**Remove duplicates from a sorted arrayCW**

class Solution {

public:

int removeDuplicates(vector<int>& nums) {

set<int> st;

for(int i=0;i<nums.size();i++){

st.insert(nums[i]);

}

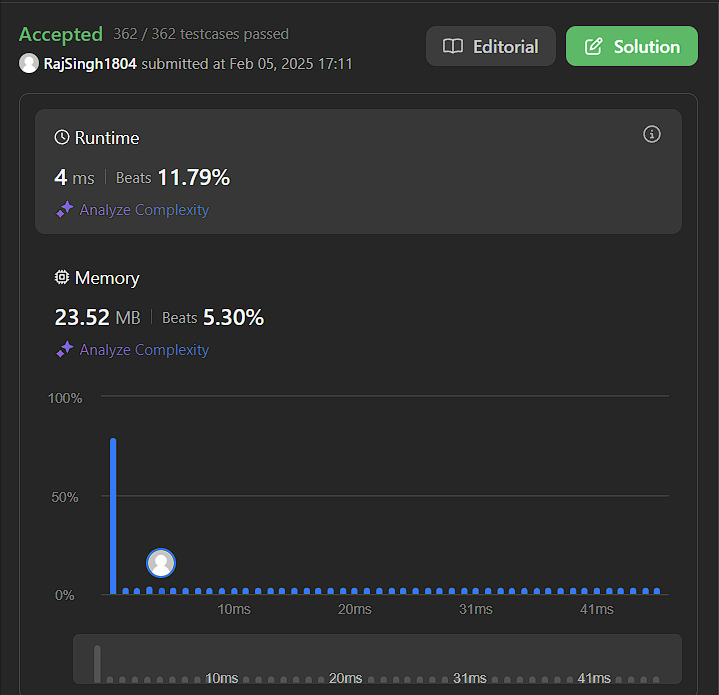
nums.clear();

for(auto i:st){

nums.push\_back(i);

}

return nums.size();



**Implementing insertion sortCW**

void insert(vector<int> & arr, int 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;

}

void insertionSort(vector<int>& arr) {

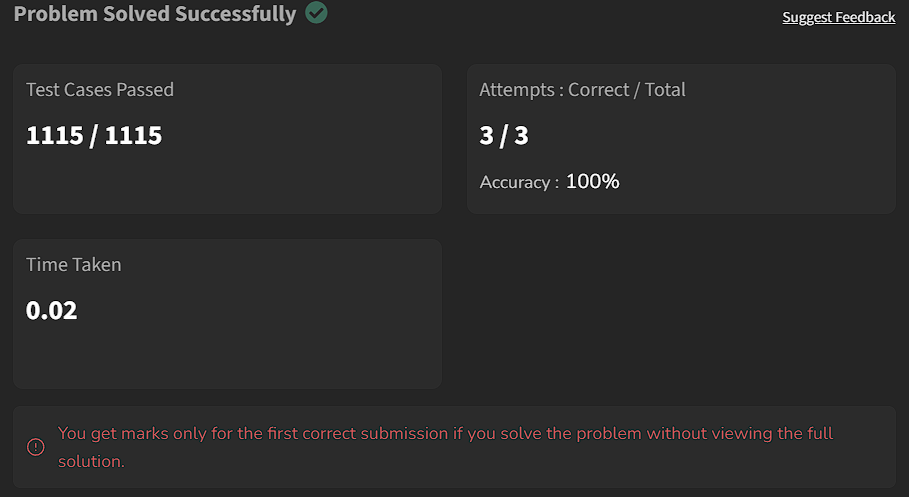
// code here

for(int i = 1; i < arr.size(); i++){

insert(arr, i);

}

}



**Contains duplicateHW**

bool containsDuplicate(vector<int>& nums) {

        bool cond;

        sort(nums.begin(),nums.end());

        for(int i=0;i<nums.size()-1;i++){

            if(nums[i]==nums[i+1]){

                cond=true;

                break;

            }

            else{

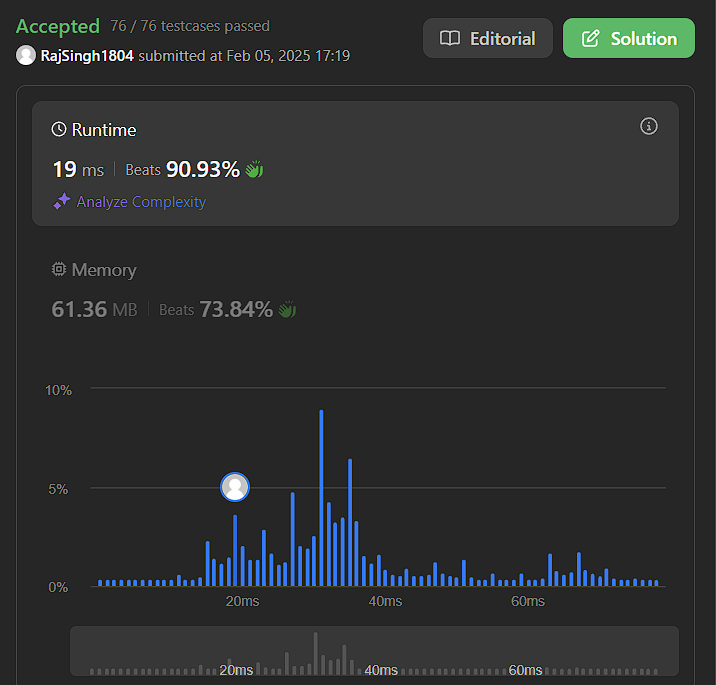
                cond=false;

            }

        }

        return cond;

    }



**Two SumCW**

vector<int> twoSum(vector<int>& nums, int target) {

       unordered\_map<int, int> mpp;

    for (int i = 0; i < nums.size(); i++) {

        int num = nums[i];

        int req = target - num;

        if (mpp.find(req) != mpp.end()) {

            return {mpp[req], i};

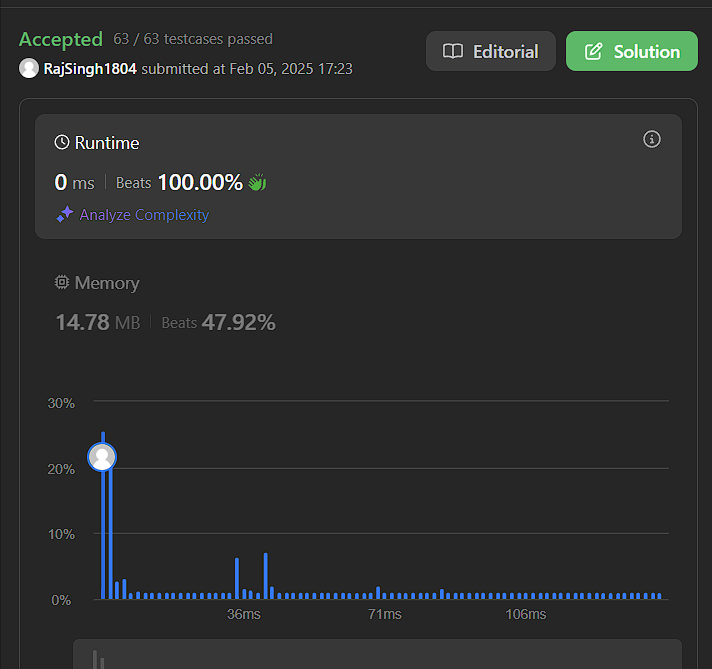
        }

        mpp[num] = i;

    }

    return { -1, -1};

}



**Majority ElementHW**

int majorityElement(vector<int>& nums) {

        map<int,int> mp;

        int greater;

        for(int i=0;i<nums.size();i++){

            mp[nums[i]]++;

        }

        for(auto i:mp){

            if(i.second>(nums.size()/2)){

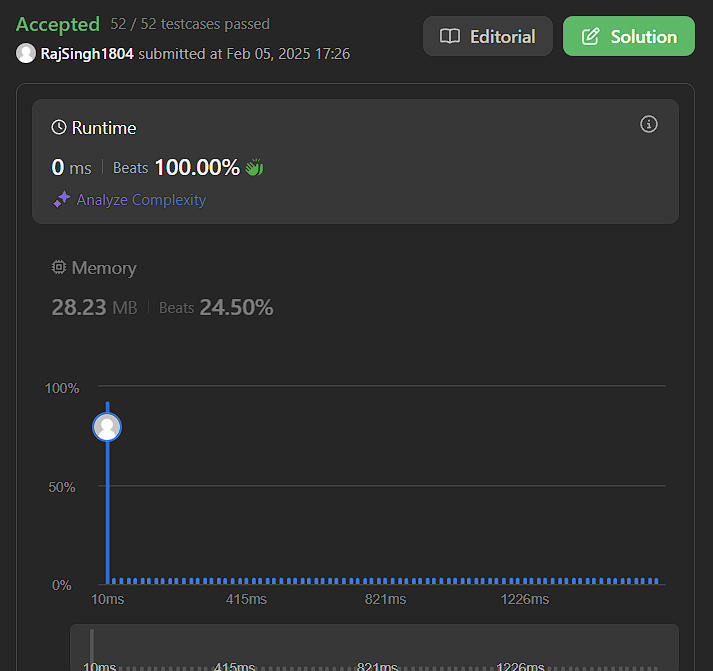
                greater=i.first;

            }

        }

        return greater;

    }



**Valid PalindromCW**

bool isPalindrome(string s) {

        int l = 0;

        int e = s.size() - 1;

        while (l <= e) {

            if (!isalnum(s[l])) {

                l++;

            } else if (!isalnum(s[e])) {

                e--;

            } else {

                char t1 = tolower(s[l]);

                char t2 = tolower(s[e]);

                if (t1 != t2) return false;

                l++;

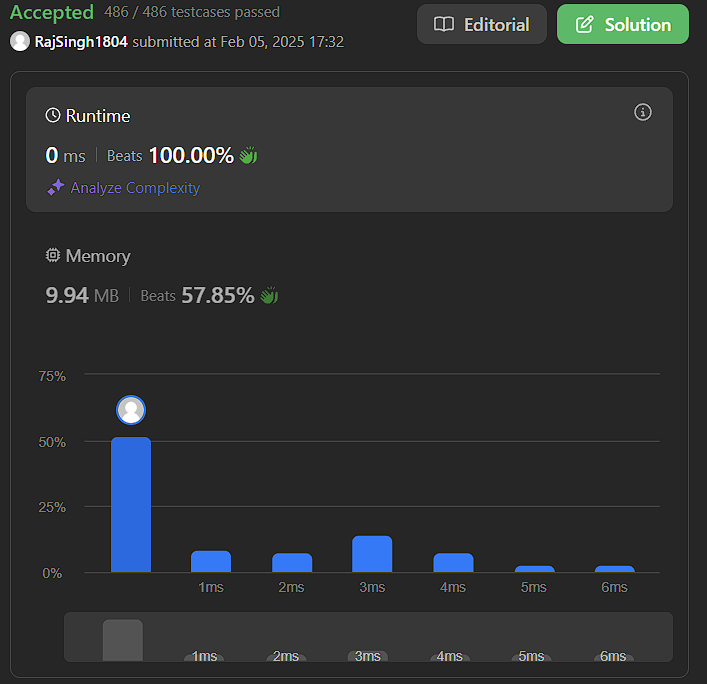
                e--;

            }

        }

        return true;

    }



**Set Matrix zeroesCW**

void setZeroes(vector<vector<int>>& matrix) {

int n=matrix.size();

int m=matrix[0].size();

int col0=1;

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

for(int j=0;j<m;j++){

if(matrix[i][j]==0){

matrix[i][0]=0;

if(j!=0){

matrix[0][j]=0;

}

else{

col0=0;

}

}

}

}

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

for(int j=1;j<m;j++){

if(matrix[i][j]!=0){

if(matrix[0][j]==0||matrix[i][0]==0){

matrix[i][j]=0;

}

}

}

}

if(matrix[0][0]==0){

for(int j=0;j<m;j++){

matrix[0][j]=0;

}

}

if(col0==0){

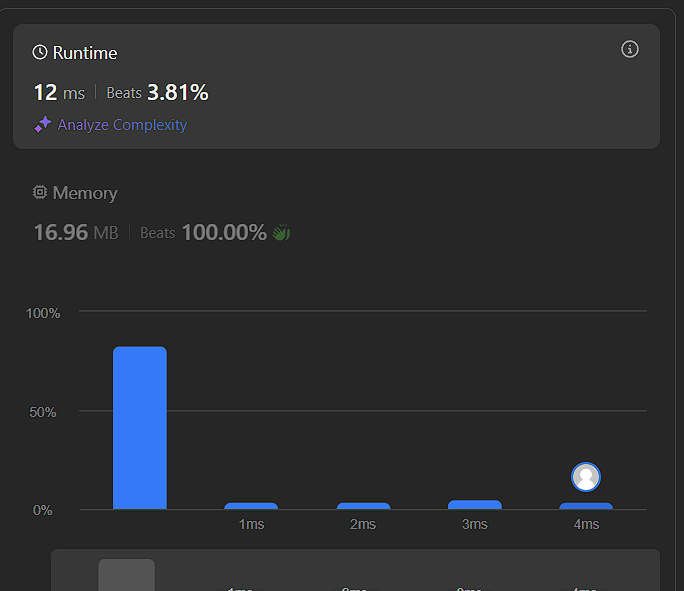
for(int i=0;i<n;i++){

matrix[i][0]=0;

}

}

}



**Print linked listCW**

void printList(Node \*head) {

// your code goes here

Node\* temp = head;

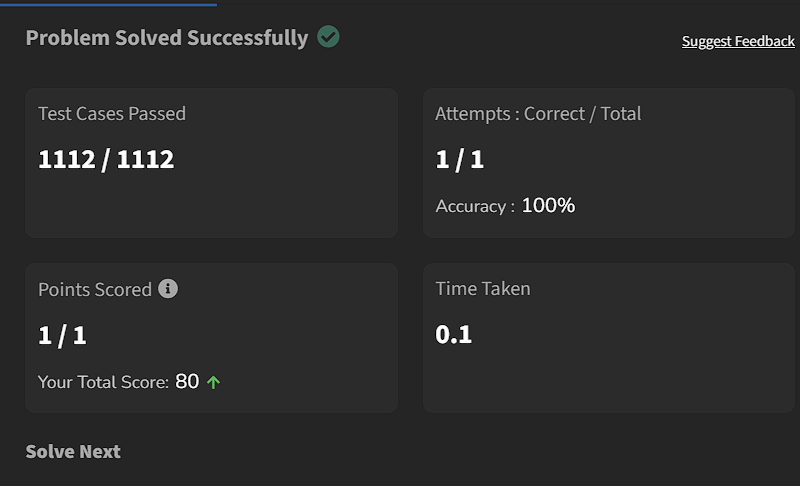
while(temp != NULL){

cout<<temp->data<<" ";

temp = temp->next;

}

}



**Reverse a linked listCW**

ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = NULL;

        ListNode\* cur = head;

        while(cur != NULL){

            ListNode\* temp = cur->next;

            cur->next = prev;

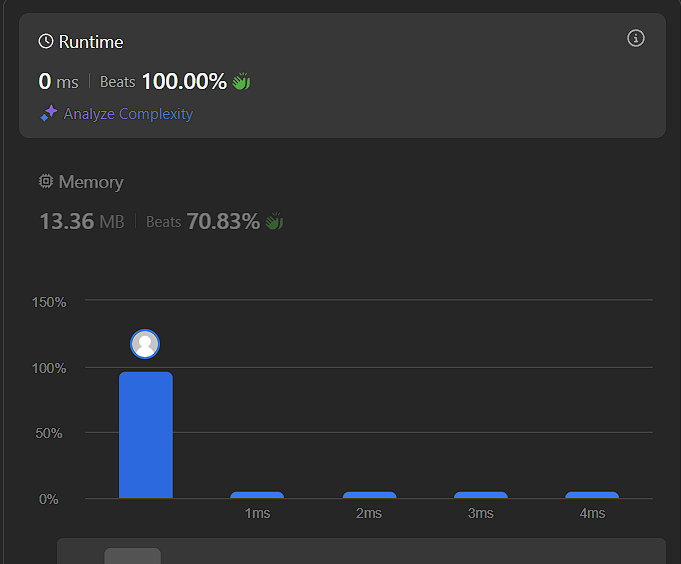
            prev = cur;

            cur = temp;

        }

        return prev;

    }



**Delete middle node of a listCW**

ListNode\* deleteMiddle(ListNode\* head) {

        ListNode\* fast = head;

        ListNode\* slow = head;

        ListNode\* temp;

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

            fast = fast->next->next;

            temp = slow;

            slow = slow->next;

        }

        if(fast == slow) return NULL;

        if(temp != NULL && temp->next !=NULL){

            temp->next = temp->next->next;

        }

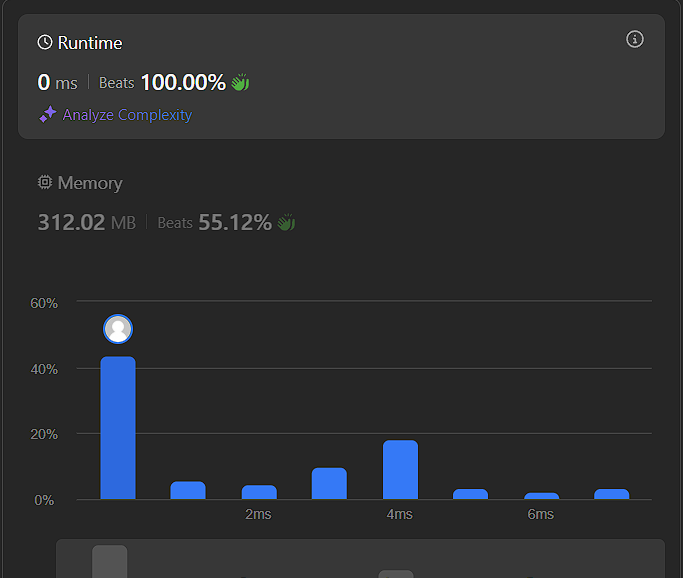
        else{

            temp->next = nullptr;

        }

        return head;

    }



**Merge two sorted linked listsCW**

ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

ListNode\* temp1 = list1;

ListNode\* temp2 = list2;

ListNode\* ll = new ListNode(0);

ListNode\* curr = ll;

while( temp1 != NULL && temp2 != NULL){

if(temp1->val<temp2->val){

curr->next = temp1;

temp1 = temp1->next;

curr = curr->next;

}

else{

curr->next = temp2;

temp2 = temp2->next;

curr = curr->next;

}

}

if(temp1!=NULL){

curr->next = temp1;

}

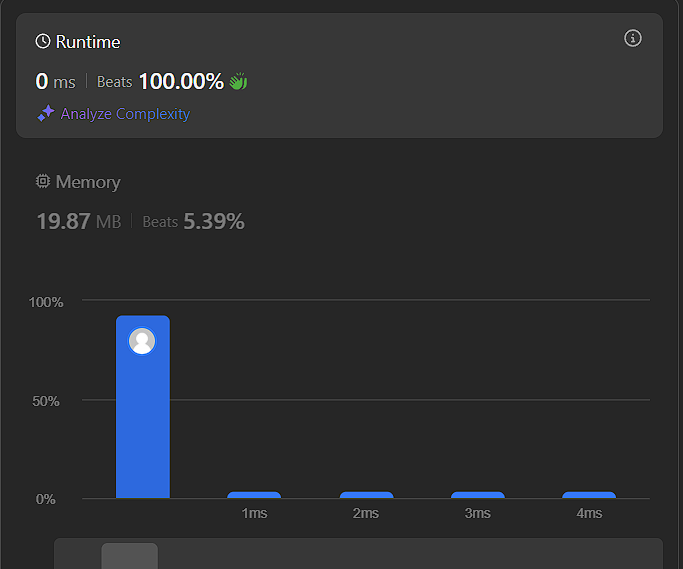
else{

curr->next = temp2;

}

return ll->next;

}



**Detect a cycle in a linked listCW**

bool hasCycle(ListNode \*head) {

        if(head == NULL || head->next == NULL) return false;

        ListNode\* slow = head;

        ListNode\* fast = head;

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

            fast = fast->next->next;

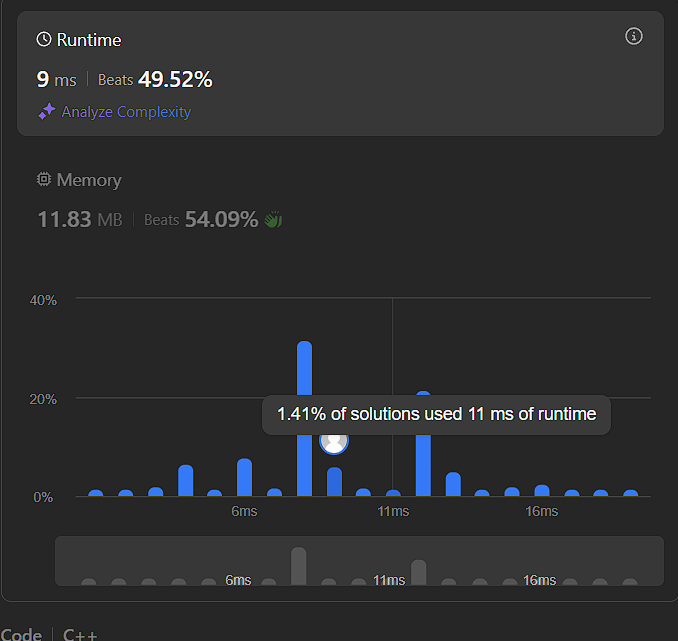
            slow = slow->next;

            if(fast == slow ) return true;

        }

        return false;

    }

****

**Sort ListHW**

ListNode\* findmiddle(ListNode\* head){

        ListNode\* slow = head;

        ListNode\* fast = head->next->next;

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

            slow = slow->next;

            fast = fast->next->next;

        }

        return slow;

    }

    ListNode\* merge(ListNode\* node1, ListNode\* node2){

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

        ListNode\* temp = dummy;

        while(node1 != NULL && node2 !=NULL){

            if(node1->val < node2->val){

                temp->next = node1;

                temp = node1;

                node1 = node1->next;

            }

            else{

                temp->next = node2;

                temp = node2;

                node2 = node2->next;

            }

        }

        if(node1) temp->next = node1;

        else temp->next = node2;

        return dummy->next;

    }

    ListNode\* sortList(ListNode\* head) {

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

        ListNode\* middle = findmiddle(head);

        ListNode\* lefthead = head;

        ListNode\* righthead = middle->next;

        middle->next = nullptr;

        lefthead = sortList(lefthead);

        righthead = sortList(righthead);

        return merge(lefthead, righthead);

    }

