**Problem 1**

**Aim:**

The Longest Increasing Subsequence

**Code:**

import java.io.\*;

import java.util.\*;

class Result {

public static int longestIncreasingSubsequence(List<Integer> arr) {

if (arr == null || arr.isEmpty()) return 0;

List<Integer> sub = new ArrayList<>();

for (int num : arr) {

int idx = Collections.binarySearch(sub, num);

if (idx < 0) idx = -(idx + 1); // Convert negative index to insertion point

if (idx < sub.size()) {

sub.set(idx, num); // Replace element at found position

} else {

sub.add(num); // Append new element to extend LIS

}

}

return sub.size();

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> arr = new ArrayList<>();

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

int arrItem = Integer.parseInt(bufferedReader.readLine().trim());

arr.add(arrItem);

}

int result = Result.longestIncreasingSubsequence(arr);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

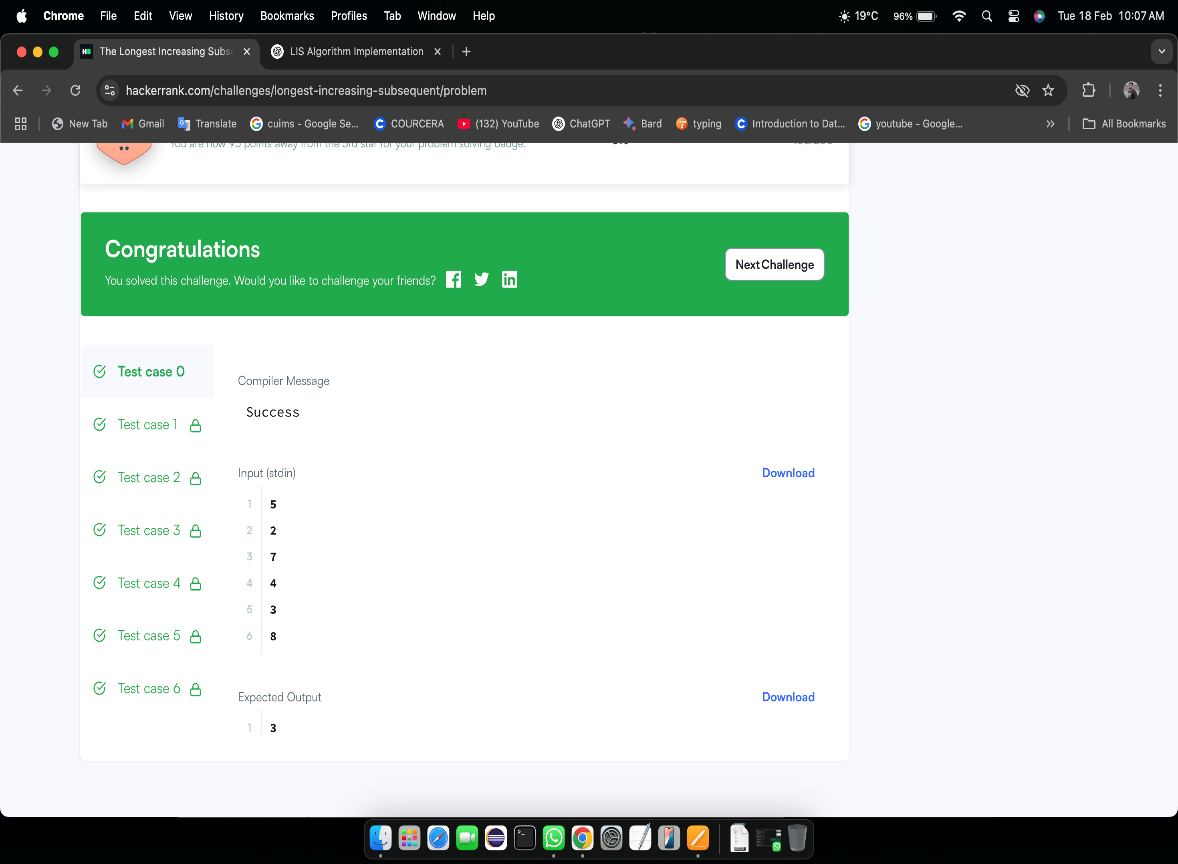
bufferedReader.close();

bufferedWriter.close();

}

}

**Output:**



**Problem 2**

**Aim:**

Maximum Product Subarray

**Code:**

class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0];

int currentSum = nums[0];

for (int i = 1; i < nums.length; i++) {

currentSum = Math.max(nums[i], currentSum + nums[i]);

maxSum = Math.max(maxSum, currentSum);

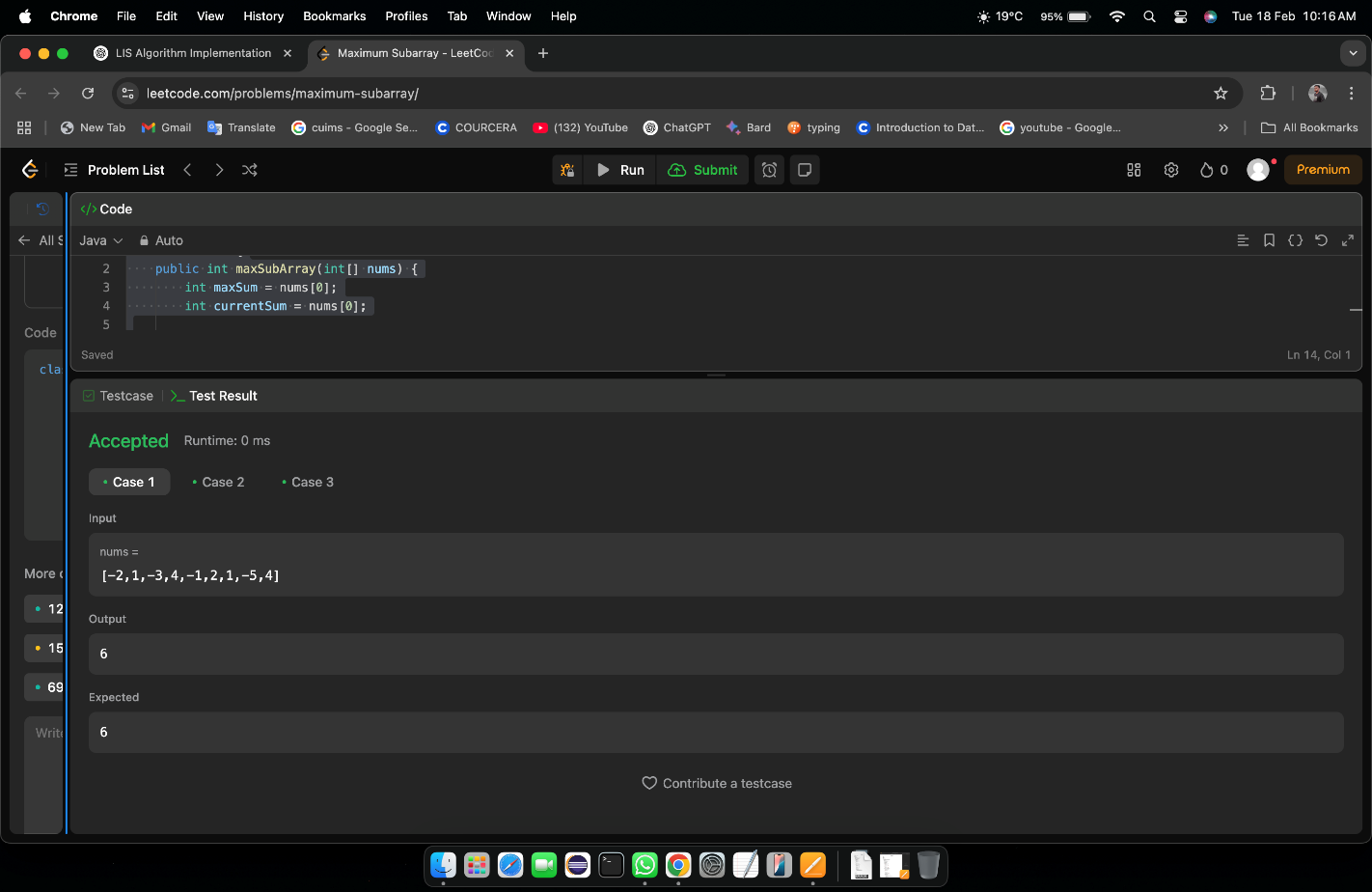
}

return maxSum;

    }

}

**Output:**



Test Case 1

**Problem 3**

**Aim:**

Unique Path

**Code:**

**class Solution {**

**public int uniquePaths(int m, int n) {**

**int N = m + n - 2; // Total moves**

**int K = Math.min(m - 1, n - 1); // Choose the smaller value to reduce computations**

**long result = 1; // Use long to prevent overflow**

**// Compute C(N, K) using iterative multiplication**

**for (int i = 1; i <= K; i++) {**

**result = result \* (N - i + 1) / i;**

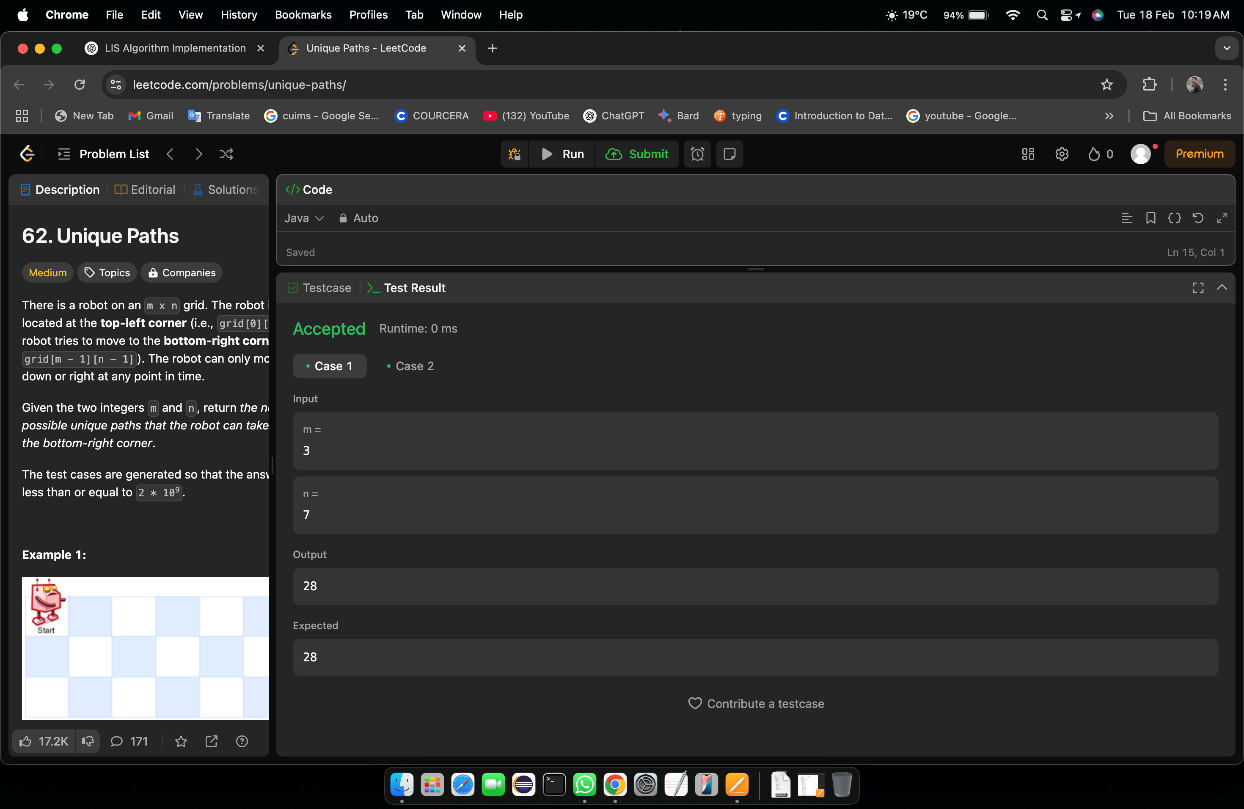
**}**

**return (int) result; // Convert back to int (safe since answer ≤ 2 \* 10^9)**

**}**

**}**

**Output:**

****

Case 1

**Problem 4**

**Aim:**

Coi Change

**Code:**

**import java.util.Arrays;**

**class Solution {**

**public int coinChange(int[] coins, int amount) {**

**int max = amount + 1; // A large value representing "infinity"**

**int[] dp = new int[amount + 1];**

**Arrays.fill(dp, max);**

**dp[0] = 0; // Base case: 0 coins needed for amount 0**

**for (int coin : coins) {**

**for (int i = coin; i <= amount; i++) {**

**dp[i] = Math.min(dp[i], 1 + dp[i - coin]); // DP transition**

**}**

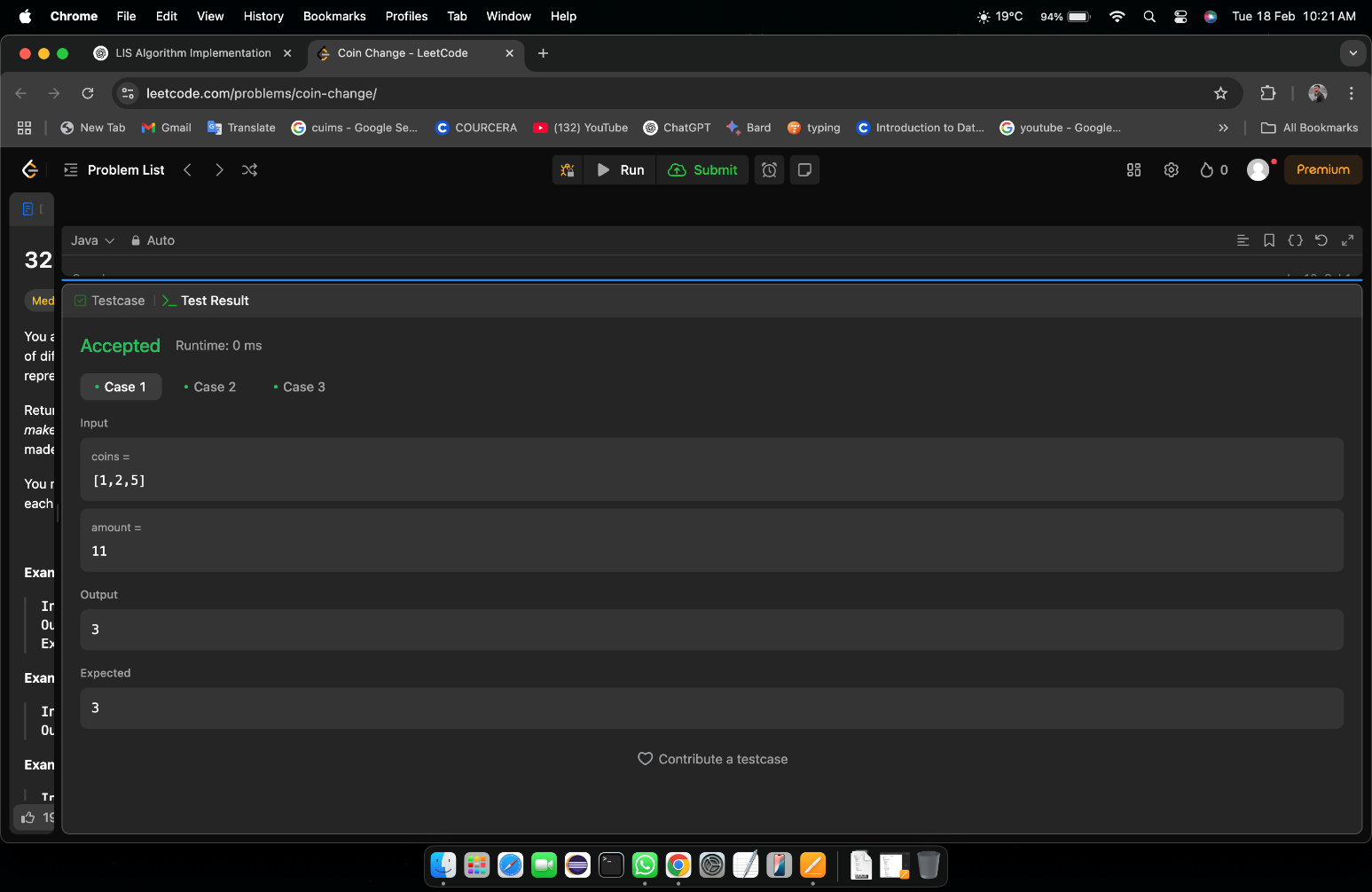
**}**

**return dp[amount] == max ? -1 : dp[amount];**

**}**

**}**

**Output:**

****

**Problem 5**

**Aim:**

Perfect Square

**Code:**

import java.util.\*;

class Solution {

public int numSquares(int n) {

Queue<Integer> queue = new LinkedList<>();

Set<Integer> visited = new HashSet<>();

queue.add(n);

visited.add(n);

int level = 0;

while (!queue.isEmpty()) {

level++;

int size = queue.size();

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

int remainder = queue.poll();

for (int j = 1; j \* j <= remainder; j++) {

int next = remainder - (j \* j);

if (next == 0) return level; // Found the answer

if (!visited.contains(next)) {

queue.add(next);

visited.add(next);

}

}

}

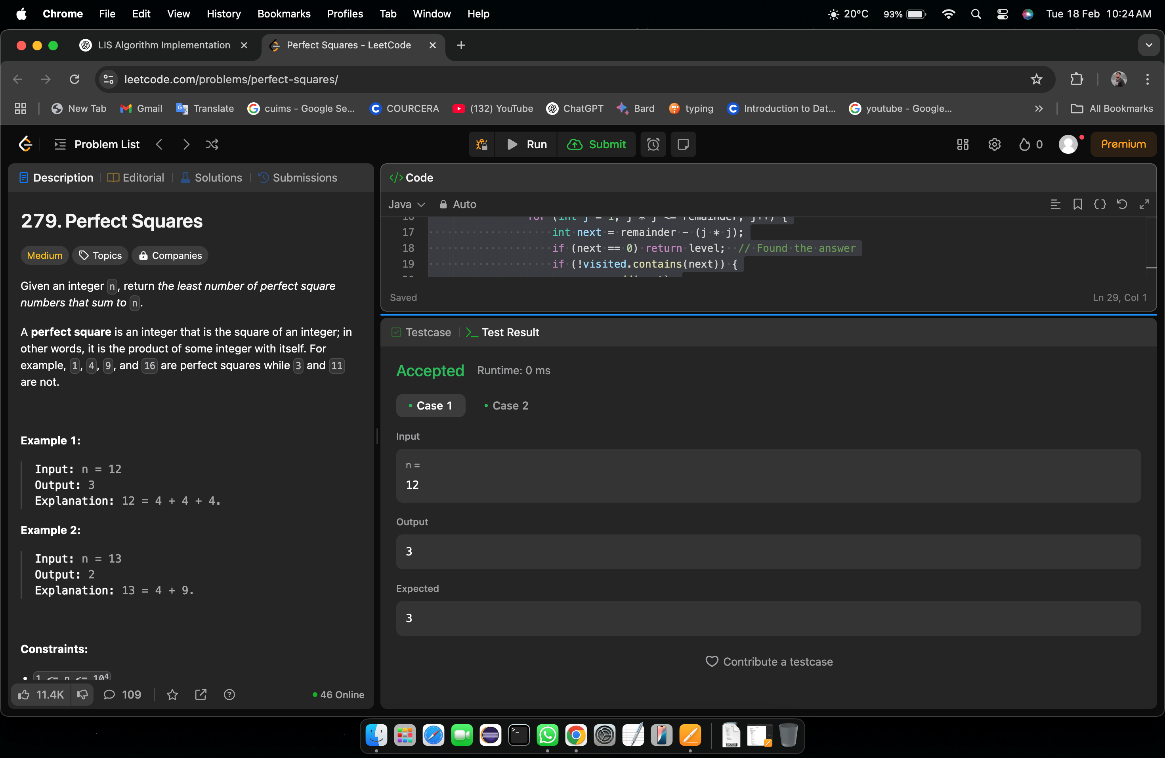
}

return -1;

}

}

**Output:**

****

**Problem 6**

**Aim:**

Super Pow

**Code:**

class Solution {

private:

int solve(int base, int power, int mod) {

int ans = 1;

while (power > 0) {

if (power & 1) {

ans = (ans \* base) % mod;

}

base = (base \* base) % mod;

power >>= 1;

}

return ans;

}

public:

int superPow(int a, vector<int>& b) {

a%=1337;

int n = b.size();

int m = 1140;

int expi = 0;

for(int i : b){

expi = (expi\*10+i)%m;

}

if (expi == 0) {

expi = m;

}

return solve(a,expi,1337);

}

};

**Output:**

A screenshot of a black screen

Description automatically generated A screenshot of a black screen

Description automatically generated A screenshot of a computer

Description automatically generated

Case 1 Case 2 Case 3

**Problem 7**

**Aim:**

Beautiful Array

**Code:**

class Solution {

public:

int partition(vector<int> &v, int start, int end, int mask)

{

int j = start;

for(int i = start; i <= end; i++)

{

if((v[i] & mask) != 0)

{

swap(v[i], v[j]);

j++;

}

}

return j;

}

void sort(vector<int> & v, int start, int end, int mask)

{

if(start >= end) return;

int mid = partition(v, start, end, mask);

sort(v, start, mid - 1, mask << 1);

sort(v, mid, end, mask << 1);

}

vector<int> beautifulArray(int N) {

vector<int> ans;

for(int i = 0; i < N; i++) ans.push\_back(i + 1);

sort(ans, 0, N - 1, 1);

return ans;

}

};

**Output:**

**A screenshot of a black screen

Description automatically generated A screenshot of a computer

Description automatically generated**

Case 1 Case 2

**Problem 8**

**Aim:**

The Skyline Problem

**Code:**

class Solution {

public:

vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

int edge\_idx = 0;

vector<pair<int, int>> edges;

priority\_queue<pair<int, int>> pq;

vector<vector<int>> skyline;

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

const auto &b = buildings[i];

edges.emplace\_back(b[0], i);

edges.emplace\_back(b[1], i);

}

std::sort(edges.begin(), edges.end());

while (edge\_idx < edges.size()) {

int curr\_height;

const auto &[curr\_x, \_] = edges[edge\_idx];

while (edge\_idx < edges.size() &&

curr\_x == edges[edge\_idx].first) {

const auto &[\_, building\_idx] = edges[edge\_idx];

const auto &b = buildings[building\_idx];

if (b[0] == curr\_x)

pq.emplace(b[2], b[1]);

++edge\_idx;

}

while (!pq.empty() && pq.top().second <= curr\_x)

pq.pop();

curr\_height = pq.empty() ? 0 : pq.top().first;

if (skyline.empty() || skyline.back()[1] != curr\_height)

skyline.push\_back({curr\_x, curr\_height});

}

return skyline;

}

};

**Output:**

**A screenshot of a computer

Description automatically generatedA screenshot of a phone

Description automatically generated**

Case 1 Case 2