Department of CSE-AI & ML

III Year II Sem

MR22-1 CS0208- DESIGN AND ANALYSIS OF ALGORITHMS

Holiday Assignment Questions

LEETCODE DAA Challenges

**Name:** P. Manik Tanay Reddy

**Roll No:** 2211cs020308

**Section:** AIML Zeta

You are expected to work through the DAA exercises and challenges available at the following

link: <https://leetcode.com/problem-list/linked-list/>

**Q1. Find the Index of the First occurrence in a String:**

**Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.**

**Input: haystack = "sadbutsad", needle = "sad"**

**Output: 0**

**Explanation: "sad" occurs at index 0 and 6.**

**The first occurrence is at index 0, so we return 0.**

**Example 2:**

**Input: haystack = "leetcode", needle = "leeto"**

**Output: -1**

**Explanation: "leeto" did not occur in "leetcode", so we return -1.**

**Constraints:**

* **1 <= haystack.length, needle.length <= 104**
* **haystack and needle consist of only lowercase English characters.**

**1A. Code:**

#include <string>

using namespace std;

class Solution {

public:

int strStr(string haystack, string needle) {

// Find the position of the first occurrence of needle in haystack

size\_t pos = haystack.find(needle);

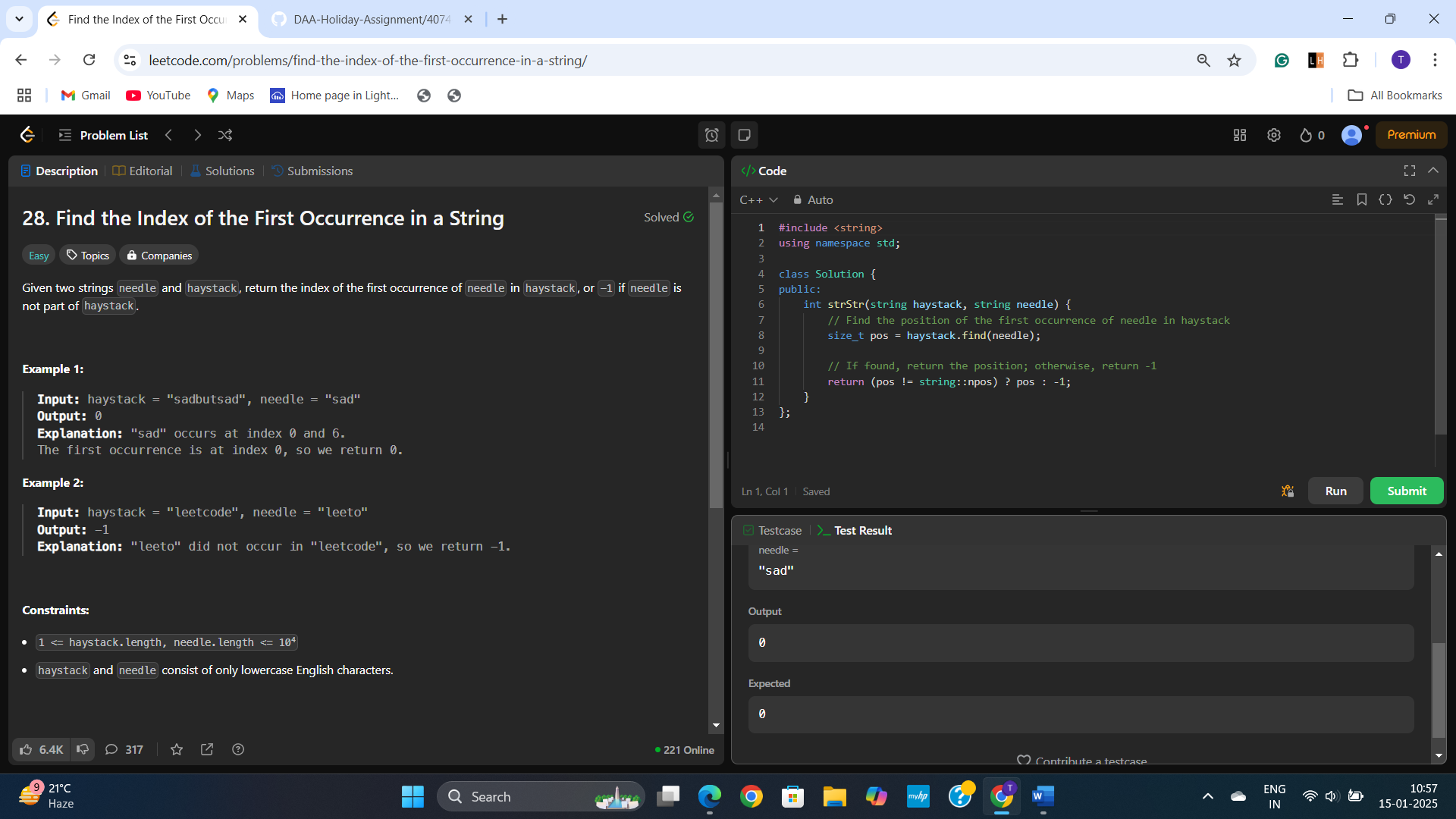
// If found, return the position; otherwise, return -1

return (pos != string::npos) ? pos : -1;

}

};

**Output:**



**Q2. Bitwise and of Number Range e**

**201. Bitwise AND of Numbers Range**

**Given two integers left and right that represent the range [left, right], return *the bitwise AND of all numbers in this range, inclusive*.**

**Example 1:**

**Input: left = 5, right = 7**

**Output: 4**

**Example 2:**

**Input: left = 0, right = 0**

**Output: 0**

**Example 3:**

**Input: left = 1, right = 2147483647**

**Output: 0**

**Constraints:**

* **0 <= left <= right <= 231 - 1**

**2A. Code:**

class Solution {

public:

int rangeBitwiseAnd(int left, int right) {

//Keep shifting left and right until they become equal

int shiftCount = 0;

while (left < right) {

left >>= 1;

right >>= 1;

shiftCount++;

}

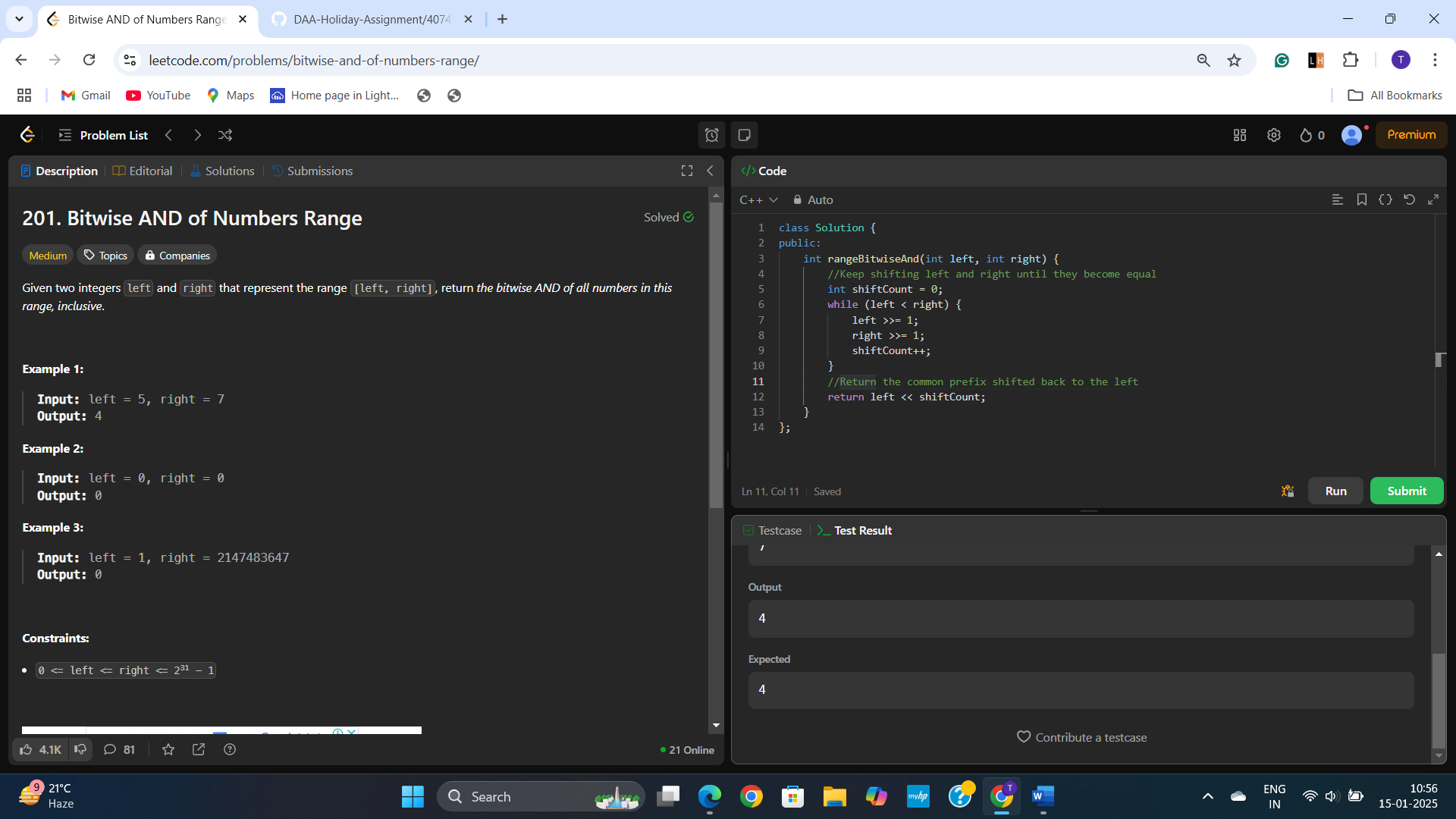
//Return the common prefix shifted back to the left

return left << shiftCount;

}

};

**Output:**



**Q3. Square Root**

**69. Sqrt(x)**

**Given a non-negative integer x, return *the square root of*x*rounded down to the nearest integer*. The returned integer should be non-negative as well.**

**You must not use any built-in exponent function or operator.**

* **For example, do not use pow(x, 0.5) in c++ or x \*\* 0.5 in python.**

**Example 1:**

**Input: x = 4**

**Output: 2**

**Explanation: The square root of 4 is 2, so we return 2.**

**Example 2:**

**Input: x = 8**

**Output: 2**

**Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.**

**Constraints:**

* **0 <= x <= 231 – 1**

**3A. Code:**

#include <iostream>

using namespace std;

class Solution {

public:

    int mySqrt(int x) {

        if (x == 0 || x == 1) return x; // Handle edge cases

        int left = 0, right = x, result = 0;

        while (left <= right) {

            long long mid = left + (right - left) / 2; // Use long long to avoid overflow

            long long square = mid \* mid;

            if (square == x) {

                return mid; // Perfect square

            } else if (square < x) {

                result = mid; // Update result

                left = mid + 1; // Search in the right half

            } else {

                right = mid - 1; // Search in the left half

            }

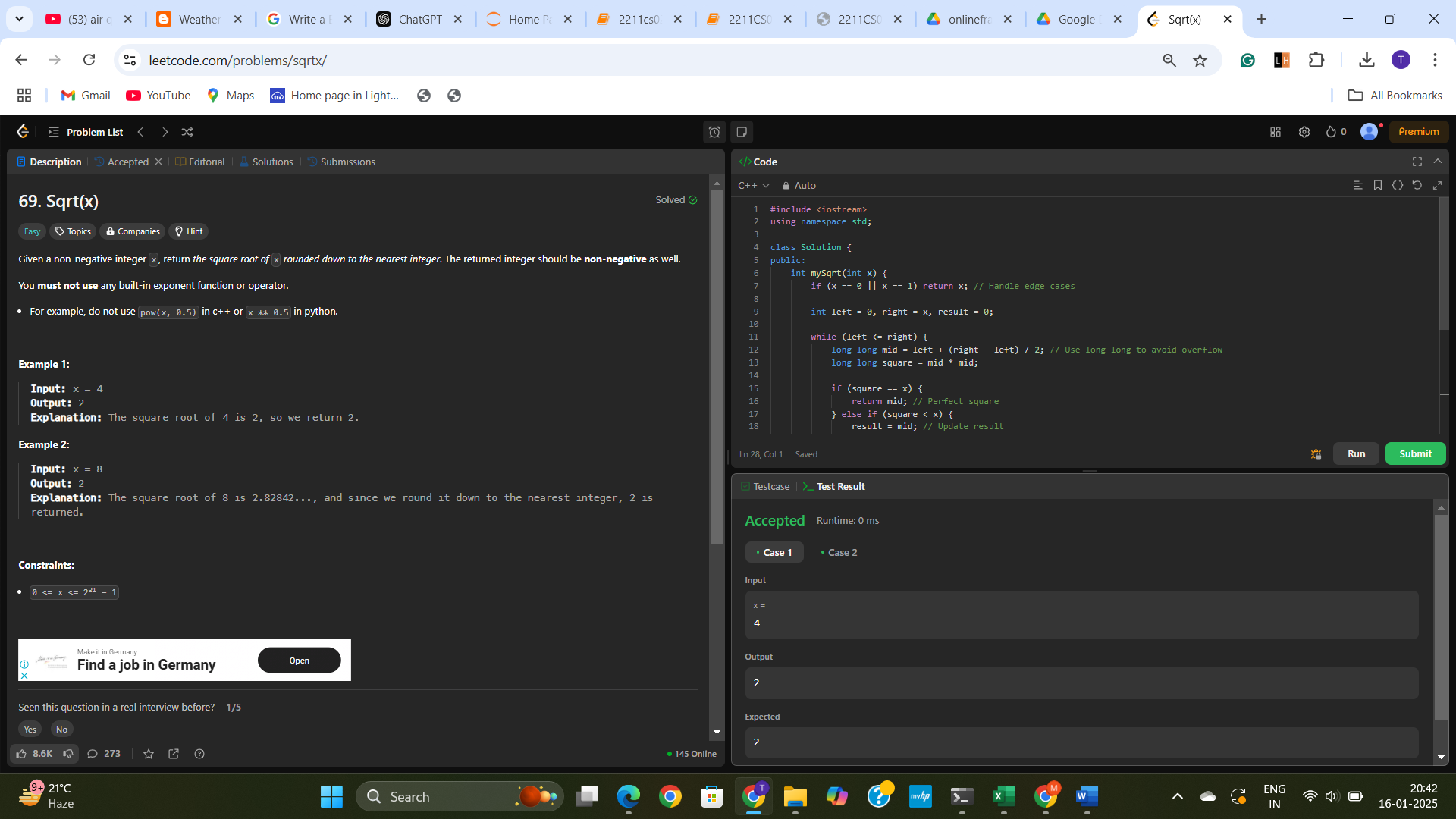
        }

        return result; // Return the largest integer whose square <= x

    }

};

**Output:**



**Q4. Largest Number**

**179. Largest Number**

**Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.**

**Since the result may be very large, so you need to return a string instead of an integer.**

**Example 1:**

**Input: nums = [10,2]**

**Output: "210"**

**Example 2:**

**Input: nums = [3,30,34,5,9]**

**Output: "9534330"**

**Constraints:**

* **1 <= nums.length <= 100**
* **0 <= nums[i] <= 109**

**4A. Code:**

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

class Solution {

public:

string largestNumber(vector<int>& nums) {

vector<string> strNums;

for (int num : nums) {

strNums.push\_back(to\_string(num));

}

auto compare = [](const string& a, const string& b) {

return a + b > b + a;

};

sort(strNums.begin(), strNums.end(), compare);

string result;

for (const string& str : strNums) {

result += str;

}

if (result[0] == '0') {

return "0";

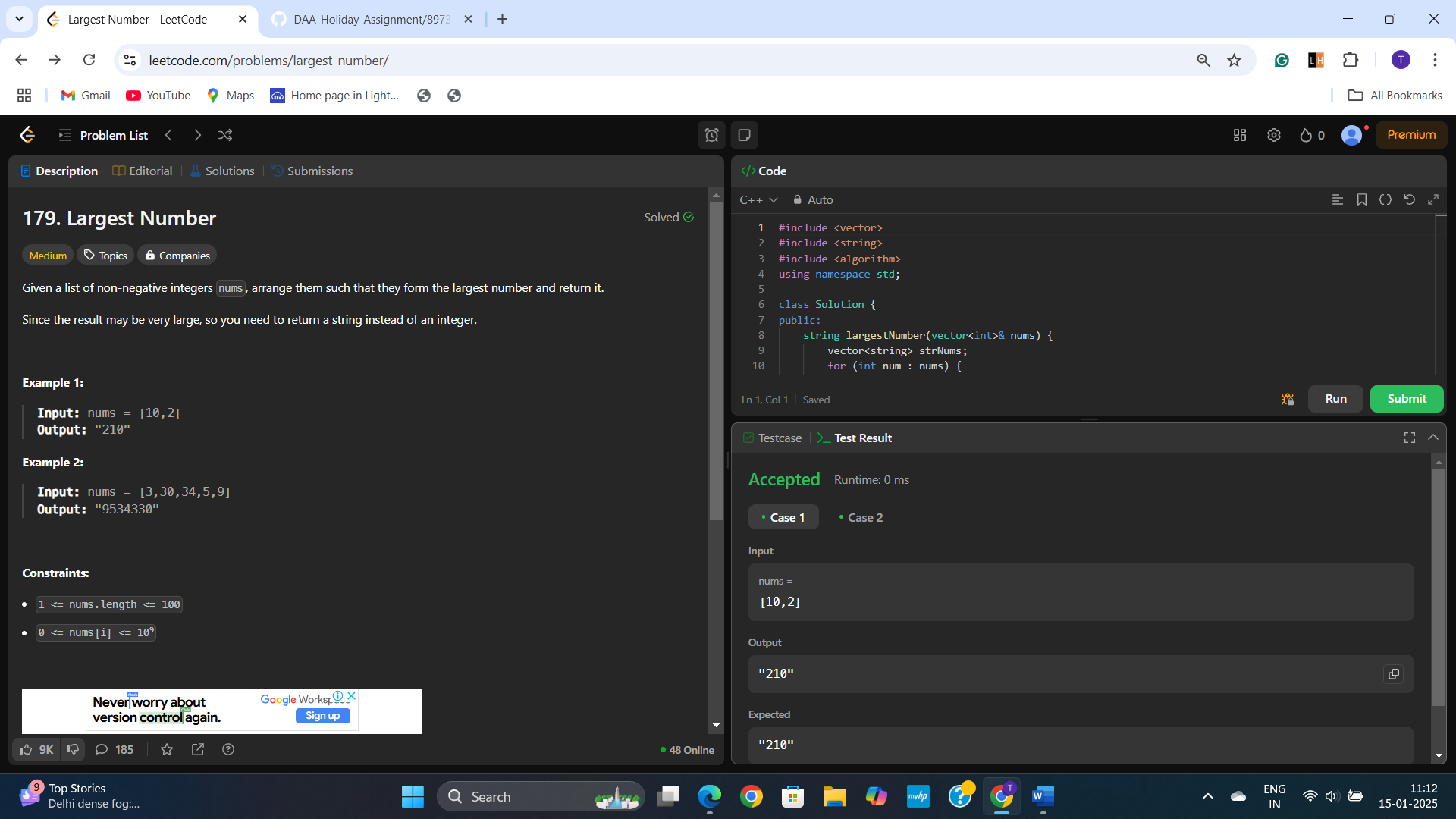
}

return result;

}

};

**Output:**

****

**Q5. Valid Parenthesis**

**20. Valid Parentheses**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

**Example 4:**

**Input:** s = "([])"

**Output:** true

**Constraints:**

* 1 <= s.length <= 104
* s consists of parentheses only '()[]{}'.

**5A. Code:**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

class Solution {

public:

bool isValid(string s) {

stack<char> stack;

for (char c : s) {

// If the character is an opening bracket, push it onto the stack

if (c == '(' || c == '{' || c == '[') {

stack.push(c);

}

// If the character is a closing bracket, check if it matches the top of the stack

else if (c == ')' || c == '}' || c == ']') {

if (stack.empty()) {

return false; // No opening bracket to match with

}

char top = stack.top();

stack.pop();

// Check if the top of the stack matches the corresponding opening bracket

if ((c == ')' && top != '(') || (c == '}' && top != '{') || (c == ']' && top != '[')) {

return false;

}

}

}

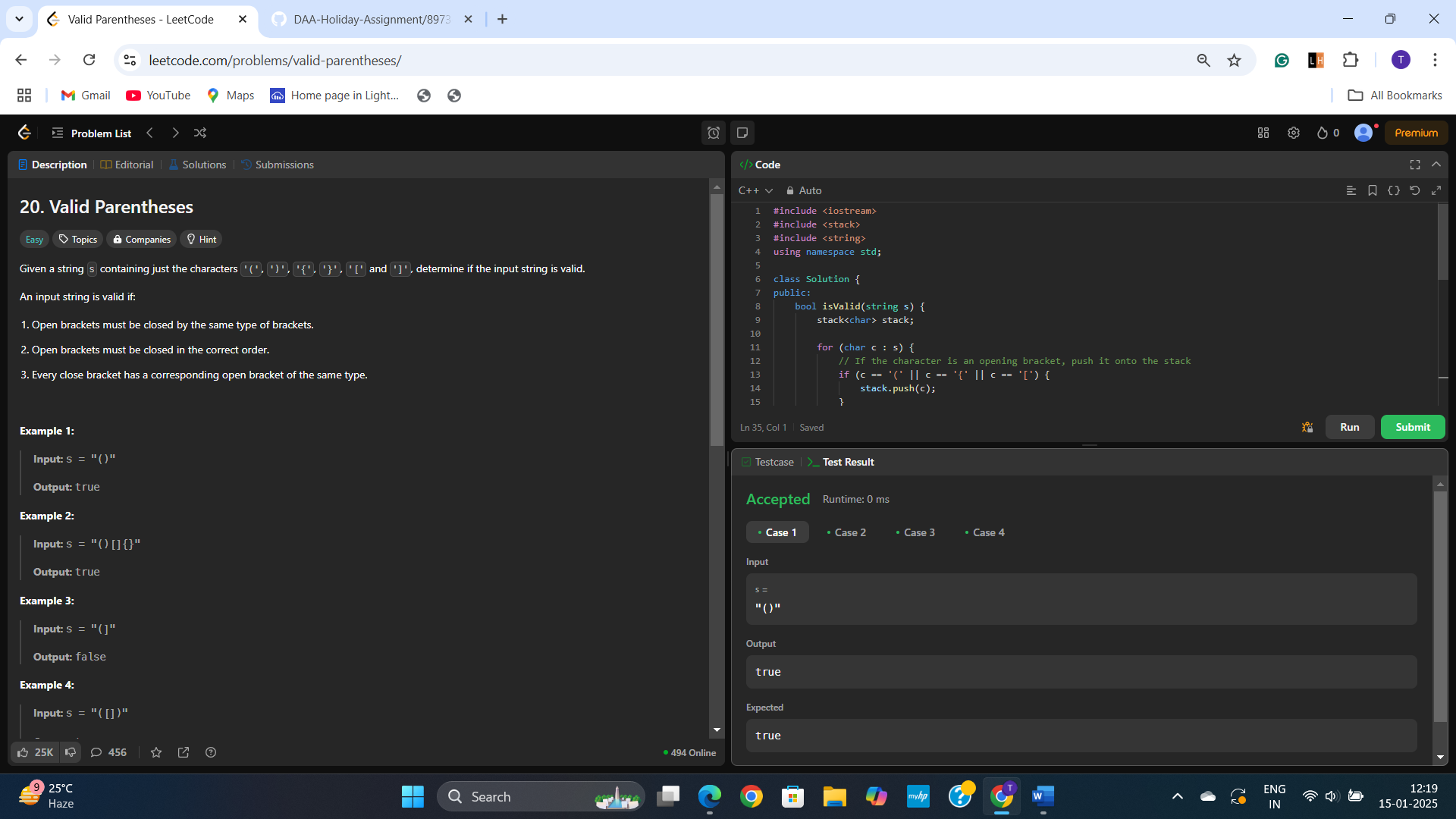
// If the stack is empty, all brackets were matched, otherwise return false

return stack.empty();

}

};

**Output:**

****

**Q6. Merge two Sorted List**

**21. Merge Two Sorted Lists**

**You are given the heads of two sorted linked lists list1 and list2.**

**Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.**

**Return *the head of the merged linked list*.**

**Example 1:**

**[](https://camo.githubusercontent.com/61bfb646a6ffe622c02f5d1e5a8be1e76f188a0e69e00a6705ed10343d7f24f1/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032302f31302f30332f6d657267655f6578312e6a7067)**

**Input: list1 = [1,2,4], list2 = [1,3,4]**

**Output: [1,1,2,3,4,4]**

**Example 2:**

**Input: list1 = [], list2 = []**

**Output: []**

**Example 3:**

**Input: list1 = [], list2 = [0]**

**Output: [0]**

**Constraints:**

* **The number of nodes in both lists is in the range [0, 50].**
* **-100 <= Node.val <= 100**
* **Both list1 and list2 are sorted in non-decreasing order.**

**6A. Code:**

#include <iostream>

#include <vector>

// Include the ListNode header

using namespace std;

class Solution {

public:

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

ListNode\* dummy = new ListNode(0); // Create a dummy node to start the merged list

ListNode\* current = dummy; // Pointer to build the new list

// Traverse both lists

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

if (list1->val <= list2->val) {

current->next = list1; // Attach list1 node to merged list

list1 = list1->next; // Move the list1 pointer forward

} else {

current->next = list2; // Attach list2 node to merged list

list2 = list2->next; // Move the list2 pointer forward

}

current = current->next; // Move the current pointer forward in the merged list

}

// If there are remaining nodes in list1 or list2, attach them

if (list1 != nullptr) {

current->next = list1;

} else if (list2 != nullptr) {

current->next = list2;

}

return dummy->next; // Return the merged list starting from the first node

}

};

// Helper function to create a linked list from a vector

ListNode\* createList(const vector<int>& nums) {

ListNode\* head = nullptr;

ListNode\* current = nullptr;

for (int num : nums) {

if (!head) {

head = new ListNode(num);

current = head;

} else {

current->next = new ListNode(num);

current = current->next;

}

}

return head;

}

// Helper function to print the linked list

void printList(ListNode\* head) {

while (head != nullptr) {

cout << head->val << " ";

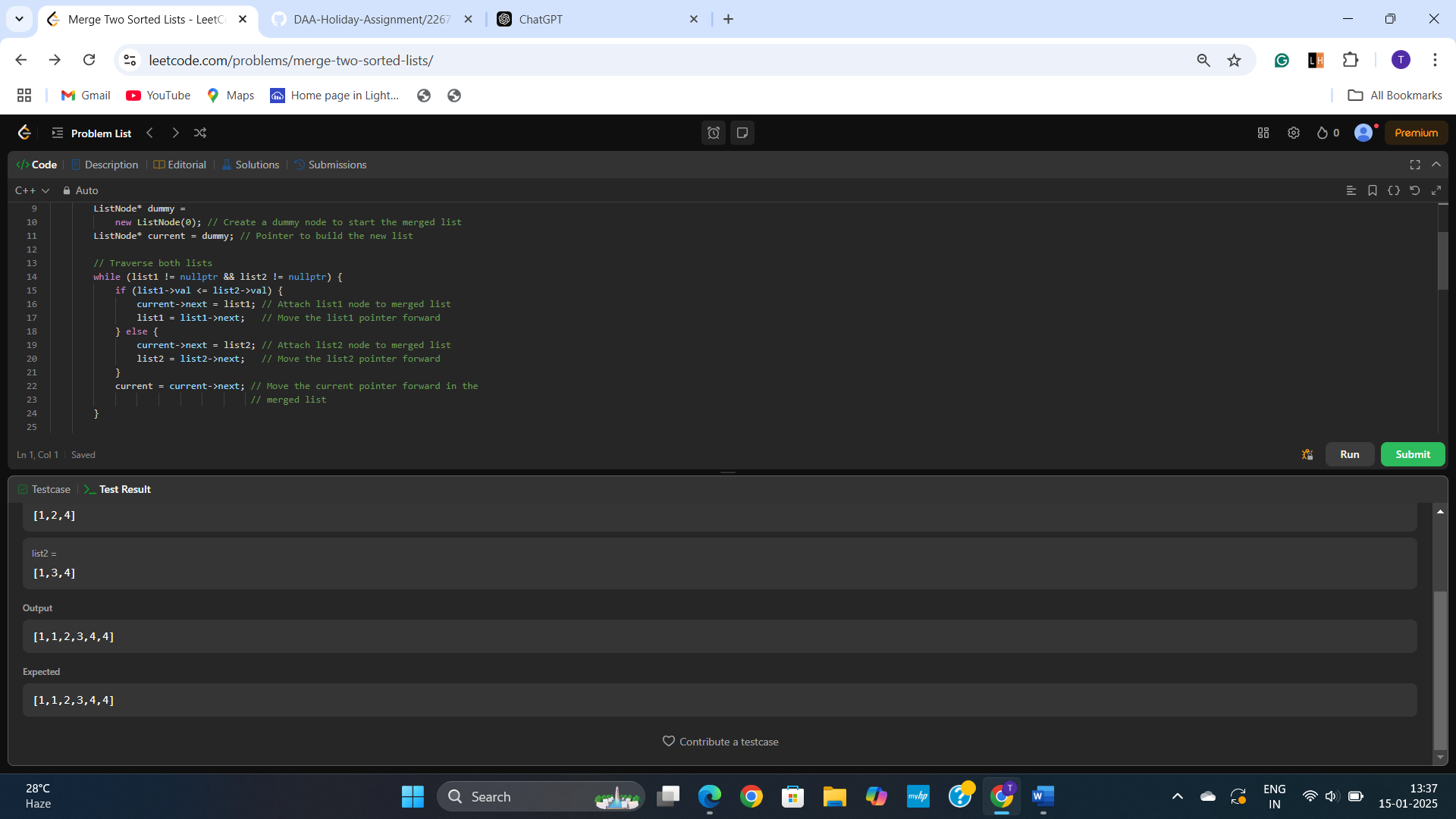
head = head->next;

}

cout << endl;

}

**Output:**

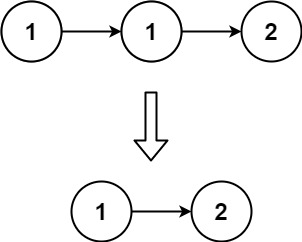
****

**Q7. Remove duplicates from sorted list**

**83. Remove Duplicates from Sorted List**

**Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list sorted as well*.**

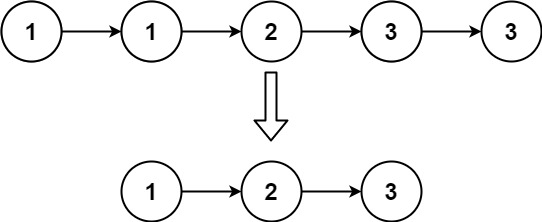
**Example 1:**

**[](https://camo.githubusercontent.com/9d7f578d8fa367a0425014ce8f5e24002c9469e82dae4cb208c9d61d553fc1dd/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032312f30312f30342f6c697374312e6a7067)**

**Input: head = [1,1,2]**

**Output: [1,2]**

**Example 2:**

**[](https://camo.githubusercontent.com/4d28c2173d8d73a19ee8da43c2300c4df82814495048f1dc12b4bef21e2e1c0b/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032312f30312f30342f6c697374322e6a7067)**

**Input: head = [1,1,2,3,3]**

**Output: [1,2,3]**

**Constraints:**

* **The number of nodes in the list is in the range [0, 300].**
* **-100 <= Node.val <= 100**
* **The list is guaranteed to be sorted in ascending order.**

**7A. Code:**

#include <iostream>

using namespace std;

// Assume ListNode structure is already defined and precompiled.

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* current = head;

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

if (current->val == current->next->val) {

// Skip the duplicate node

current->next = current->next->next;

} else {

// Move to the next node

current = current->next;

}

}

return head;

}

};

// Helper function to print the linked list

void printList(ListNode\* head) {

while (head != nullptr) {

cout << head->val << " ";

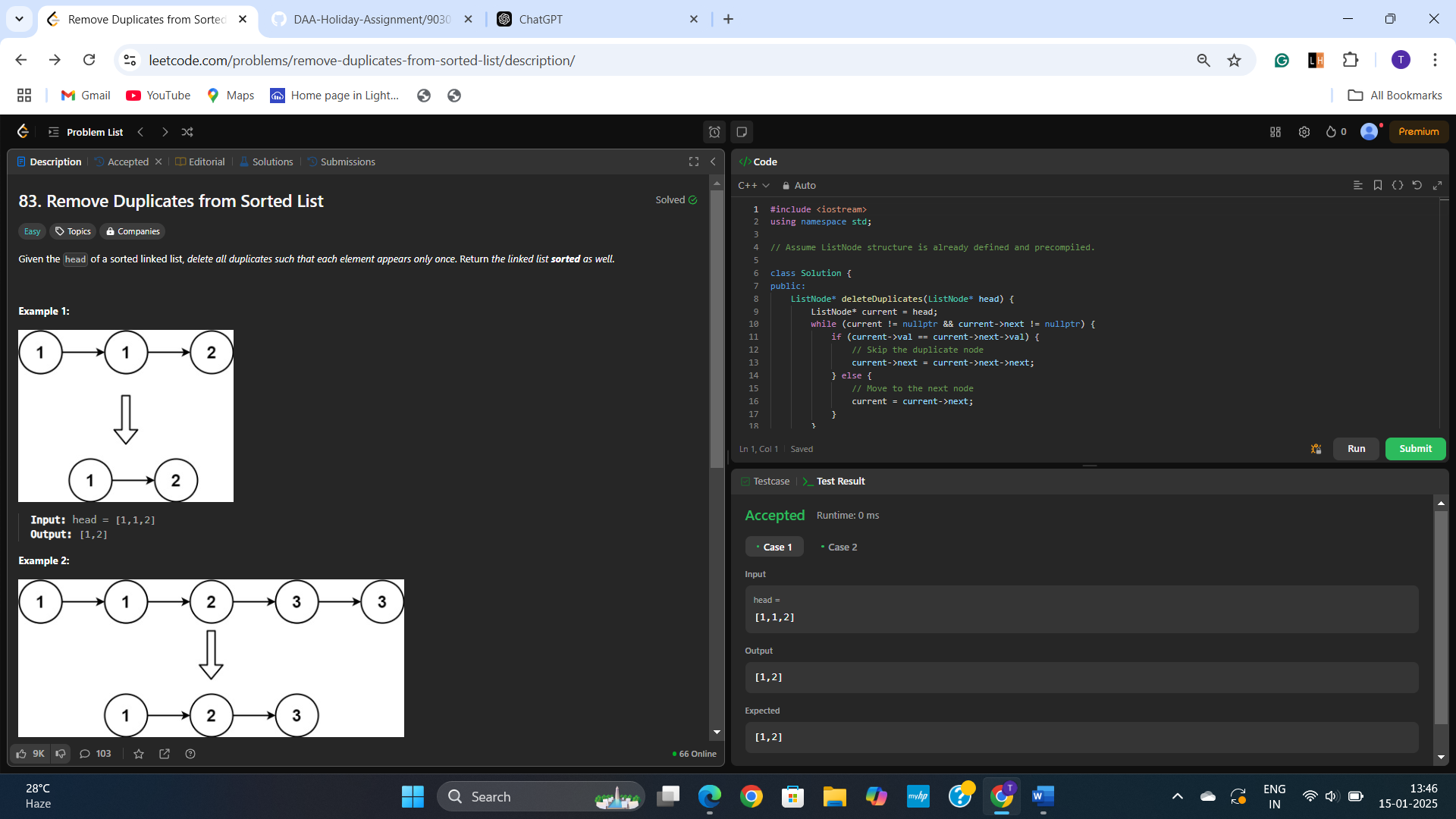
head = head->next;

}

cout << endl;

}

**Output:**



**Q8. Find a Peek Element**

**162. Find Peak Element**

**A peak element is an element that is strictly greater than its neighbors.**

**Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.**

**You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.**

**You must write an algorithm that runs in O(log n) time.**

**Example 1:**

**Input: nums = [1,2,3,1]**

**Output: 2**

**Explanation: 3 is a peak element and your function should return the index number 2.**

**Example 2:**

**Input: nums = [1,2,1,3,5,6,4]**

**Output: 5**

**Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.**

**Constraints:**

* **1 <= nums.length <= 1000**
* **-231 <= nums[i] <= 231 - 1**
* **nums[i] != nums[i + 1] for all valid i.**

**8A. Code:**

#include <iostream>

#include <vector>

using namespace std;

class Solution {

public:

int findPeakElement(vector<int>& nums) {

int left = 0, right = nums.size() - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] > nums[mid + 1]) {

right = mid;

} else {

left = mid + 1;

}

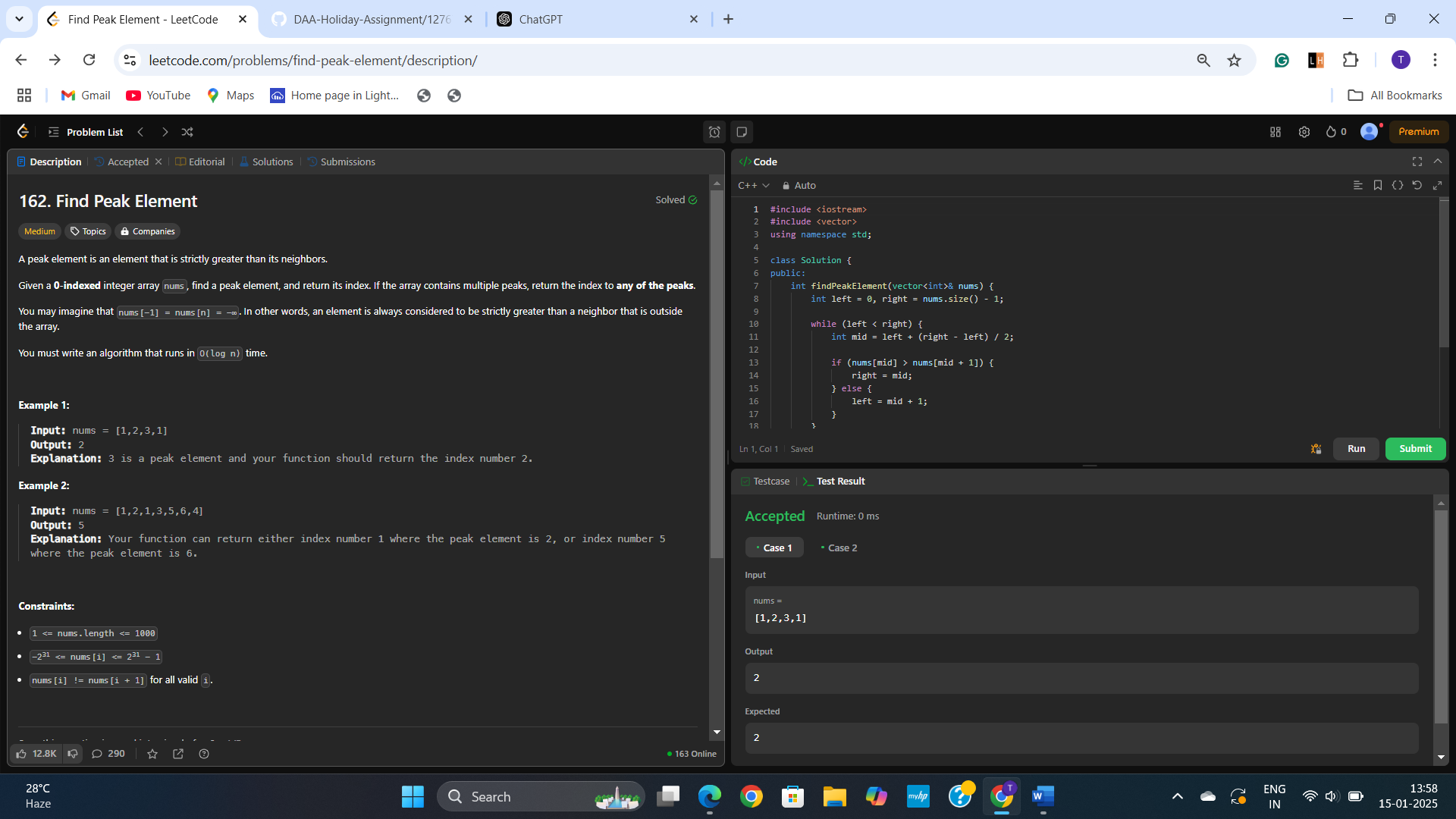
}

return left;

}

};

**Output:**

****

**Q9. Binary Tree: IN order Traversal**

**94. Binary Tree Inorder Traversal**

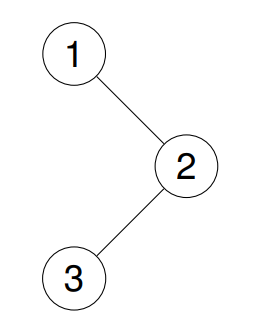
Given the root of a binary tree, return *the inorder traversal of its nodes' values*.

**Example 1:**

**Input:** root = [1,null,2,3]

**Output:** [1,3,2]

**Explanation:**

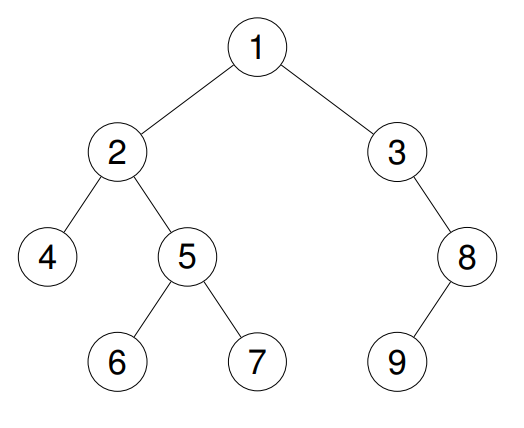
[](https://camo.githubusercontent.com/55f5dc5609831d93bd2a493b6e54cbc8d0b8dd30b9b9c5142457060a95e23381/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032342f30382f32392f73637265656e73686f742d323032342d30382d32392d3230323734332e706e67)

**Example 2:**

**Input:** root = [1,2,3,4,5,null,8,null,null,6,7,9]

**Output:** [4,2,6,5,7,1,3,9,8]

**Explanation:**

[](https://camo.githubusercontent.com/0cdbff84444ce9d495811854103734a12867356b0a72a0f3d7d7afec9ae167fb/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032342f30382f32392f747265655f322e706e67)

**Example 3:**

**Input:** root = []

**Output:** []

**Example 4:**

**Input:** root = [1]

**Output:** [1]

**Constraints:**

* The number of nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**Follow up:** Recursive solution is trivial, could you do it iteratively?

**9A. Code:**

#include <iostream>

#include <vector>

using namespace std;

class Solution {

public:

void inorderTraversalHelper(TreeNode\* root, vector<int>& result) {

if (root == nullptr) {

return;

}

// Traverse left subtree

inorderTraversalHelper(root->left, result);

// Visit the root node

result.push\_back(root->val);

// Traverse right subtree

inorderTraversalHelper(root->right, result);

}

vector<int> inorderTraversal(TreeNode\* root) {

vector<int> result;

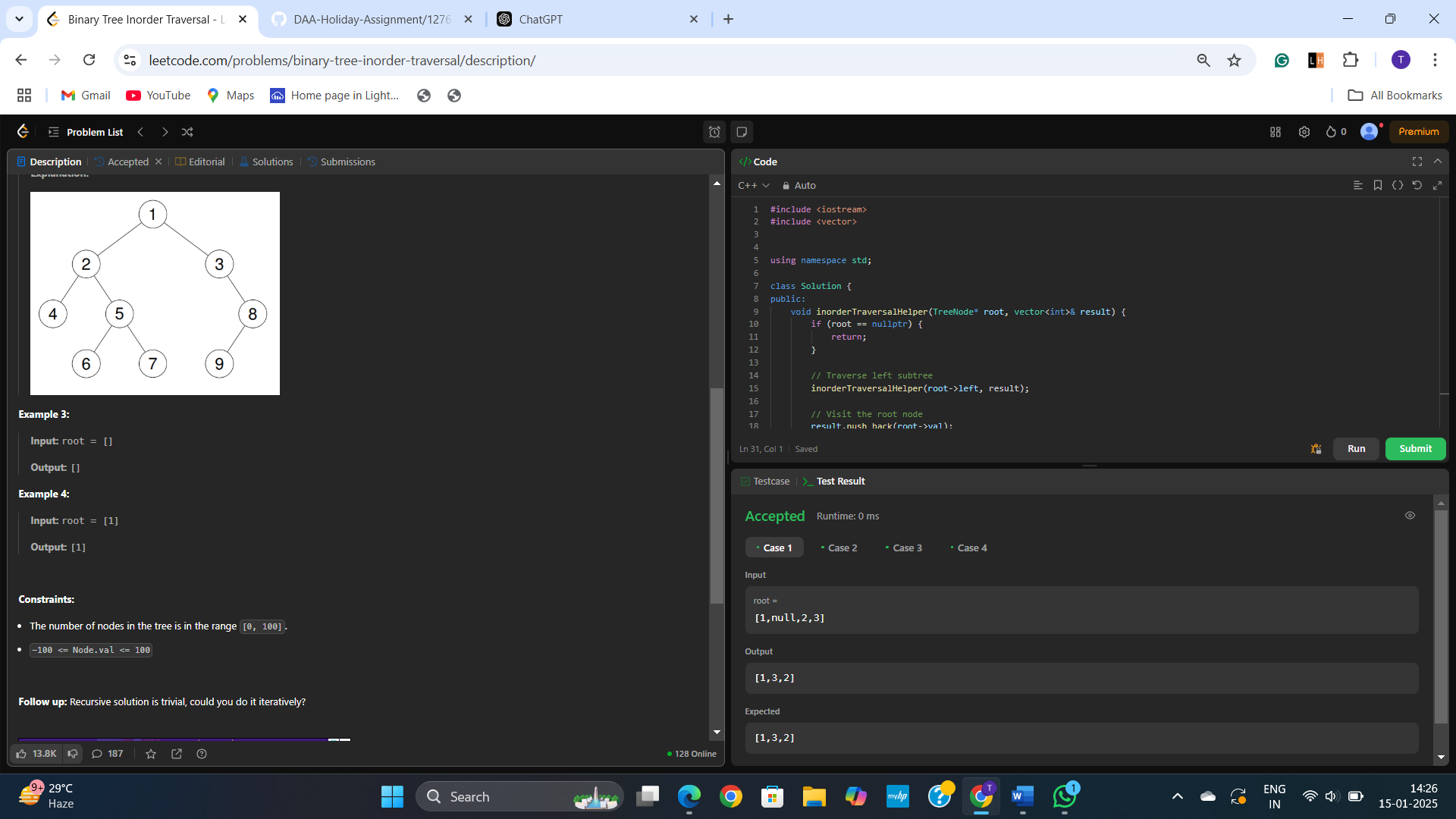
inorderTraversalHelper(root, result);

return result;

}

};

**Output:**

****

**Q10. N-Queens**

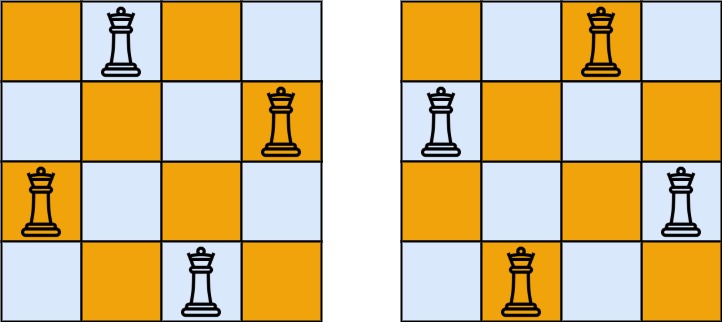
**51. N-Queens**

**The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.**

**Given an integer n, return *all distinct solutions to the n-queens puzzle*. You may return the answer in any order.**

**Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.**

**Example 1:**

**[](https://camo.githubusercontent.com/b1618bce59404eef073f9478edfa7eb4ca03404cc69c7e2af297eaf240b31c4f/68747470733a2f2f6173736574732e6c656574636f64652e636f6d2f75706c6f6164732f323032302f31312f31332f717565656e732e6a7067)**

**Input: n = 4**

**Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]**

**Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above**

**Example 2:**

**Input: n = 1**

**Output: [["Q"]]**

**Constraints:**

* **1 <= n <= 9**

**10A. Code:**

#include <vector>

#include <string>

using namespace std;

class Solution {

public:

    vector<vector<string>> solveNQueens(int n) {

        vector<vector<string>> solutions;

        vector<string> board(n, string(n, '.')); // Initialize an empty n x n board

        vector<int> leftRow(n, 0), upperDiag(2 \* n - 1, 0), lowerDiag(2 \* n - 1, 0);

        backtrack(0, n, board, solutions, leftRow, upperDiag, lowerDiag);

        return solutions;

    }

private:

    void backtrack(int col, int n, vector<string>& board, vector<vector<string>>& solutions,

                   vector<int>& leftRow, vector<int>& upperDiag, vector<int>& lowerDiag) {

        if (col == n) {

            solutions.push\_back(board);

            return;

        }

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

            if (leftRow[row] == 0 && upperDiag[row + col] == 0 && lowerDiag[row - col + n - 1] == 0) {

                board[row][col] = 'Q';

                leftRow[row] = upperDiag[row + col] = lowerDiag[row - col + n - 1] = 1;

                backtrack(col + 1, n, board, solutions, leftRow, upperDiag, lowerDiag);

                // Undo the current placement

                board[row][col] = '.';

                leftRow[row] = upperDiag[row + col] = lowerDiag[row - col + n - 1] = 0;

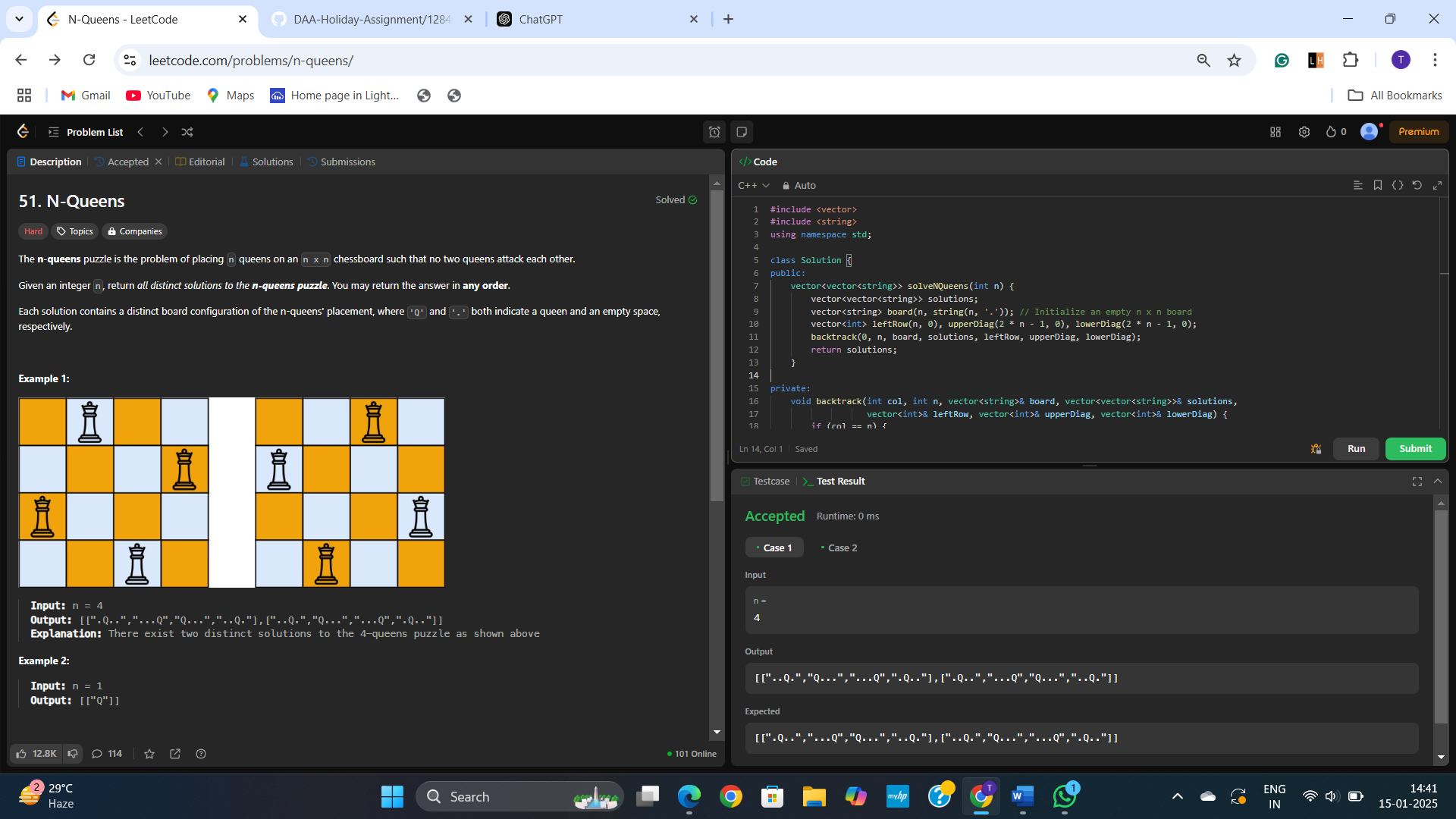
            }

        }

    }

};

**Output:**

****