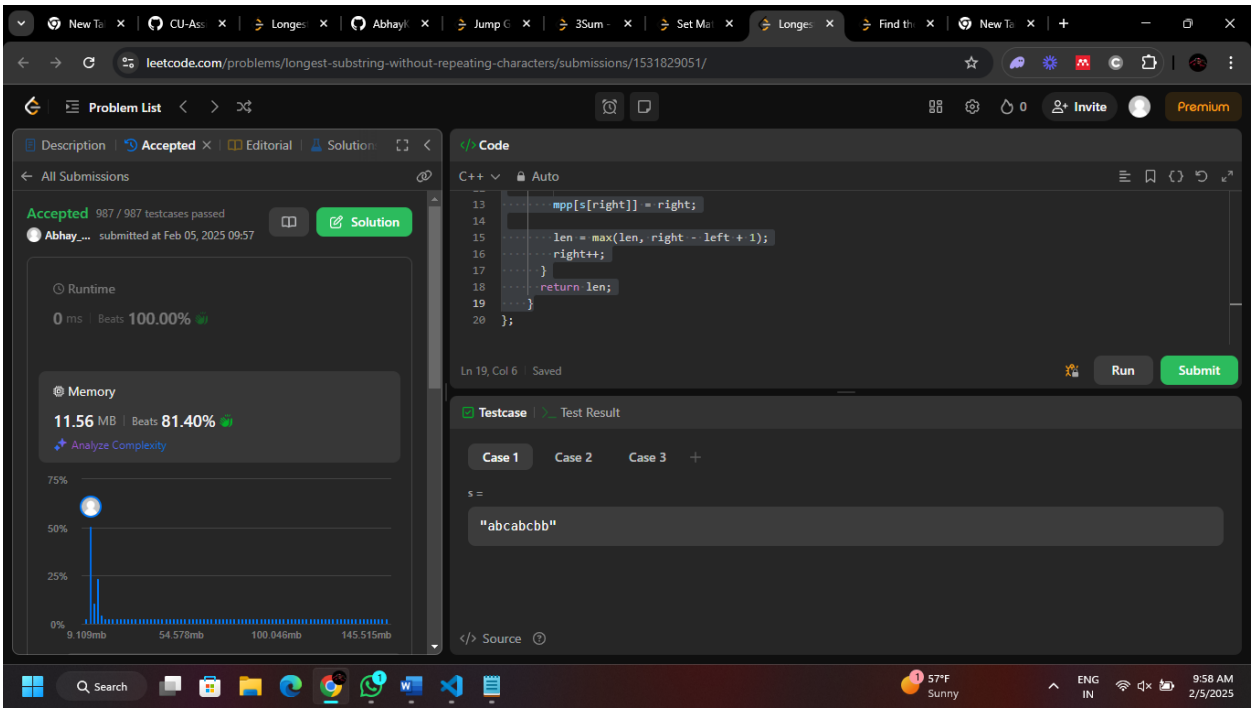
```
                matrix[i][j]=0;
            }
        }
    }
}
```



11. Longest Substring Without Repeating Characters-

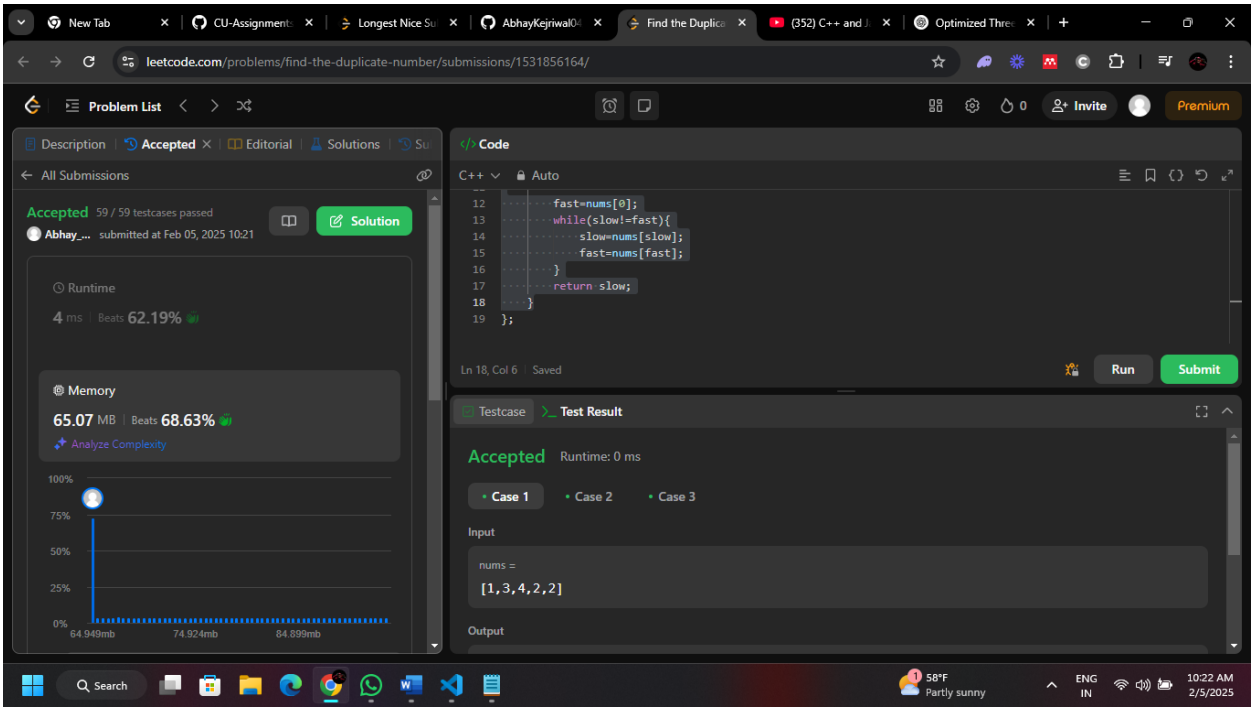
```
int lengthOfLongestSubstring(string s) {
    vector < int > mpp(256, -1);

    int left = 0, right = 0;
    int n = s.size();
    int len = 0;
    while (right < n) {
      if (mpp[s[right]] != -1)
        left = max(mpp[s[right]] + 1, left);

      mpp[s[right]] = right;

      len = max(len, right - left + 1);
```

```
        right++;
    }
    return len;
}
```



12. Finding Duplicate Number-

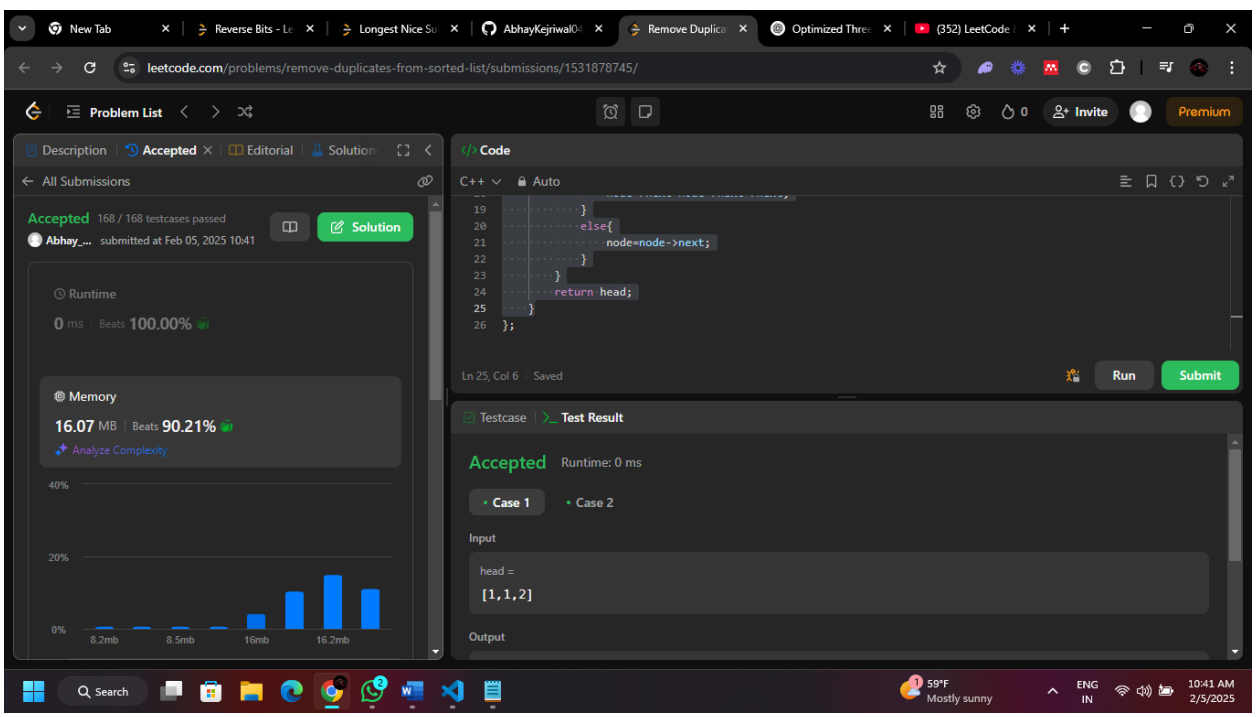
```
int findDuplicate(vector<int>& nums) {
    int slow=nums[0];
    int fast=nums[0];

    do{
        slow=nums[slow];
        fast=nums[nums[fast]];
    }while(slow!=fast);

    fast=nums[0];
    while(slow!=fast){
        slow=nums[slow];
        fast=nums[fast];
    }
    return slow;
}
```



DAY-2:

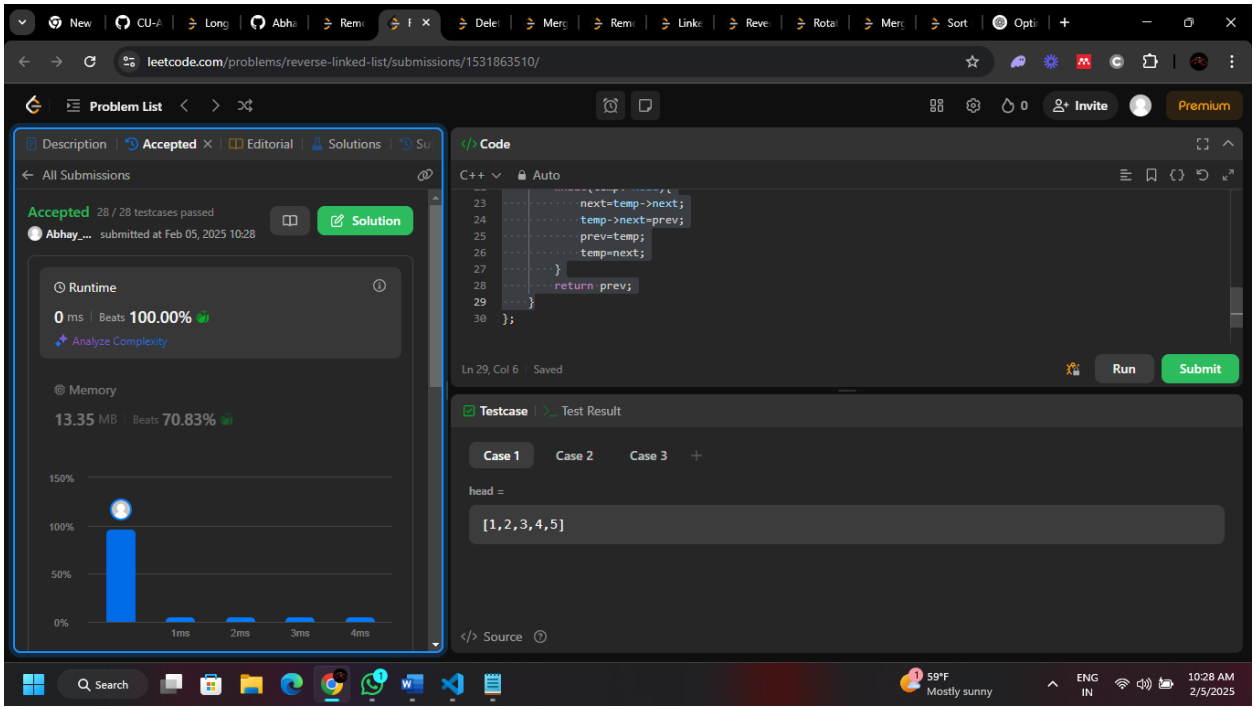
1. Remove Duplicates From A Sorted List-

```
ListNode* deleteDuplicates(ListNode* head) {
    ListNode* node=head;
    while(node!=NULL && node->next!=NULL){
      if(node->val==node->next->val){
        ListNode* temp=node->next;
        node->next=node->next->next;
      }
      else{
        node=node->next;
      }
    }
    return head;
  }
```



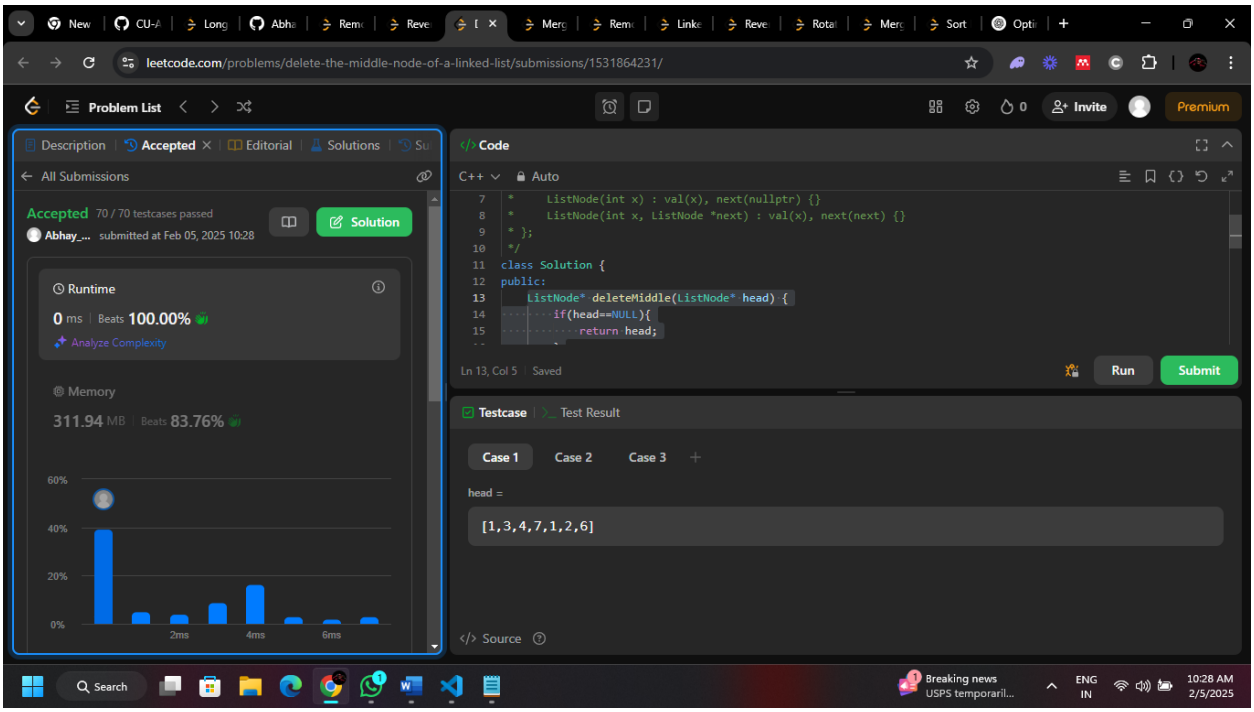
2. Reverse A Linked List –

```
ListNode* reverseList(ListNode* head) {
    if(head==NULL || head->next==NULL){
        return head;
    }

    ListNode* prev=NULL;
    ListNode* temp=head;
    ListNode* next=NULL;

    while(temp!=NULL){
        next=temp->next;
        temp->next=prev;
        prev=temp;
        temp=next;
    }
    return prev;
}
```



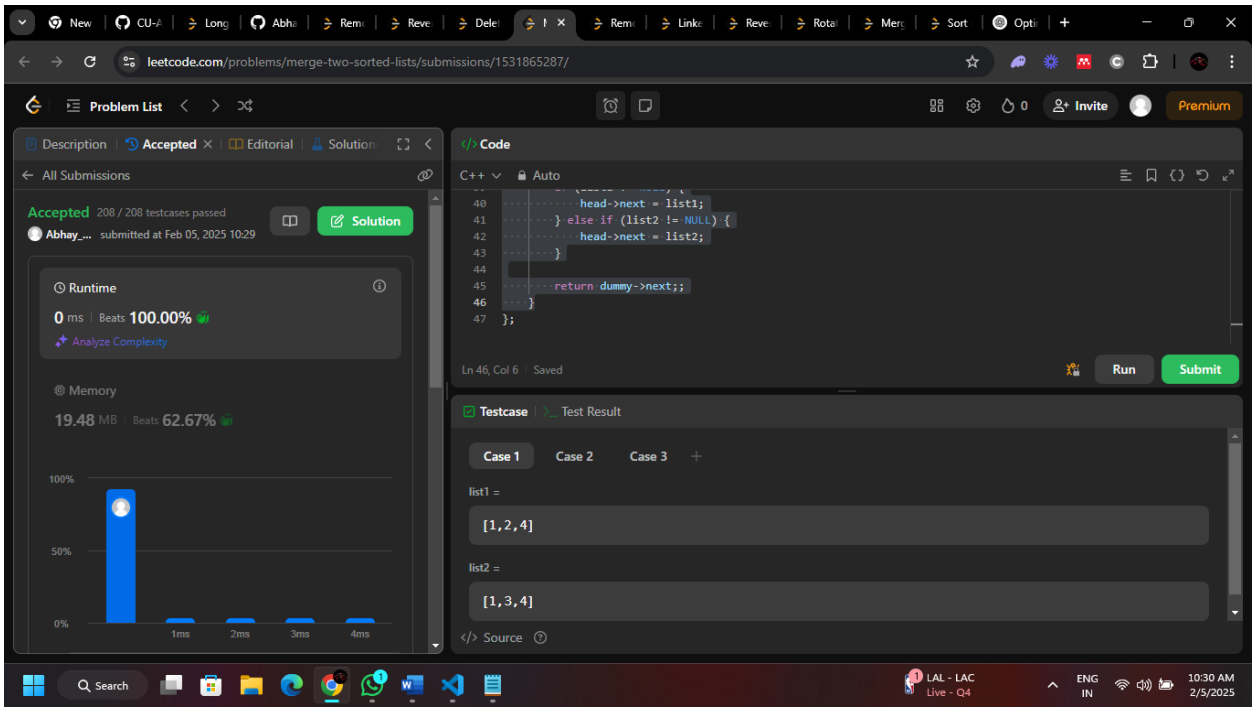
3. Delete Middle Node Of A List-

```
ListNode* deleteMiddle(ListNode* head) {
    if(head==NULL){
        return head;
    }
    if(head->next==NULL){
        head=head->next;
        return head;
    }
    ListNode* fast=head;
    ListNode* slow=head;
    ListNode* prev=NULL;
    while(fast!=NULL && fast->next!=NULL){
        fast=fast->next->next;
        prev=slow;
        slow=slow->next;
    }
    prev->next=slow->next;
    slow=slow->next;
    return head;
}
```



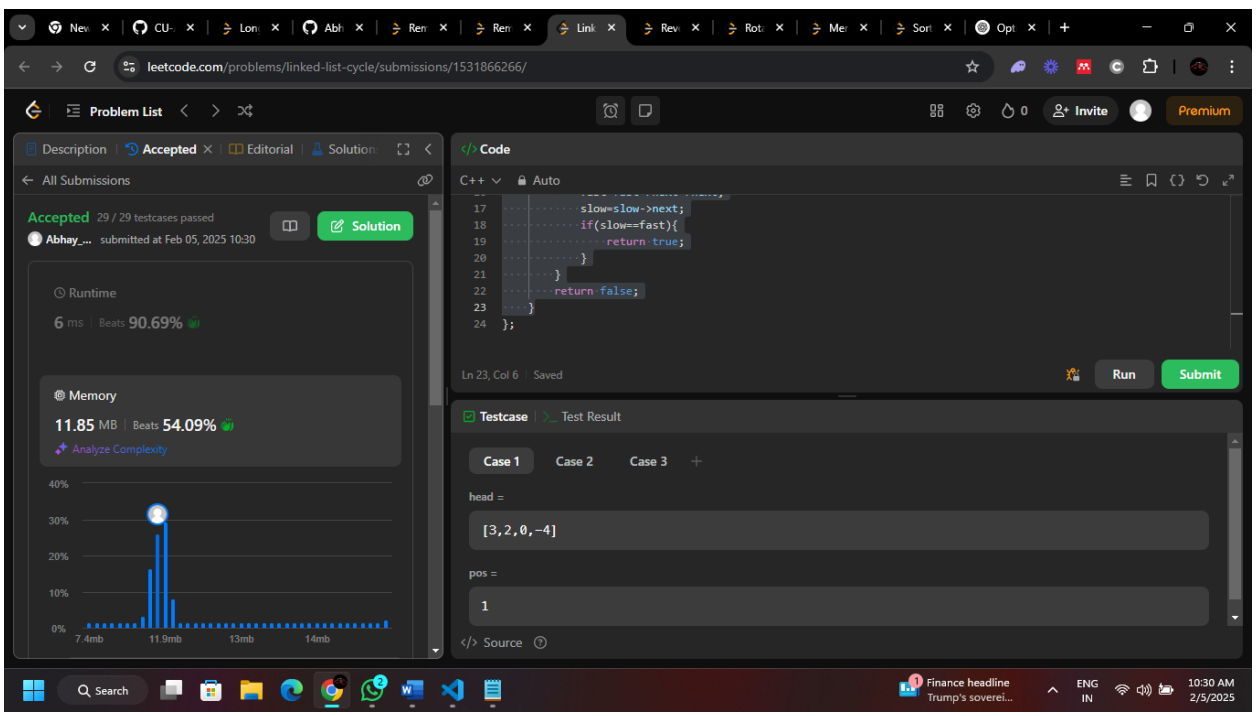
4. Merge Two Sorted Liked List-

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    if (list1 == NULL && list2 == NULL) {
        return NULL;
    }

    if (list1 == NULL) {
        return list2;
    }
    if (list2 == NULL) {
        return list1;
    }

    ListNode* dummy = new ListNode(-1);
    ListNode* head = dummy;

    while (list1 != NULL && list2 != NULL) {
        if (list1->val <= list2->val) {
            head->next = list1;
            list1 = list1->next;
        } else {
            head->next = list2;
            list2 = list2->next;
        }
        head = head->next;
    }

    if (list1 != NULL) {
        head->next = list1;
    } else if (list2 != NULL) {
        head->next = list2;
    }

    return dummy->next;;
}
```


leetcode.com/problems/merge-two-sorted-lists/submissions/1531865287/
Problem List
Description
Accepted
Editorial
Solution
All Submissions
Accepted 208 / 208 testcases passed
Abhay_... submitted at Feb 05, 2025 10:29
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
19.48 MB | Beats 62.67%
100%
50%
0%
1ms
2ms
3ms
4ms
Code
C++
Auto
head->next = list1;
} else if (list2 != NULL) {
head->next = list2;
}
return dummy->next;;
}
};
Ln 46, Col 6
Saved
Run
Submit
Testcase
Test Result
Case 1
Case 2
Case 3
list1 =
[1,2,4]
list2 =
[1,3,4]
Source

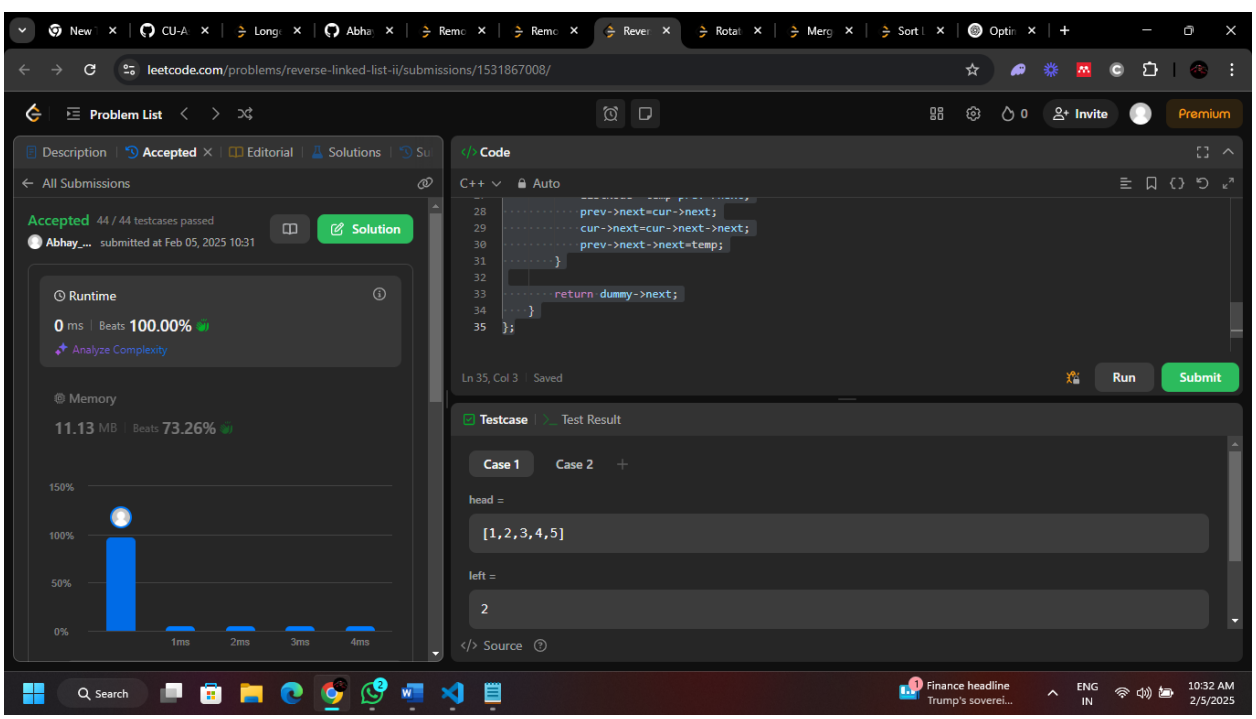
5. Detect A Cycle In A Linked List-

```
bool hasCycle(ListNode *head) {
    ListNode* fast=head;
    ListNode* slow=head;

    while(fast!=NULL && fast->next!=NULL){
        fast=fast->next->next;
        slow=slow->next;
        if(slow==fast){
            return true;
        }
    }
    return false;
}
```



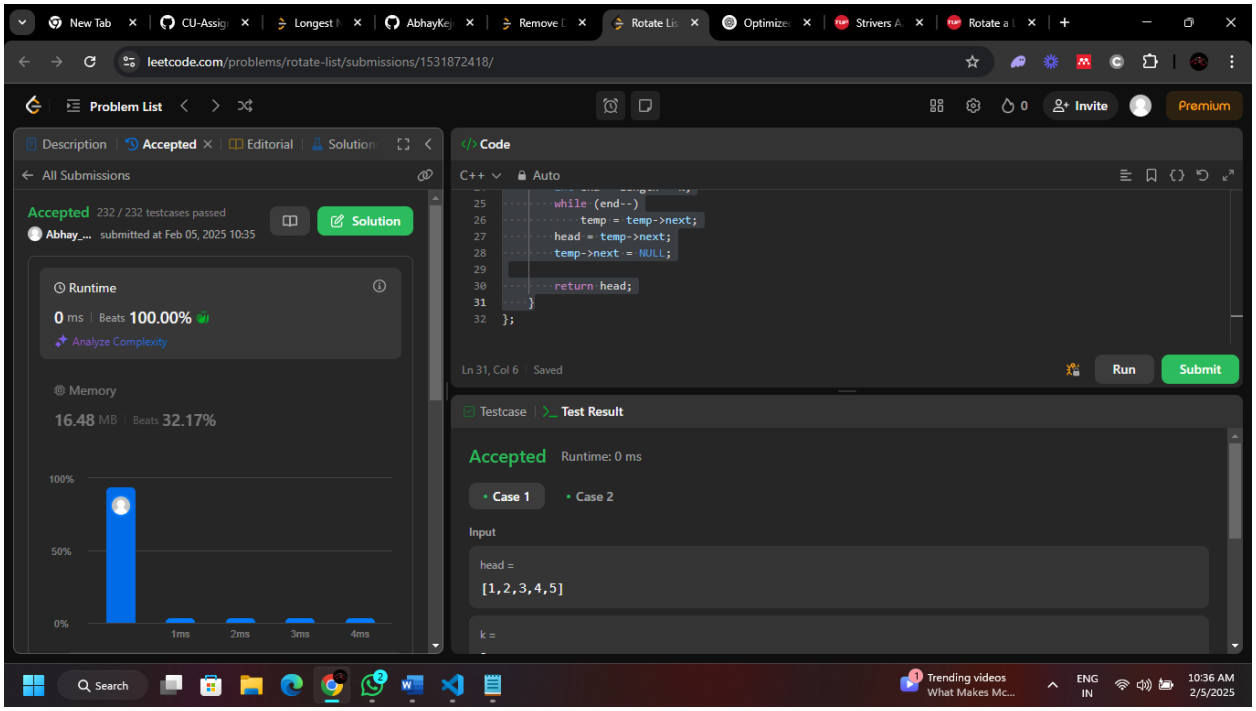
6. Reverse Linked List 2-

```
ListNode* reverseBetween(ListNode* head, int left, int right) {
    if(head==NULL || head->next==NULL){
        return head;
    }
    ListNode* dummy=new ListNode(-1);
    dummy->next=head;
    ListNode* prev=dummy;
    for(int i=1;i<left;i++){
        prev=prev->next;
    }
    ListNode* cur=prev->next;
    for(int i=0;i<right-left;i++){
        ListNode* temp=prev->next;
        prev->next=cur->next;
        cur->next=cur->next->next;
        prev->next->next=temp;
    }
    return dummy->next;
}
```



7. Rotate A List-

```
ListNode* rotateRight(ListNode* head, int k) {
    if (head == NULL || head->next == NULL || k == 0)
        return head;
    ListNode* temp = head;
    int length = 1;
    while (temp->next != NULL) {
        ++length;
        temp = temp->next;
    }
    temp->next = head;
    k = k % length;
    int end = length - k;
    while (end--)
        temp = temp->next;
    head = temp->next;
    temp->next = NULL;

    return head;
}
```



8. Sort List –

```
ListNode* findMiddle(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* mergeTwoList(ListNode* left, ListNode* right){
    ListNode* dummy=new ListNode(-1);
    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;
        }
    }

    if(left)temp->next=left;
    else temp->next=right;

    return dummy->next;
}

ListNode* sortList(ListNode* head) {
  if (head==NULL || head->next==NULL)return head;
```

```
    ListNode* middle=findMiddle(head);
    ListNode* left=head;
    ListNode* right=middle->next;
    middle->next=NULL;

    left=sortList(left);
    right=sortList(right);
    return mergeTwoList(left,right);
}
```