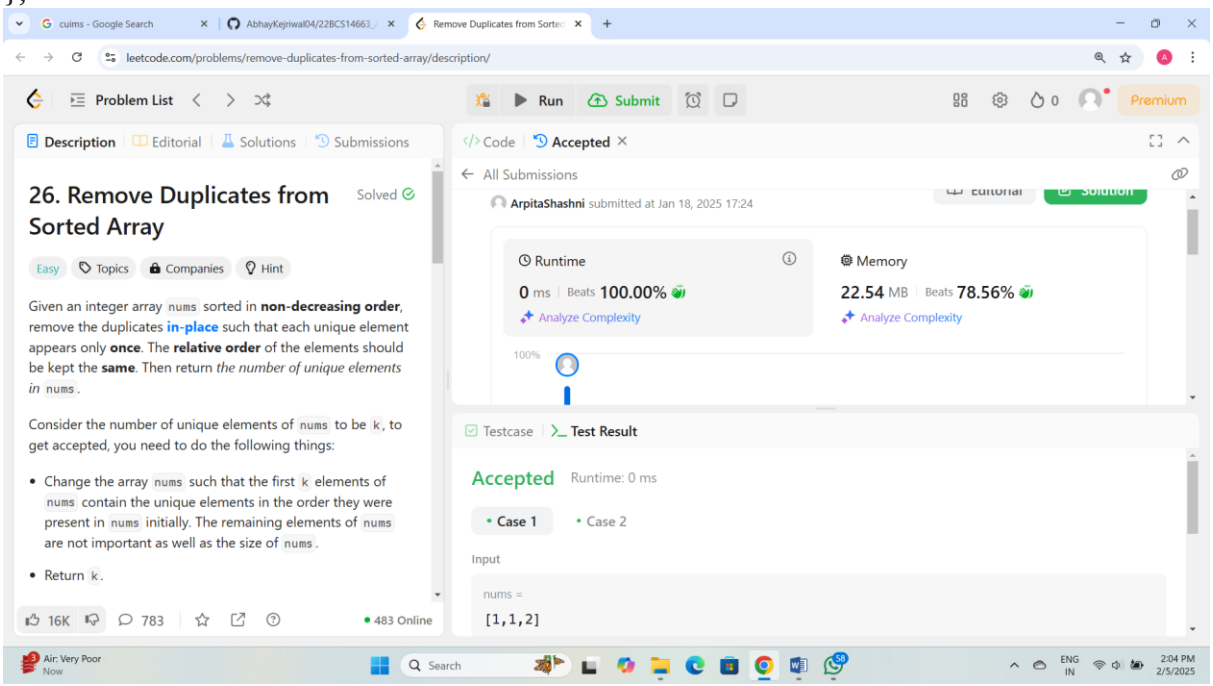
Name: Arpita
UID: 22BCS15627
Section: 601-B
Subject: Advanced Programming-II
Subject Code: 22CSP-351

DAY 1

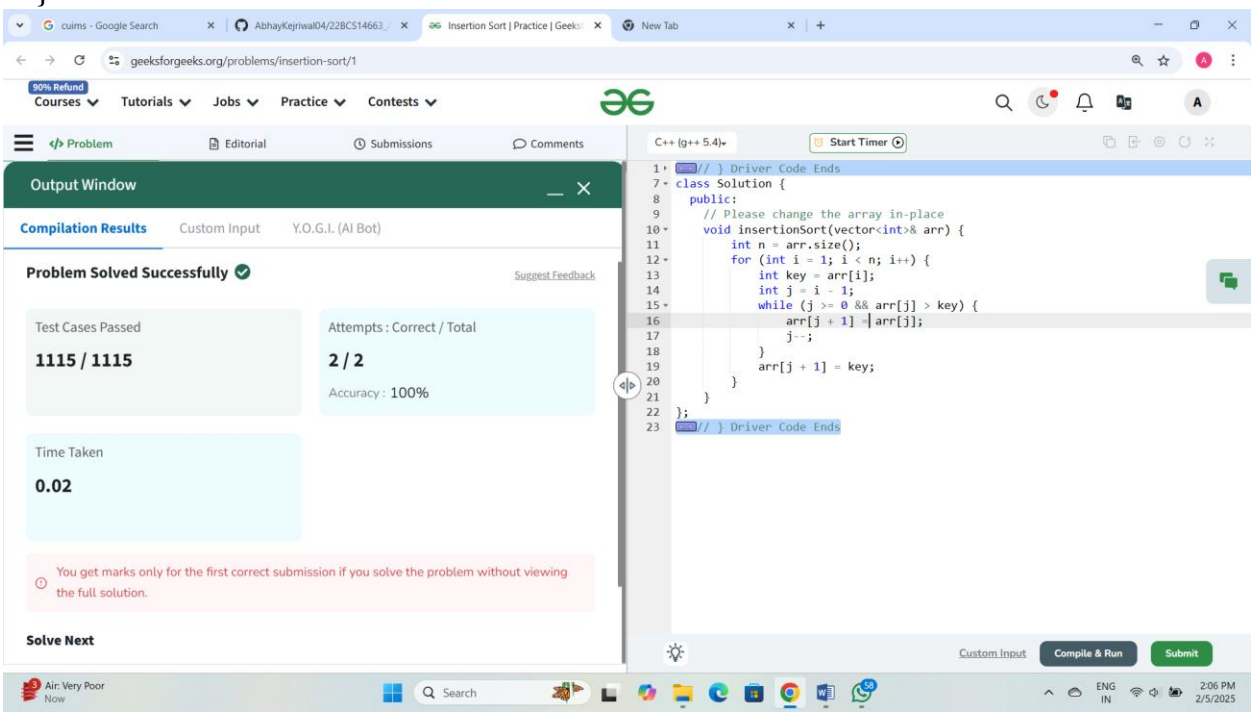
Problem 1: Remove duplicates from sorted array

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



Problem 2: Implementing insertion sort

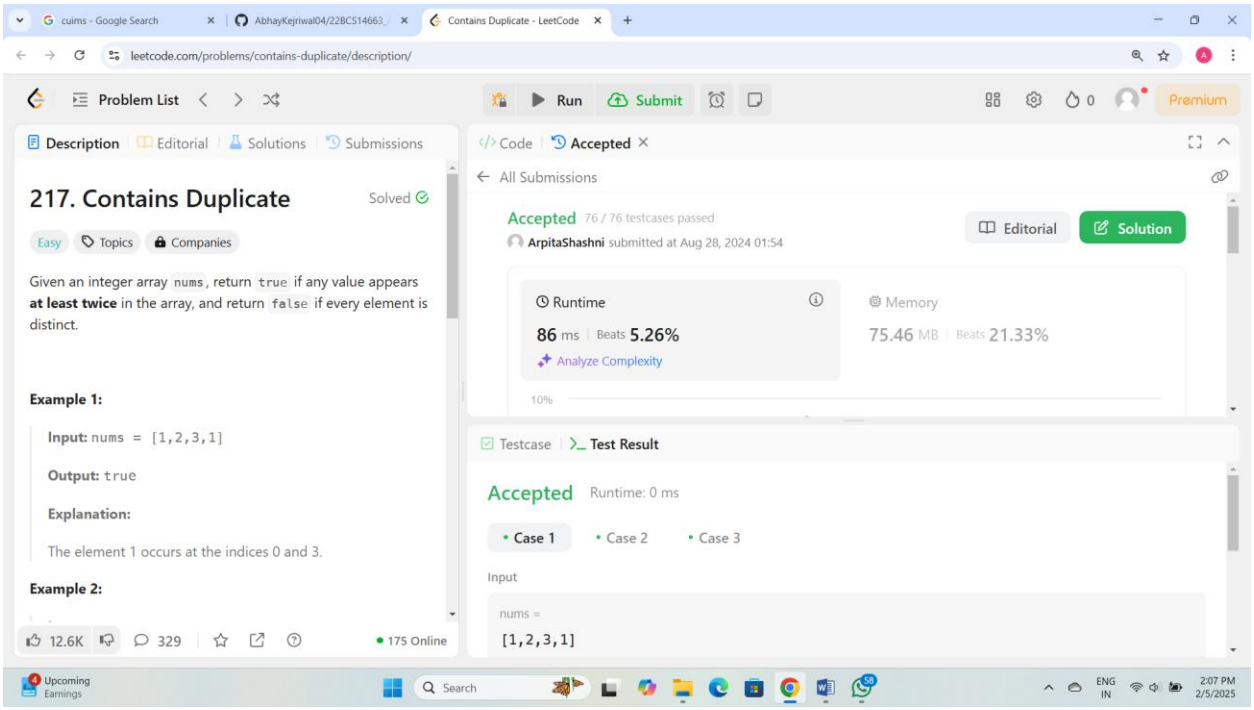
```
public:
    // Please change the array in-place
    void insertionSort(vector<int>& arr) {
        int n = arr.size();
        for (int i = 1; i < n; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
```



Problem 3: Contains Duplicate

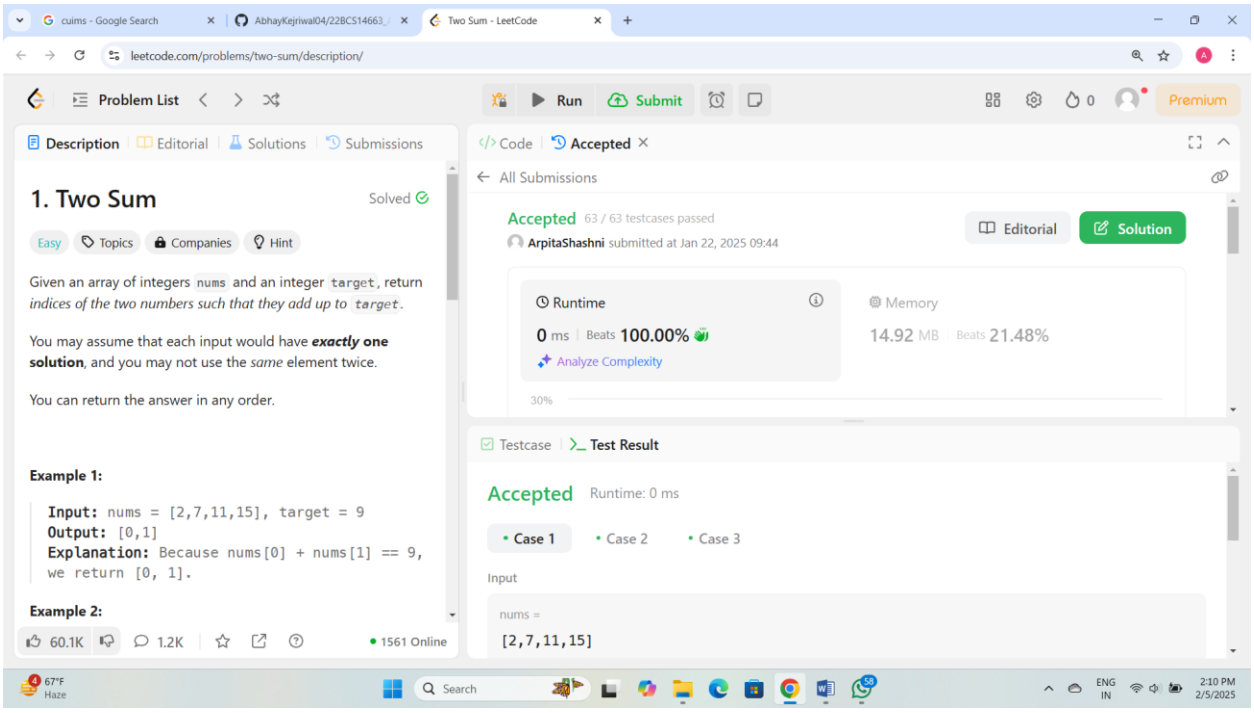
```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_map<int,int> m;
        for(int i=0;i<nums.size();i++){
            m[nums[i]]++;
        }
        for(auto i:m){
            if(i.second>1){
```

```
            return true;
        }
    }
    return false;
  }
};
```



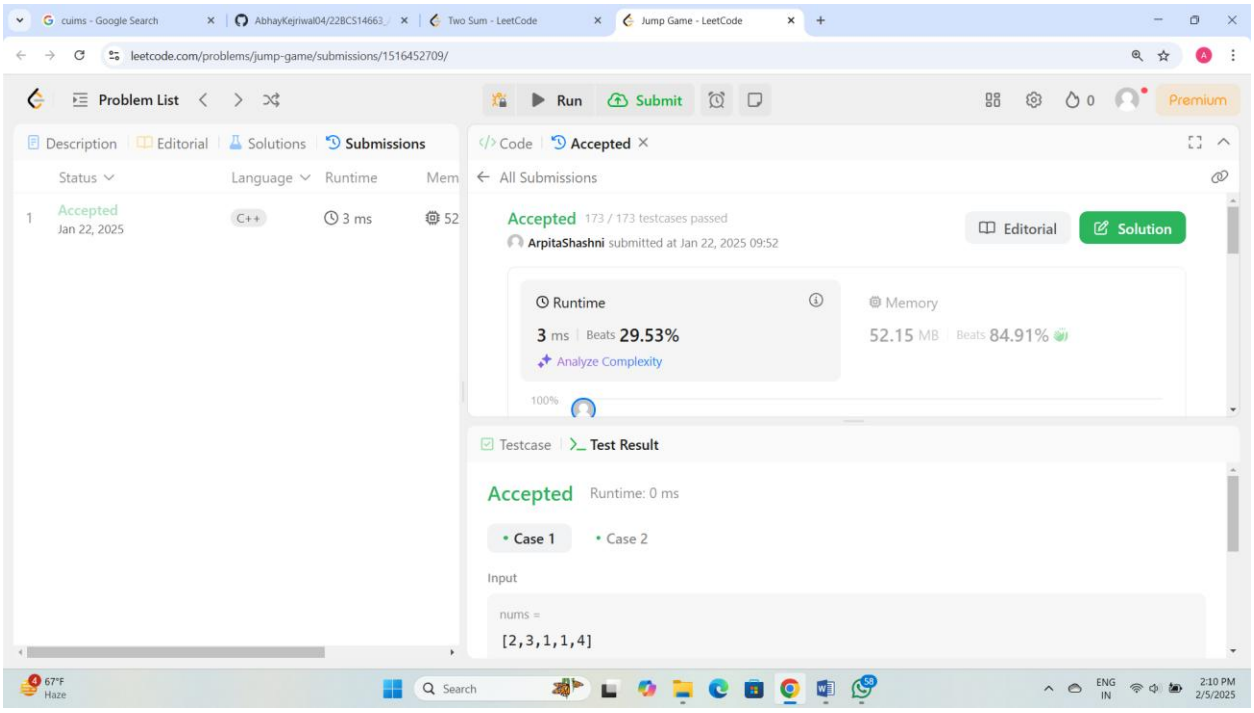
Problem 4: Two Sum

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> m;
        for(int i=0;i<nums.size();i++){
            int a=target-nums[i];
            if(m.find(a)!=m.end()){
                return {m[a],i};
            }
            m[nums[i]]=i;
        }
        return {};
    }
};
```



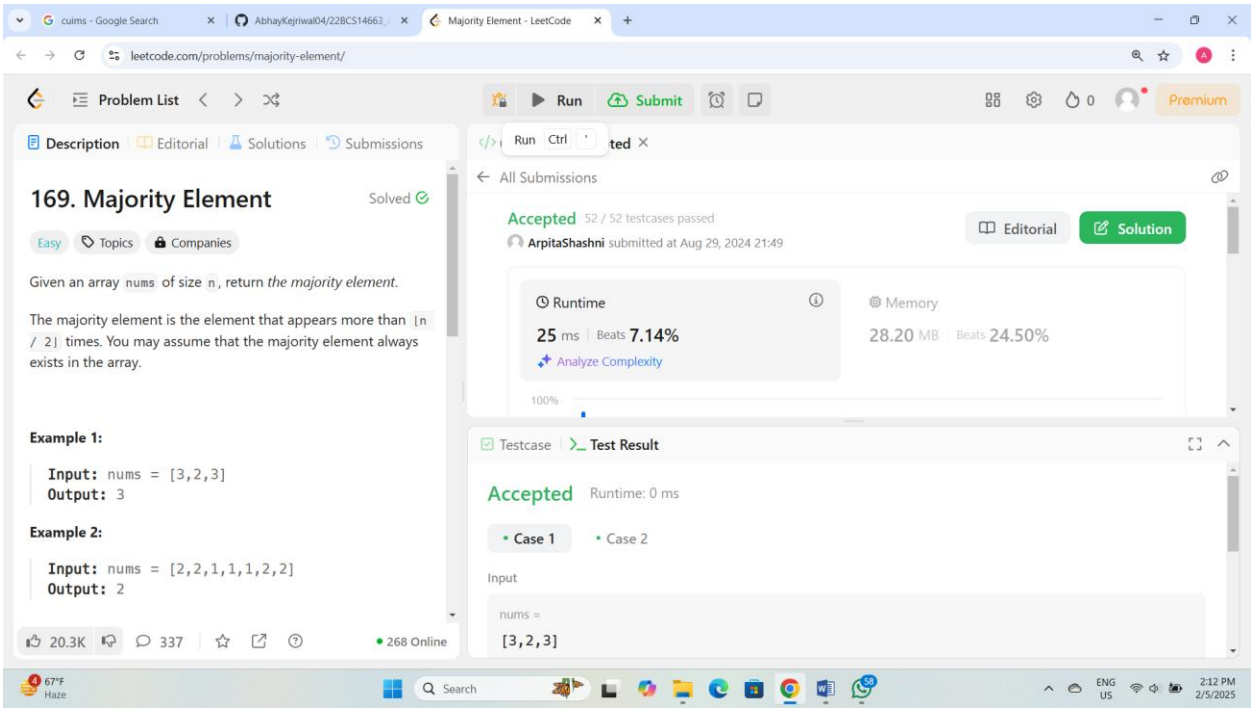
Problem 5: Jump Game

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int n=nums.size();
        int x=nums[0];
        for(int i=0;i<n;i++){
            if(i>x){
                return false;
            }
            x=max(x,nums[i]+i);
        }
        return true;
    }
};
```



Problem 6: Majority element

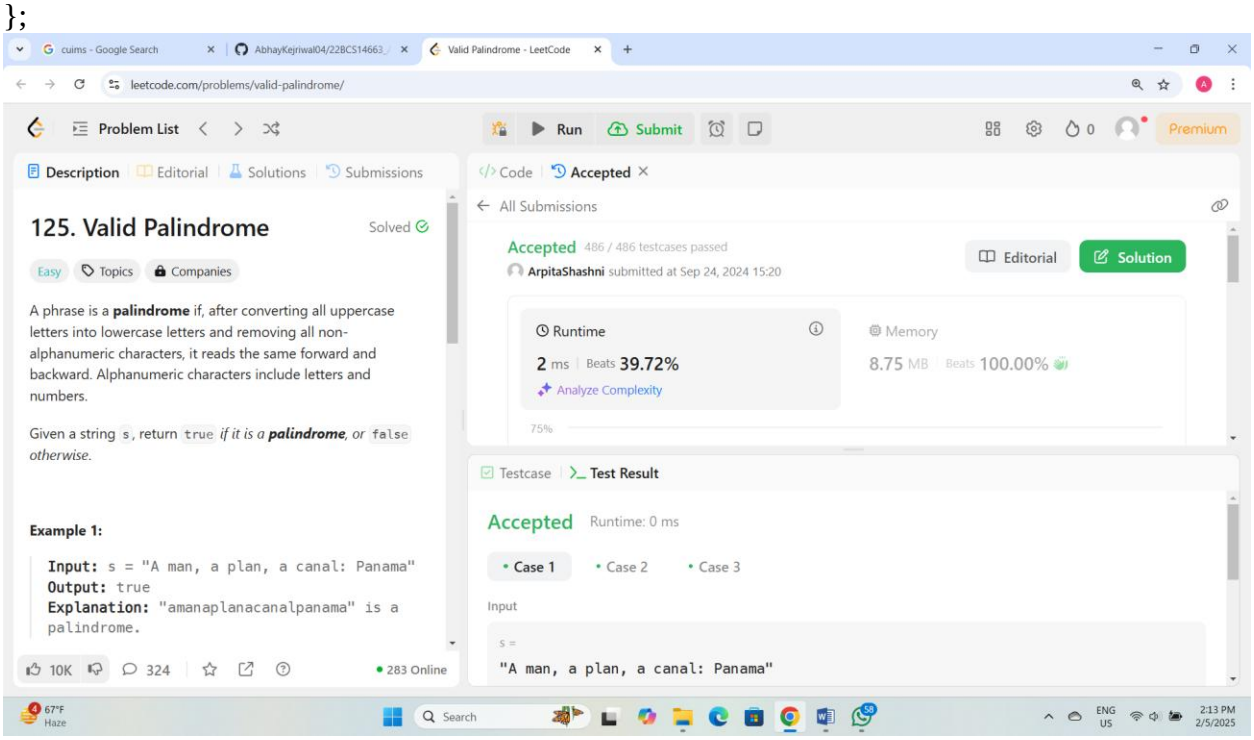
```
#include<unordered_map>
class Solution {
public:
   int majorityElement(vector<int>& nums) {
      int n=nums.size();
      unordered_map<int,int> m;
      for(int i=0;i<n;i++){
         m[nums[i]]++;
      }
      for(int i=0;i<n;i++){
         if(m[nums[i]]>n/2){
            return nums[i];
         }
      }
      return -1;
   }
};
```



Problem 7: Valid Palindrome

```
class Solution {
public:
    bool isal(char &c){
        if(c>='0' && c<='9'){
            return true;
        }
        if(c>='A' && c<='Z'){
            c=c-'A'+'a';
            return true;
        }
        if(c>='a' && c<='z'){
            return true;
        }
        return false;
    }
    bool isPalindrome(string s) {
        if(s==""){
            return true;
        }
        //s=toLowerCase(s);
        int n = s.length();
        int i = 0;
        int j = n;
        while(i<n && j>=0)
        {
```

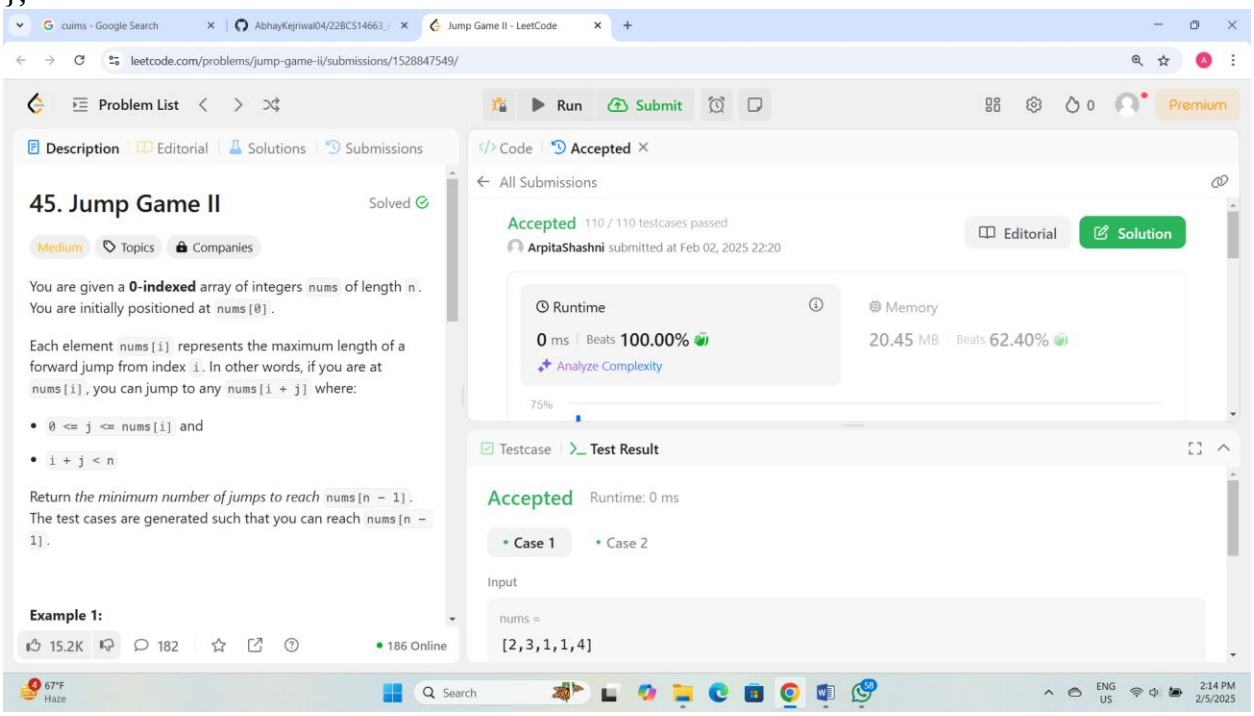
```
            if (isal(s[i]) == 0)
            {
                i++;
            }
            else if (isal(s[j]) == 0)
            {
                j--;
            }
            else if (s[i] == s[j])
            {
                ++i;
                j--;
            }
            else
            {
                return false;
            }
        }
        return true;
    }
};
```



Problem 8: Jump Game 2

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int n = nums.size();
```

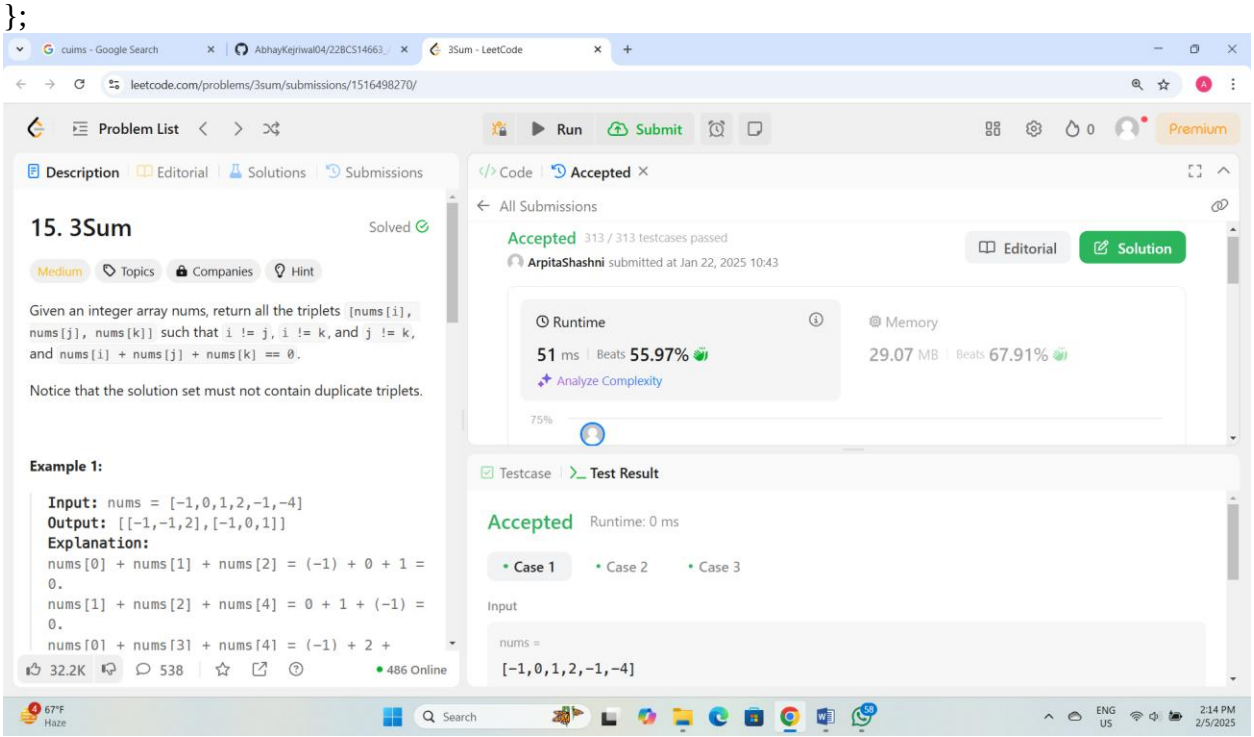
```
        if (n == 1) return 0;
        int jumps = 0, farthest = 0, currentEnd = 0;
        for (int i = 0; i < n - 1; i++) {
            farthest = max(farthest, i + nums[i]);
            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;
                if (currentEnd >= n - 1) break;
            }
        }
        return jumps;
    }
};
```



Problem 9: 3Sum

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        vector<vector<int>> ans;
        int n=nums.size();
        for(int i=0;i<n-2;i++){
            if(i>0 && nums[i]==nums[i-1]){
                continue;
            }
            int start=i+1,end=n-1;
            while(start<end){
```

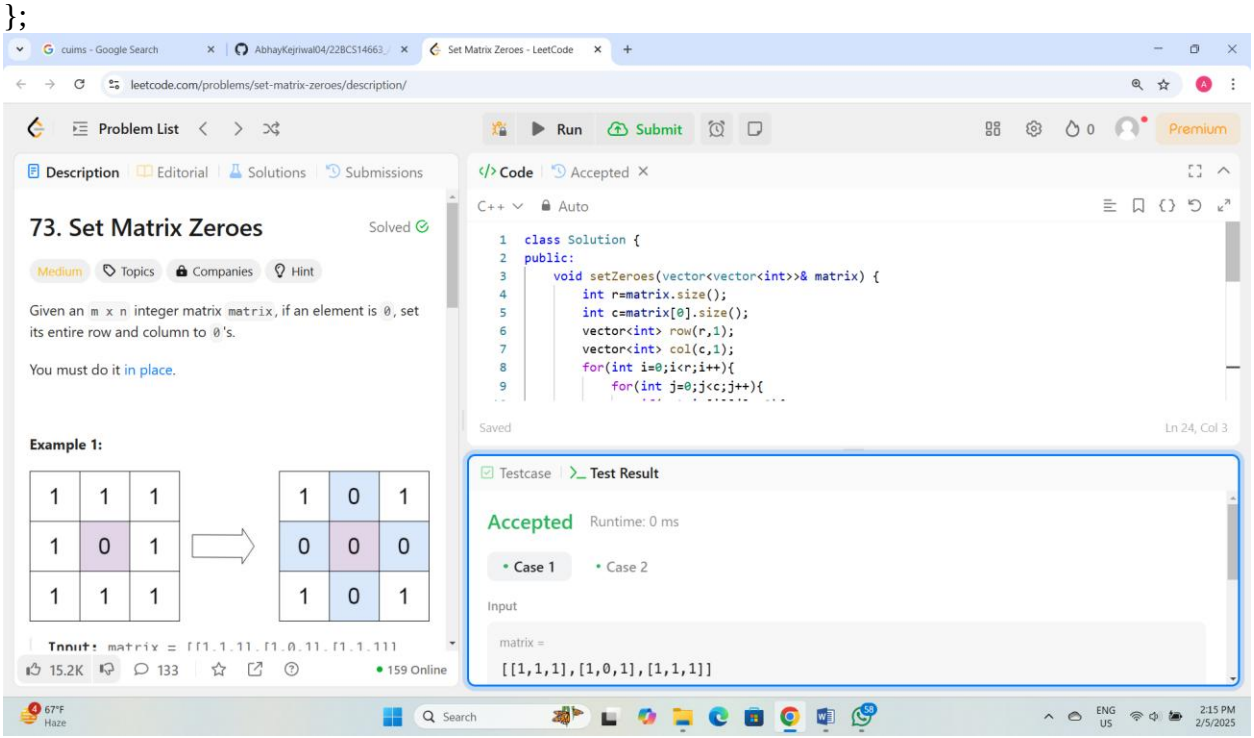
```
            int sum=nums[i]+nums[start]+nums[end];
            if(sum==0){
                ans.push_back({nums[i],nums[start],nums[end]});
                while(start<end && nums[start]==nums[start+1]){
                    start++;
                }
                while(start<end && nums[end]==nums[end-1]){
                    end--;
                }
                start++;
                end--;
            }
            else if(sum>0){
                end--;
            }
            else{
                start++;
            }
        }
    }
    return ans;
  }
};
```



Problem 10: Set Matrix Zeroes

class Solution {
public:

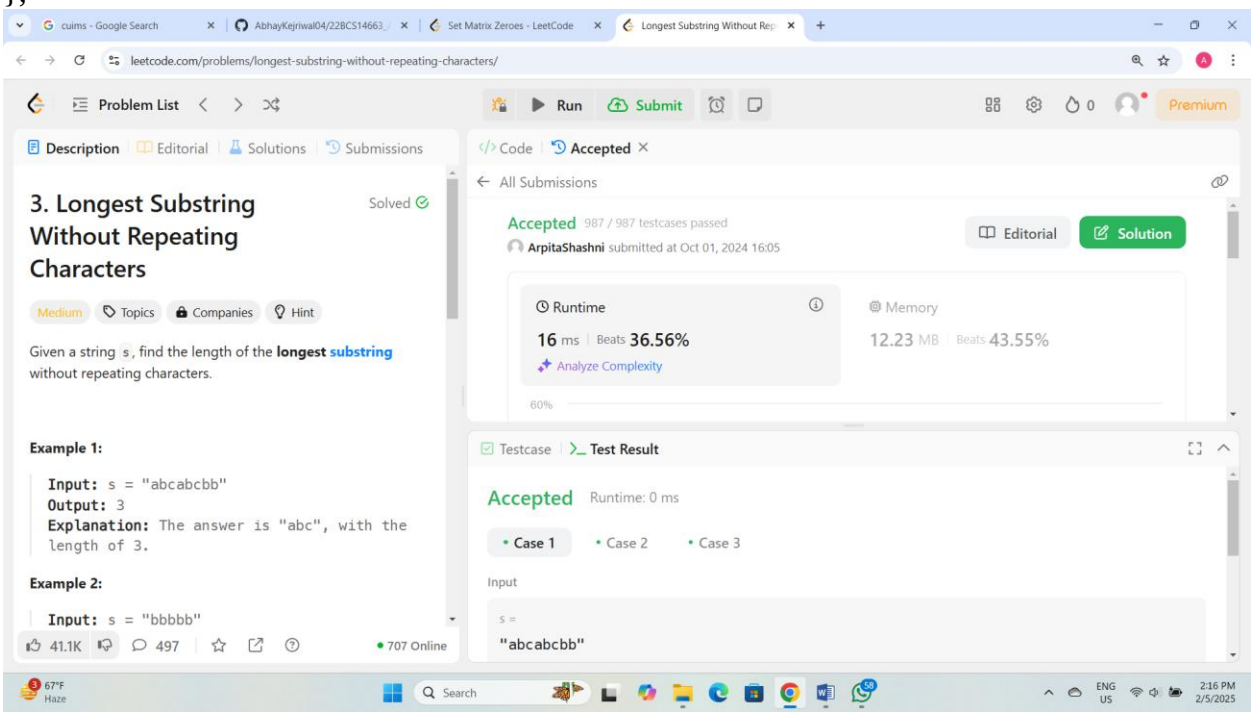
```
    void setZeroes(vector<vector<int>>& matrix) {
        int r=matrix.size();
        int c=matrix[0].size();
        vector<int> row(r,1);
        vector<int> col(c,1);
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                if(matrix[i][j]==0){
                    row[i]=0;
                    col[j]=0;
                }
            }
        }
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                if(row[i]==0||col[j]==0){
                    matrix[i][j]=0;
                }
            }
        }
    }
};
```



Problem 11: Longest substring without repeating characters

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
```

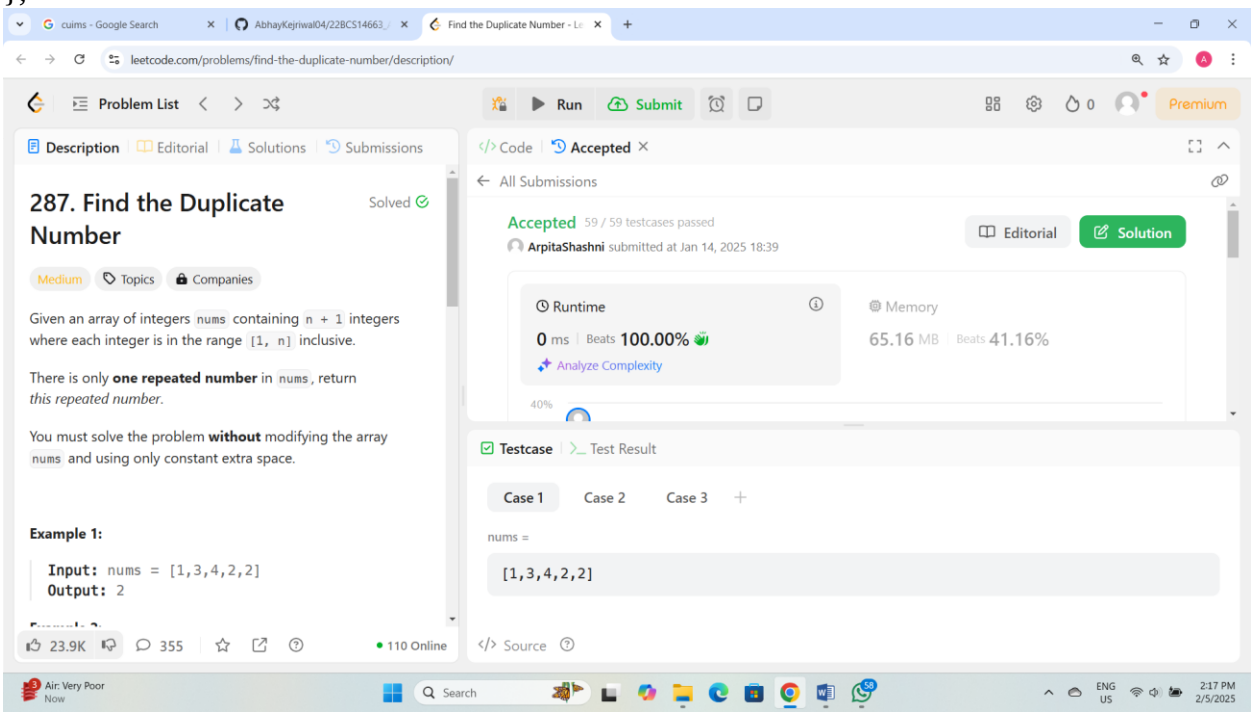
```
        unordered_map<char,int> m;
        int ans=0,left=0;
        for(int i=0;i<s.length();i++){
            char c=s[i];
            if(m.count(c) && m[c]>=left){
                left=m[c]+1;
            }
            m[c]=i;
            ans=max(ans,i-left+1);
        }
        return ans;
    }
};
```



Problem 12: Finding duplicate number

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int slow=nums[0];
        int fast=nums[0];
        do{
            slow=nums[slow];
            fast=nums[nums[fast]];
        }while(slow!=fast);
        slow=nums[0];
        while(slow!=fast){
            slow=nums[slow];
```

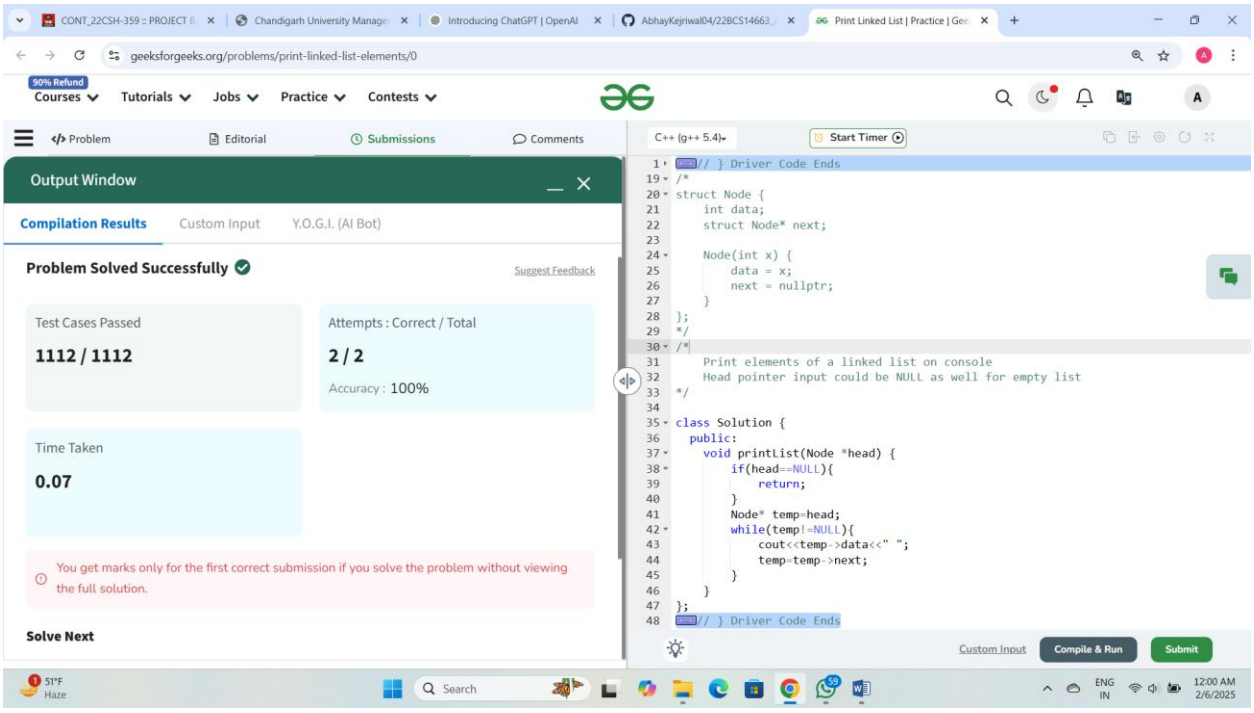
```
        fast=nums[fast];
    }
    return fast;
  }
};
```



DAY 2

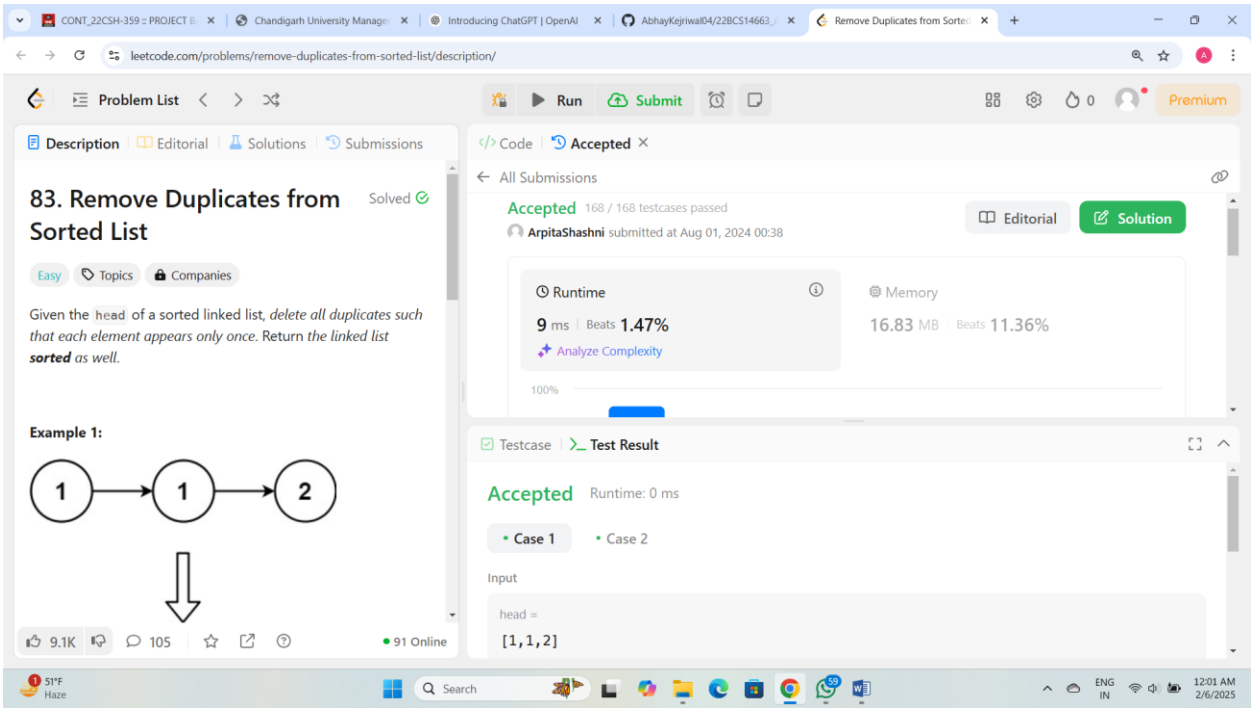
Problem 1: Print linked list

```
class Solution {
  public:
    void printList(Node *head) {
        if(head==NULL){
            return;
        }
        Node* temp=head;
        while(temp!=NULL){
            cout<<temp->data<<" ";
            temp=temp->next;
        }
    }
};
```



Problem 2: Remove duplicates from a sorted array

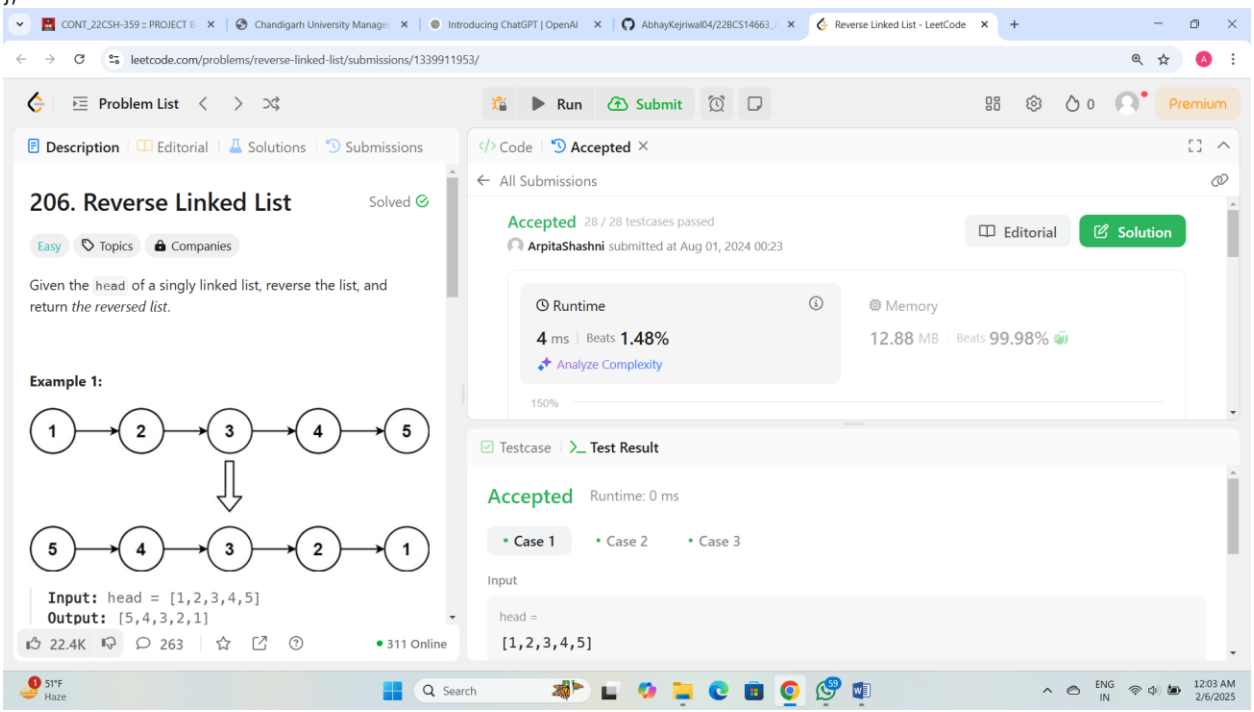
```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head==NULL){
            return NULL;
        }
        ListNode* temp=head;
        while(temp!=NULL && temp->next!=NULL){
            if((temp->val==temp->next->val)){
                ListNode* data=temp->next;
                temp->next=data->next;
                delete data;
            }
            else{
                temp=temp->next;
            }
        }
        return head;
    }
};
```



Problem 3: Reverse a linked list

```
/**
 * 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* reverseList(ListNode* head) {
        if(head==NULL || head->next==NULL){
            return head;
        }
        else{
            ListNode* temp1=NULL;
            ListNode* temp2=head;
            while(temp2!=NULL){
                ListNode* temp3=temp2->next;
                temp2->next=temp1;
                temp1=temp2;
                temp2=temp3;
            }
```

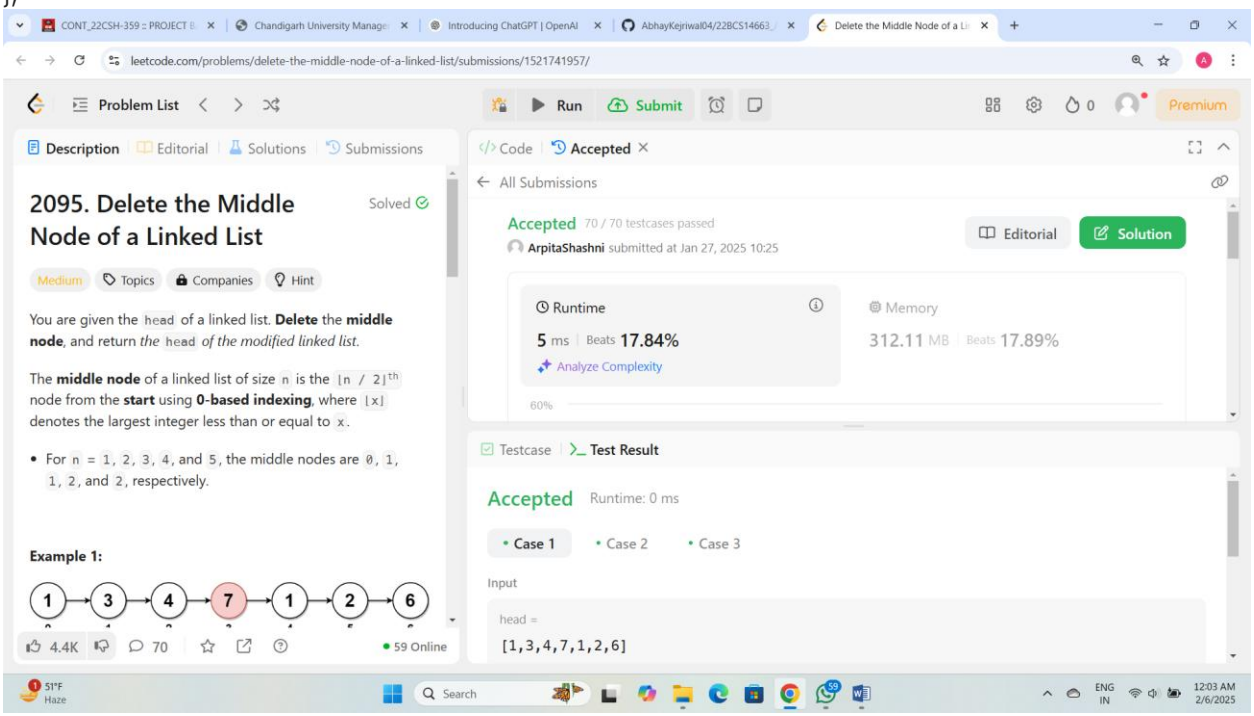
```
        return temp1;
    }
  }
};
```



Problem 4: Delete middle node of a list

```
/**
 * 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* deleteMiddle(ListNode* head) {
      if(head->next==NULL){
         return NULL;
      }
      ListNode* temp=head;
      int n=0;
      while(temp!=NULL){
         n++;
         temp=temp->next;
```

```
        }
        temp=head;
        int a=n/2;
        int x=0;
        while(temp!=NULL){
            x++;
            if(x==a){
                break;
            }
            temp=temp->next;
        }
        ListNode* c=temp->next;
        temp->next=c->next;
        delete c;
        return head;
    }
};
```

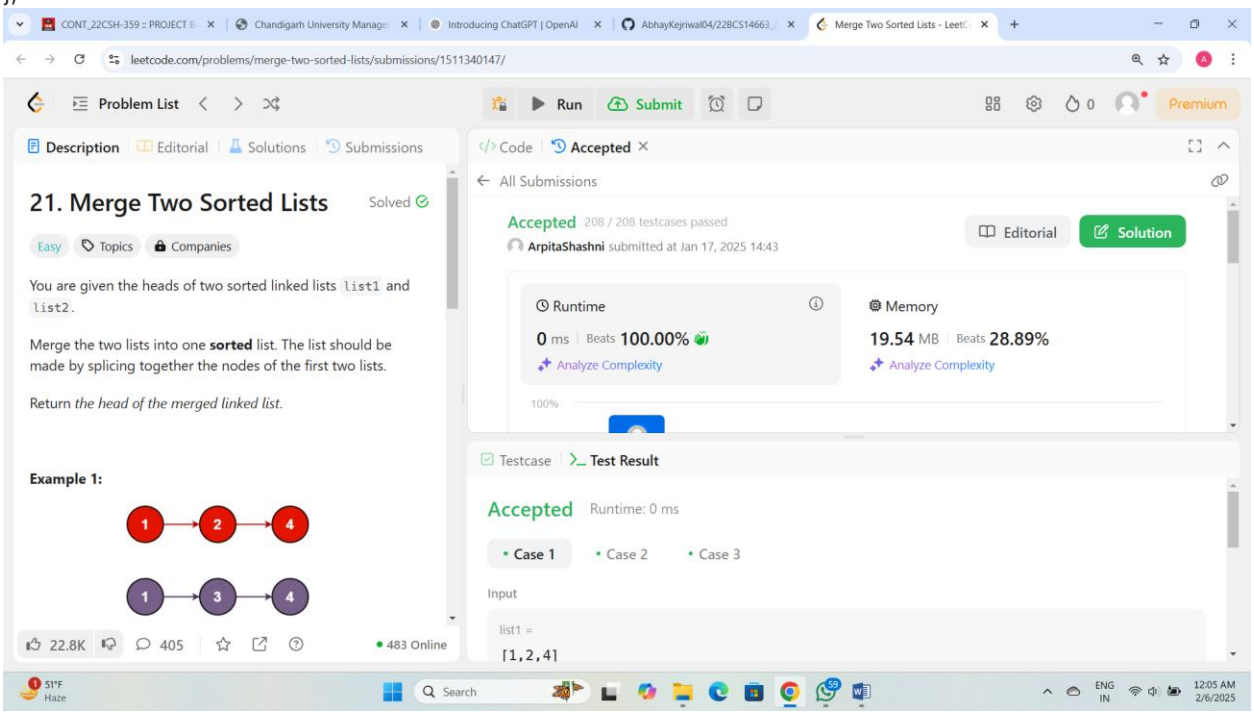


Problem 5: Merge two sorted linked lists

```
/**
 * 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* solve(ListNode* &list1,ListNode* &list2){
        if(list1->next==NULL){
            list1->next=list2;
            return list1;
        }
        ListNode* curr1=list1;
        ListNode* curr2=list2;
        ListNode* next1=list1->next;
        ListNode* next2=list2->next;
        while(next1!=NULL && curr2!=NULL){
            if(curr1->val<=curr2->val && curr2->val<=next1->val){
                curr1->next=curr2;
                next2=curr2->next;
                curr2->next=next1;
                curr1=curr2;
                curr2=next2;
            }
            else{
                curr1=next1;
                next1=next1->next;
                if(next1==NULL){
                    curr1->next=curr2;
                    return list1;
                }
            }
        }
        return list1;
    }
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if(list1==NULL){
            return list2;
        }
        if(list2==NULL){
            return list1;
        }
        if(list1->val<=list2->val){
            return solve(list1,list2);
        }
        else{
            return solve(list2,list1);
        }
    }
```

};



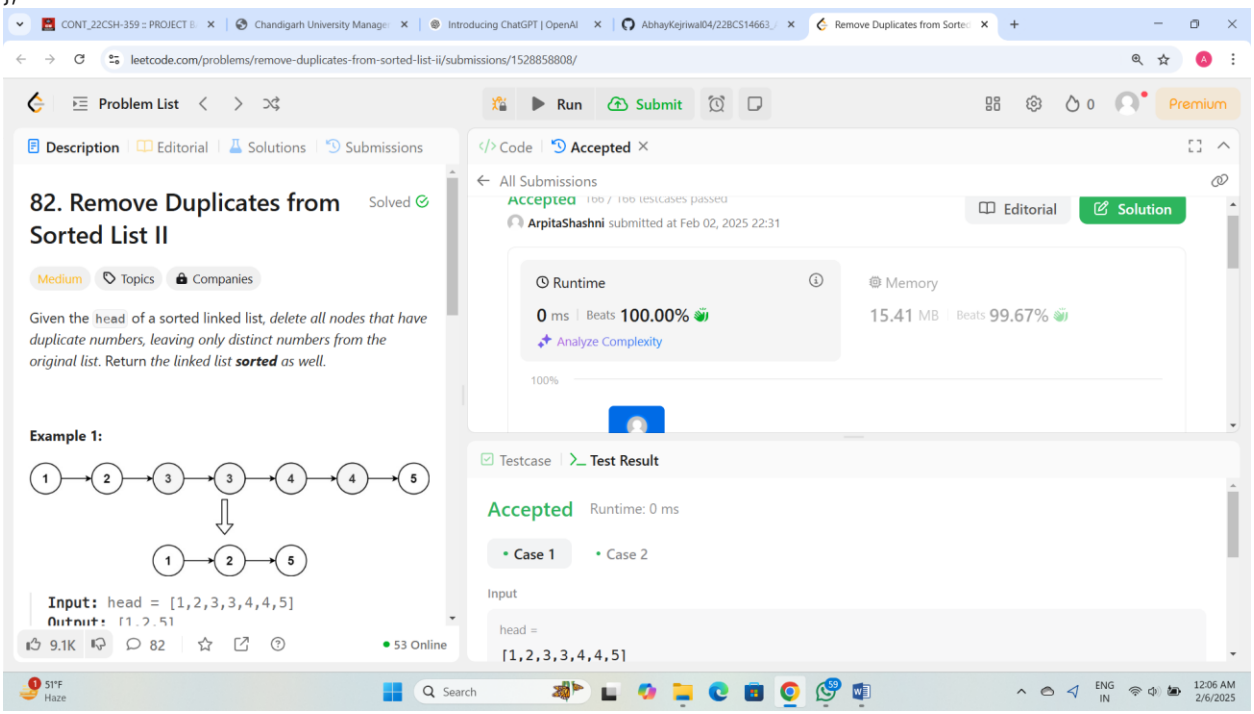
Problem 6: Remove duplicates from sorted lists 2

```
/**
 * 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* deleteDuplicates(ListNode* head) {
        if (!head || !head->next) return head;

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prev = dummy;

        while (head) {
            if (head->next && head->val == head->next->val) {
                while (head->next && head->val == head->next->val)
                    head = head->next;
```

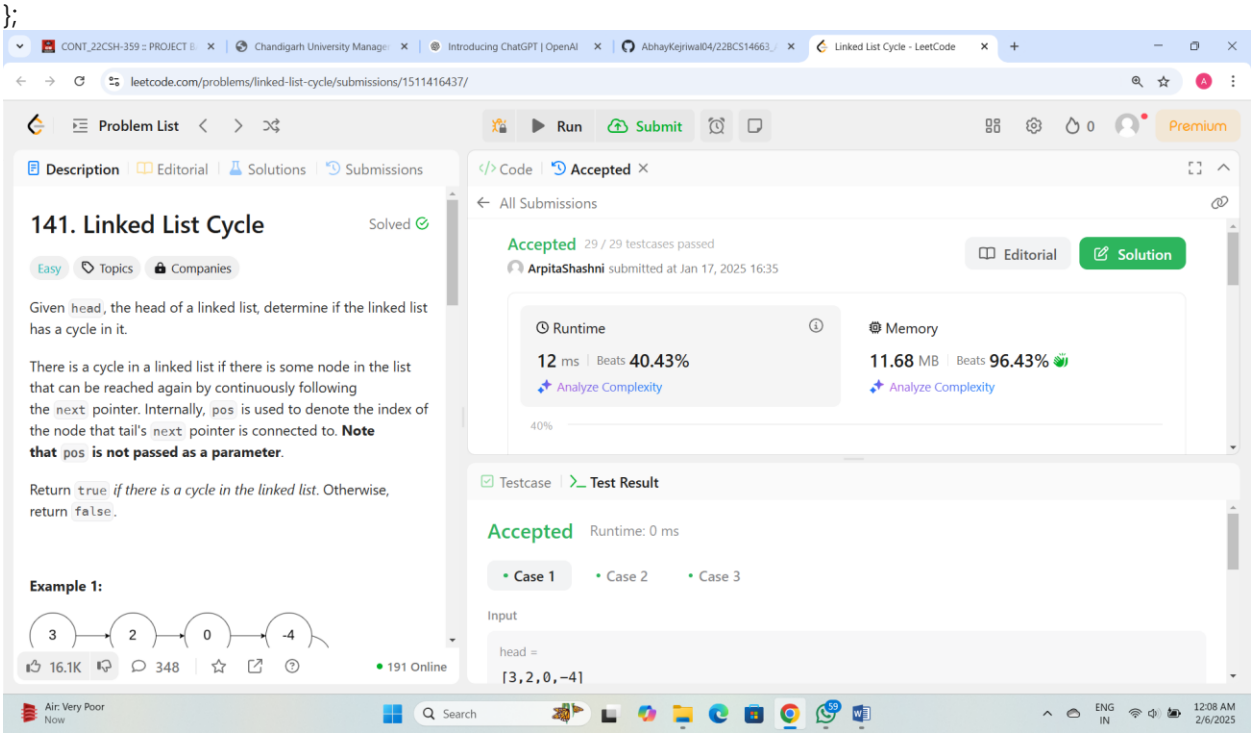
```
            prev->next = head->next;
        } else {
            prev = prev->next;
        }
        head = head->next;
    }
    return dummy->next;
  }
};
```



Problem 7: Detect a cycle in a linked list

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode* slow=head;
        ListNode* fast=head;
        if(head==NULL){
            return false;
        }
```

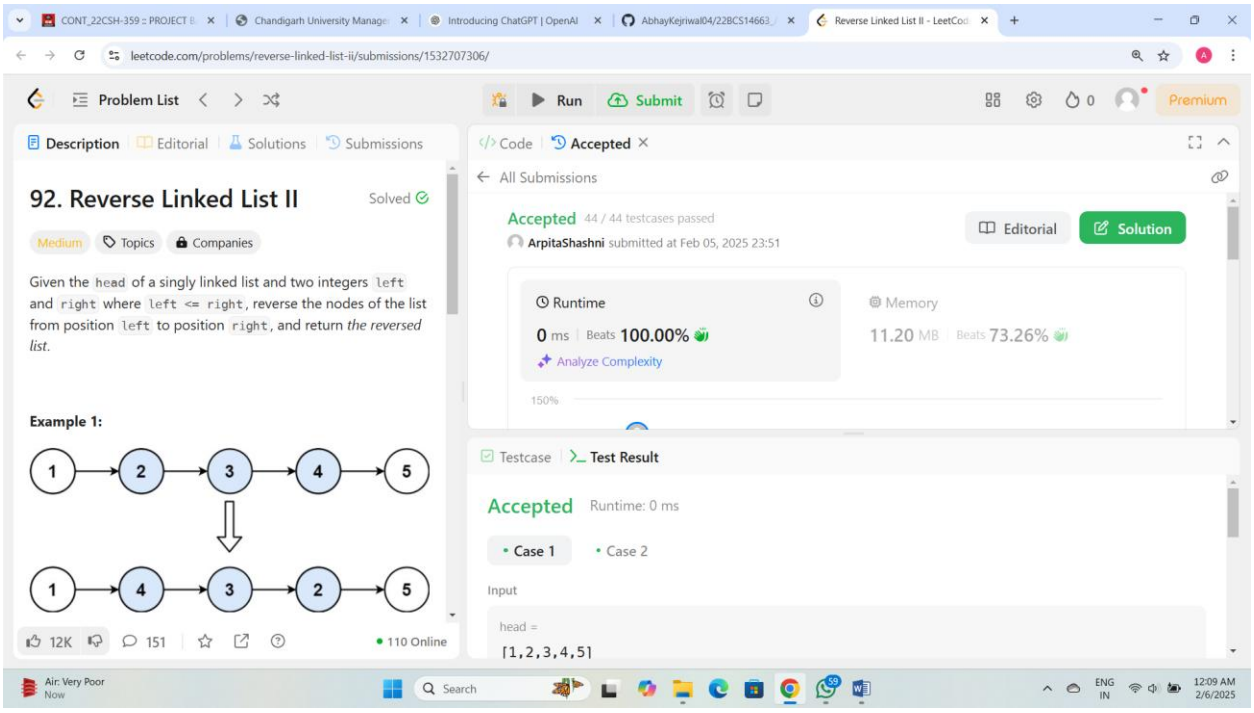
```
        if(head->next==NULL){
            return false;
        }
        while(slow!=NULL && fast!=NULL){
            slow=slow->next;
            fast=fast->next;
            if(fast!=NULL){
                fast=fast->next;
            }
            if(slow==fast){
                return true;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



Problem 8: Reverse linked list 2

```
/**
 * 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* reverse(ListNode* head){
        ListNode* curr = head;
        ListNode* prev = NULL;
        ListNode* next = NULL;
        while (curr != NULL) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        ListNode *revs = NULL, *revs_prev = NULL;
        ListNode *revend = NULL, *revend_next = NULL;
        int i = 1;
        ListNode* curr = head;
        while (curr && i <= right) {
            if (i < left)
                revs_prev = curr;
            if (i == left)
                revs = curr;
            if (i == right) {
                revend = curr;
                revend_next = curr->next;
            }
            curr = curr->next;
            i++;
        }
        revend->next = NULL;
        revend = reverse(revs);
        if (revs_prev)
            revs_prev->next = revend;
        else
            head = revend;
        revs->next = revend_next;
        return head;
    }
};
```

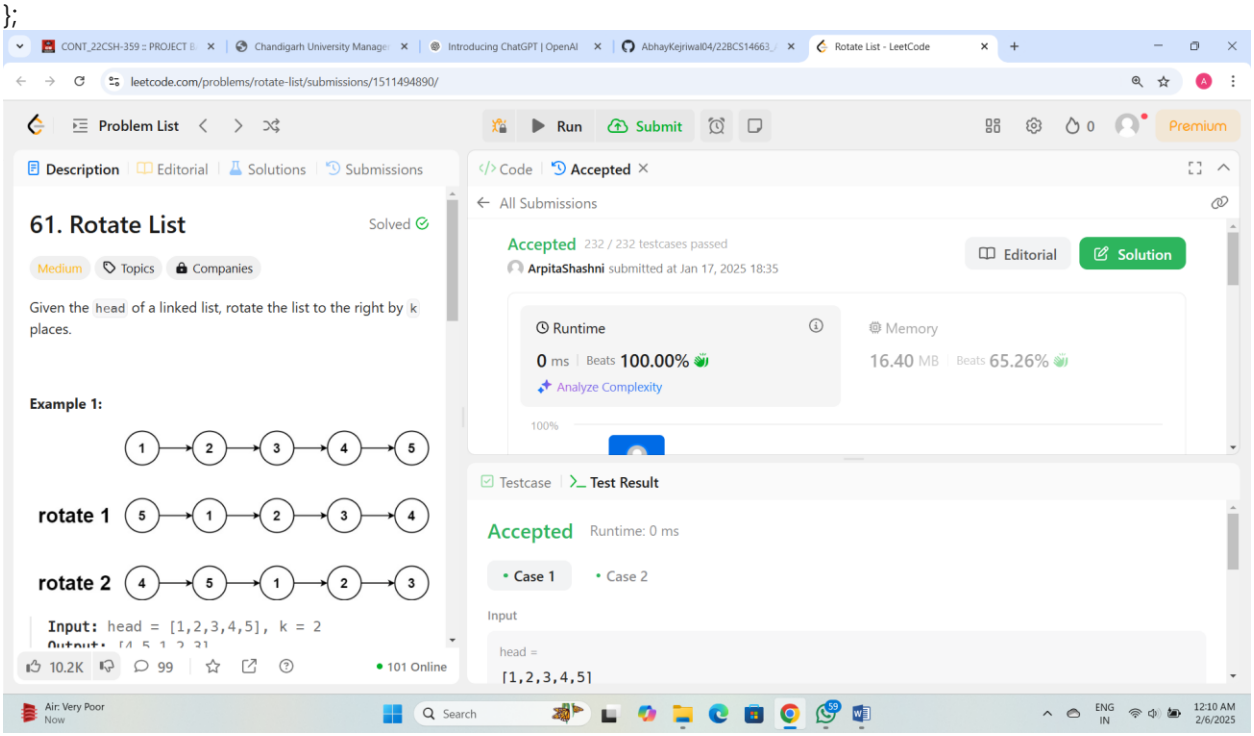


Problem 9: Rotate a list

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

```
        }
        ListNode* tail = head;
        for (int i = 1; i < len - k; i++) {
            tail = tail->next;
        }
        ListNode* newHead = tail->next;
        tail->next = NULL;
        temp = newHead;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = head;

        return newHead;
    }
};
```

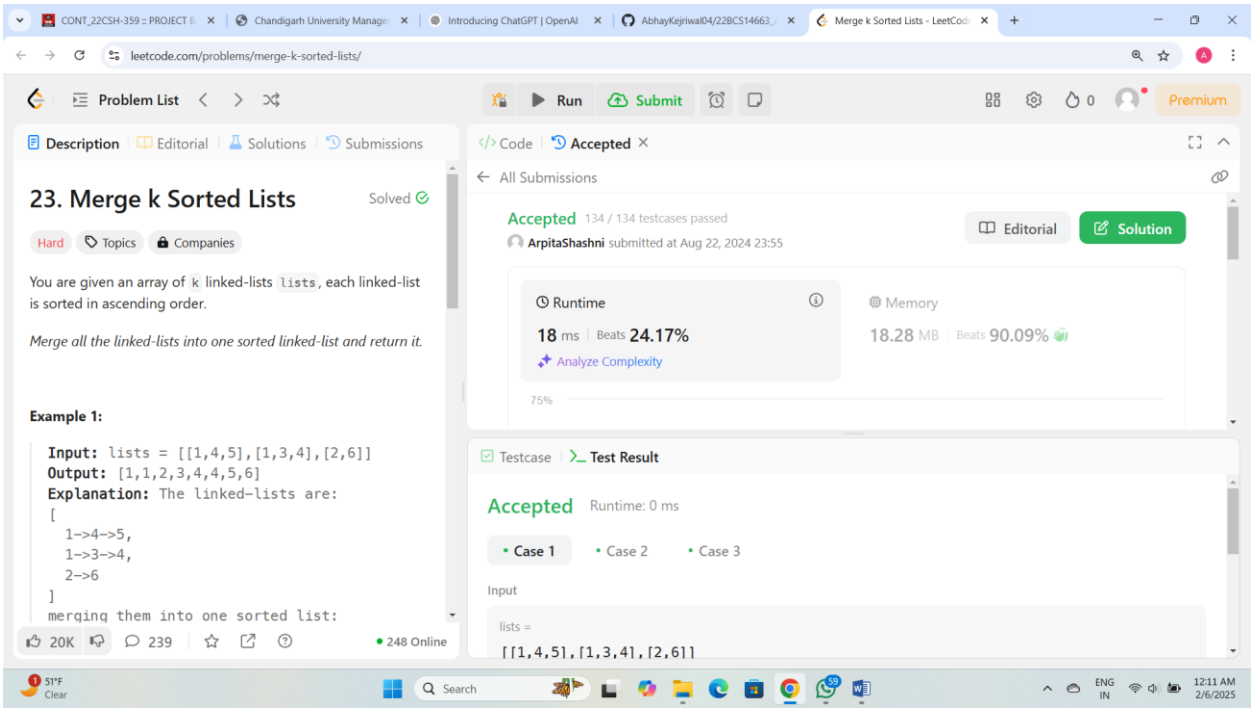


Problem 10: Merge k sorted lists

```
/**
 * 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) {}
 * };
```

```
 */
 #include<queue>
class Solution {
public:
    class compare{
    public:
        bool operator()(ListNode* a,ListNode* b){
            return a->val > b->val;
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*,vector<ListNode*>,compare> minheap;
        int k=lists.size();
        if(k==0){
            return NULL;
        }
        for(int i=0;i<k;i++){
            if(lists[i]!=NULL){
                minheap.push(lists[i]);

            }
        }
        ListNode* head=NULL;
        ListNode* tail=NULL;
        while(minheap.size()>0){
            ListNode* temp=minheap.top();
            minheap.pop();
            if(head==NULL){
                head=temp;
                tail=temp;
                if(head->next!=NULL){
                    minheap.push(tail->next);
                }
            }
            else{
                tail->next=temp;
                tail=temp;
                if(tail->next!=NULL){
                    minheap.push(tail->next);
                }
            }
        }
        return head;
    }
};
```



Problem 11: Sort list

```
/**
 * 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* getMiddle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
    ListNode* merge(ListNode* left, ListNode* right) {
        if (!left) return right;
        if (!right) return left;
        ListNode* dummy = new ListNode(0);
```

```
        ListNode* curr = dummy;
        while (left && right) {
            if (left->val < right->val) {
                curr->next = left;
                left = left->next;
            } else {
                curr->next = right;
                right = right->next;
            }
            curr = curr->next;
        }
        if (left) curr->next = left;
        if (right) curr->next = right;
        return dummy->next;
    }
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* mid = getMiddle(head);
        ListNode* rightHead = mid->next;
        mid->next = nullptr;
        ListNode* left = sortList(head);
        ListNode* right = sortList(rightHead);
        return merge(left, right);
    }
};
```