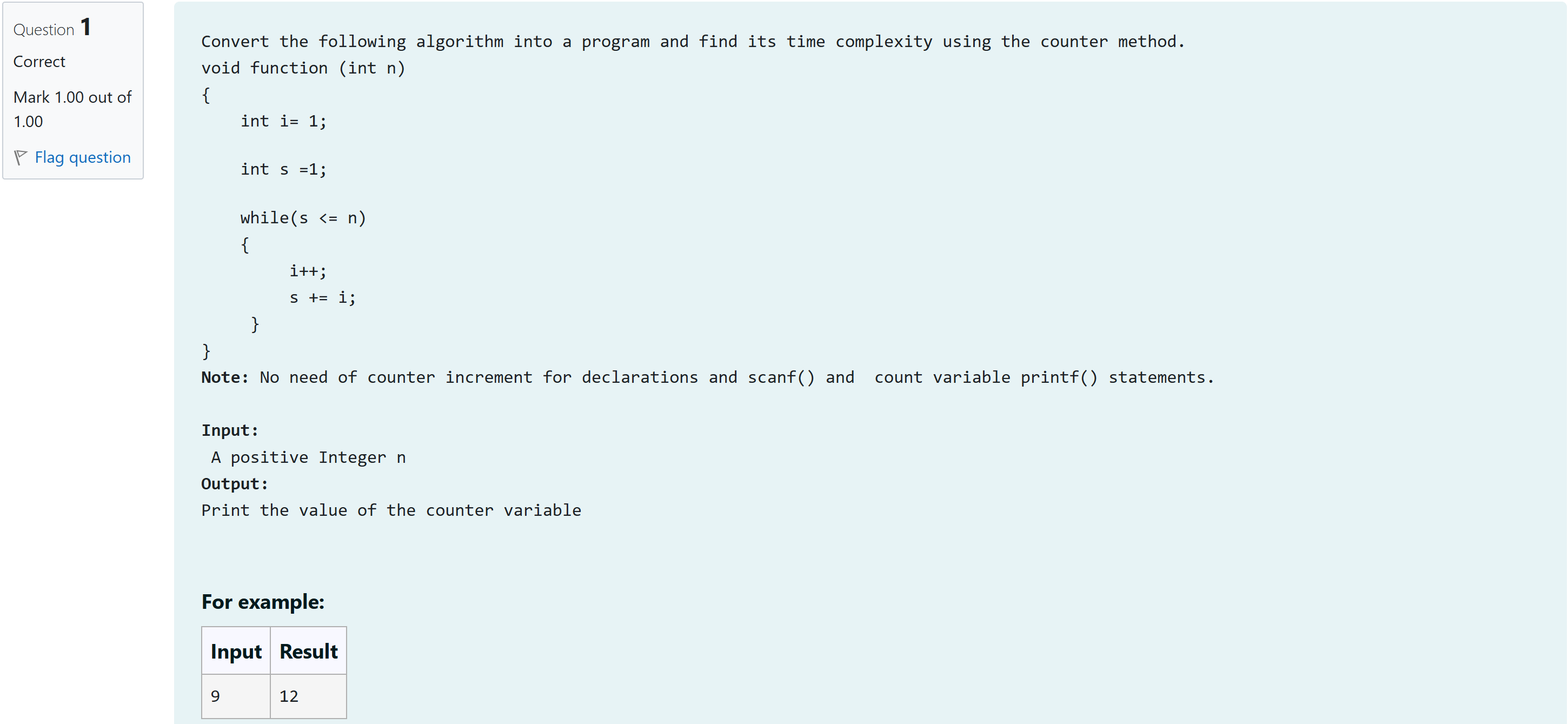
## **Problem 1: Finding Complexity using Counter Method**



#include<stdio.h>

void function (int n)

{

int count=0;

int i= 1;

count++;

int s =1;

count++;

while(s <= n)

{

count++;

i++;

count++;

s += i;

count++;

}

count++;

printf("%d",count);

}

int main()

{

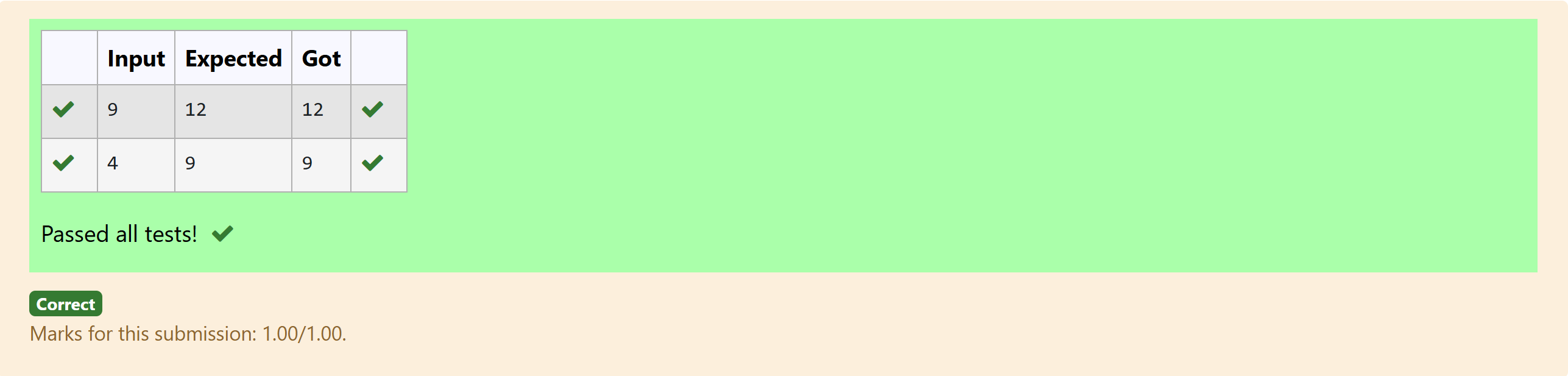
int n;

scanf("%d",&n);

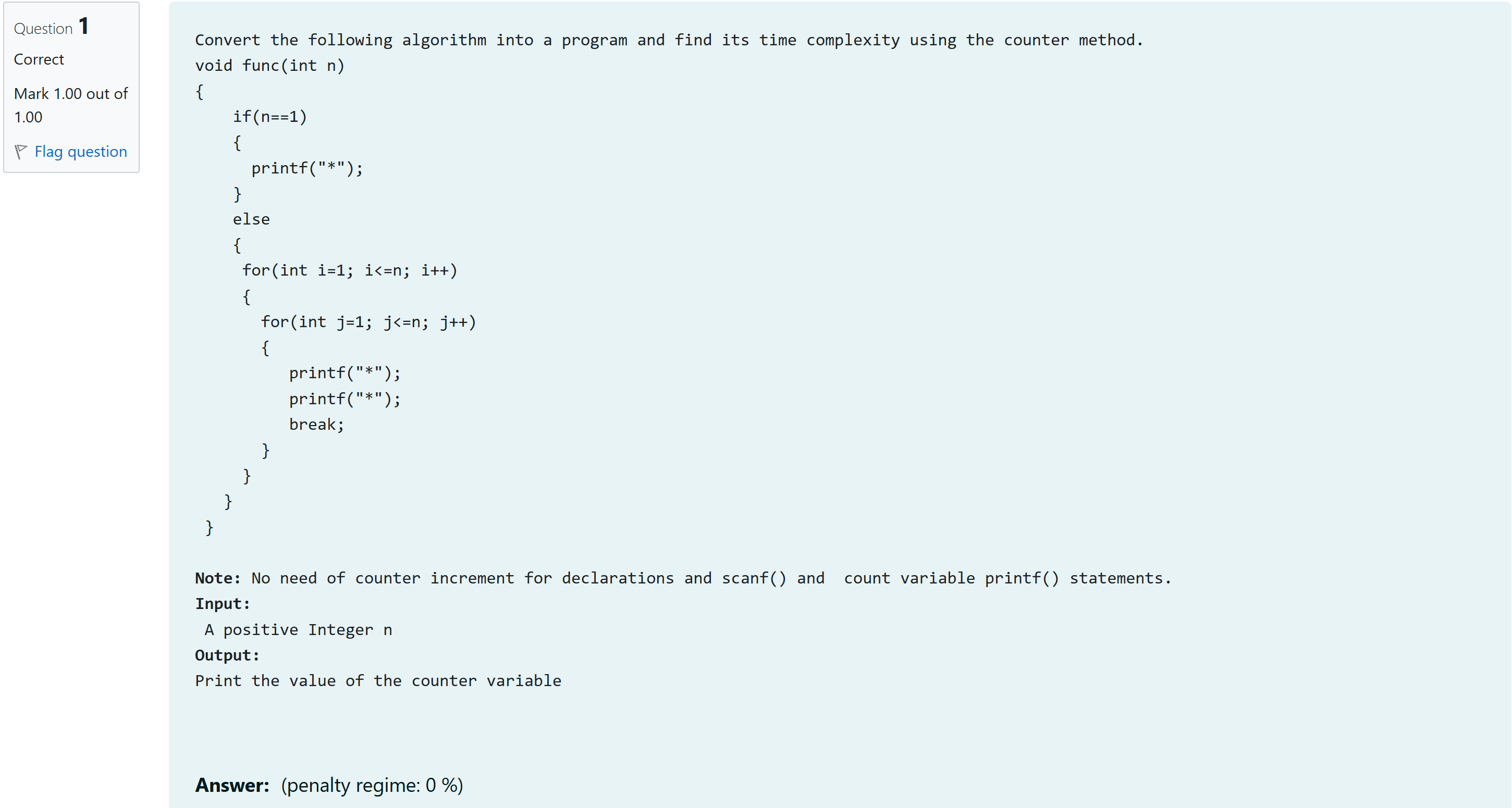
function(n);

}

Output:



## **Problem 2: Finding Complexity using Counter Method**



#include<stdio.h>

void func(int n)

{

int count=0;

if(n==1)

{

count++;

//printf("\*");

count++;

}

else

{

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

{

count++;

for(int j=1; j<=n; j++)

{

count++;

//printf("\*");

count++;

// printf("\*");

count++;

break;

}

count++;

}

count++;

}

count++;

printf("%d",count);

}

int main()

{

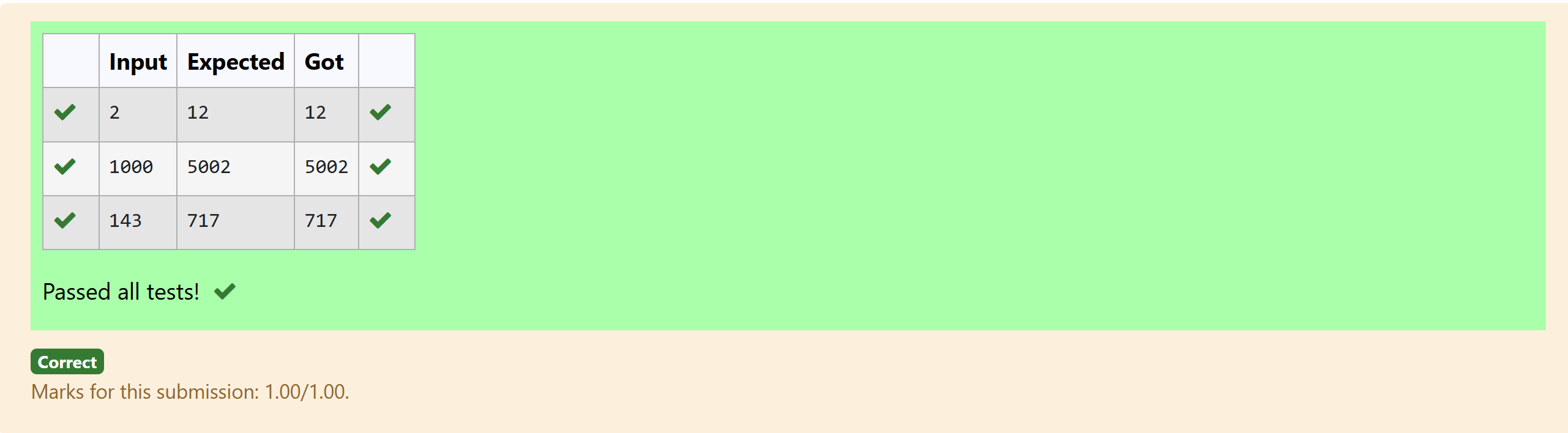
int n;

scanf("%d",&n);

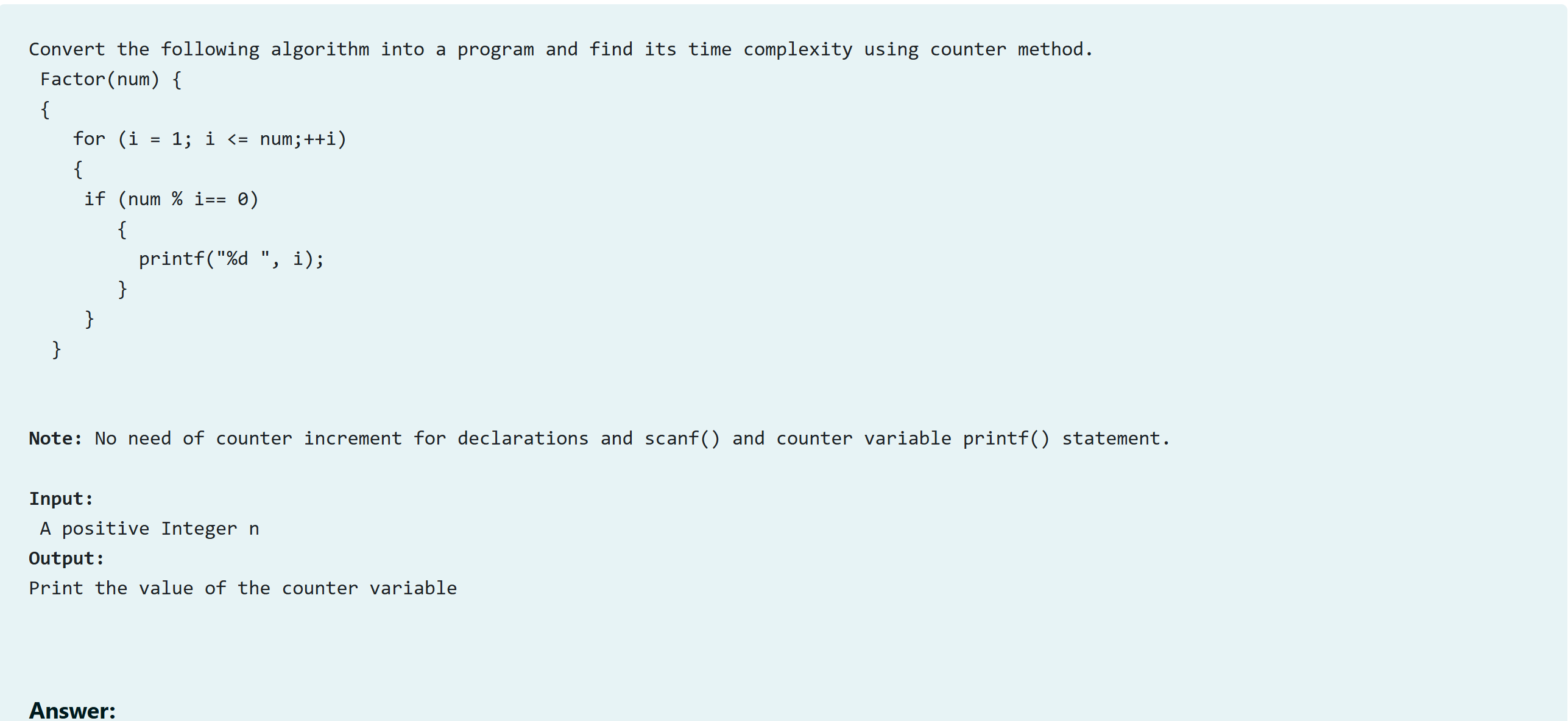
func(n);

}

Output:



## **Problem 3: Finding Complexity using Counter Method**



#include<stdio.h>

void Factor(int num)

{

int count=0;

for (int i = 1; i <= num;++i)

{

count++;

if (num % i== 0)

{

//printf("%d ", i);

count++;

}

count++;

}

count++;

printf("%d",count);

}

int main()

{

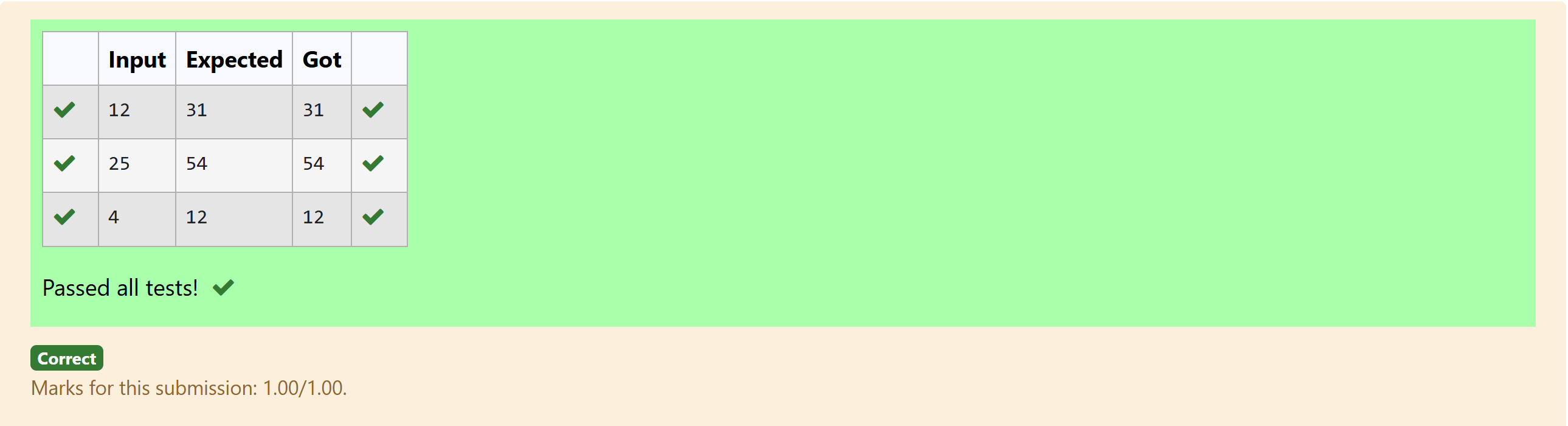
int num;

scanf("%d",&num);

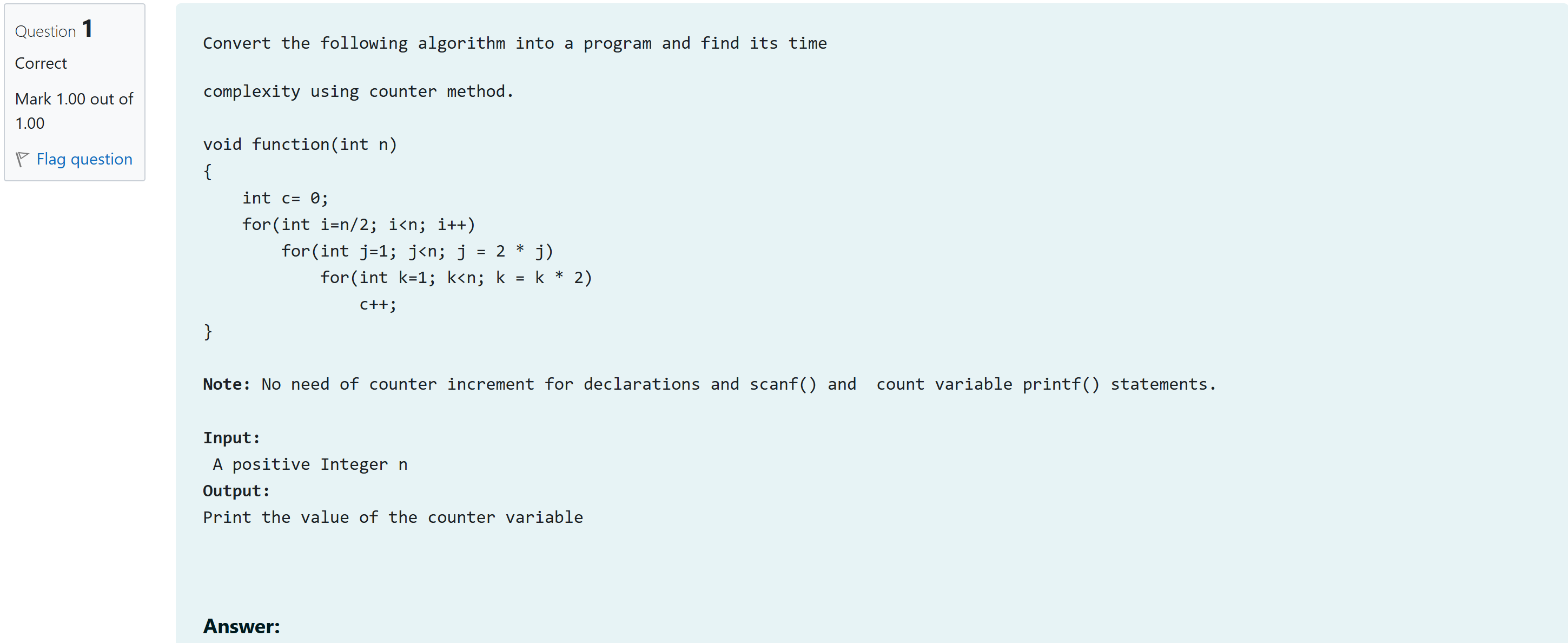
Factor(num);

}

Output:



## **Problem 4: Finding Complexity using Counter Method**



#include<stdio.h>

void function(int n)

{

int c= 0;

int count=0;

count++;

for(int i=n/2; i<n; i++)

{

count++;

for(int j=1; j<n; j = 2 \* j)

{

count++;

for(int k=1; k<n; k = k \* 2)

{

count++;

c++;

count++;

}

count++;

}

count++;

}

count++;

printf("%d",count);

}

int main()

{

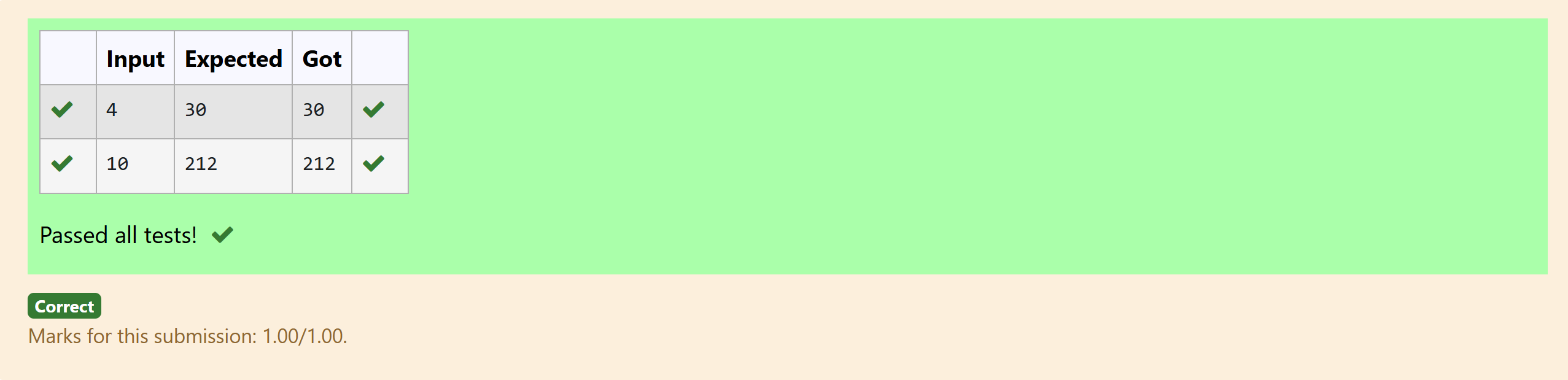
int n;

scanf("%d",&n);

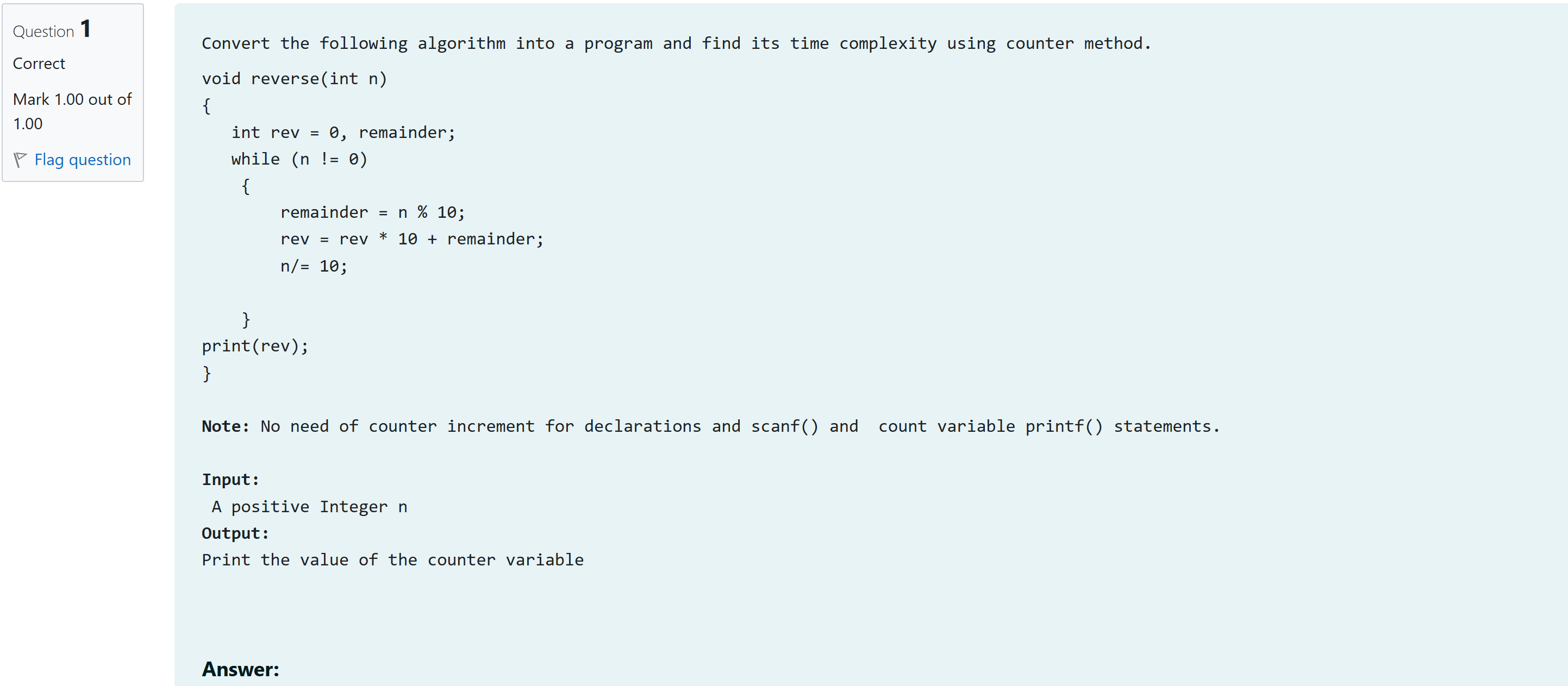
function(n);

}

Output:



## **Problem 5: Finding Complexity using Counter Method**



#include<stdio.h>

void reverse(int n)

{

int count=0;

int rev = 0, remainder;

count++;

count++;

while (n != 0)

{

count++;

remainder = n % 10;

count++;

rev = rev \* 10 + remainder;

count++;

n/= 10;

count++;

}

count++;

//print(rev);

printf("%d",count);

}

int main()

{

int n;

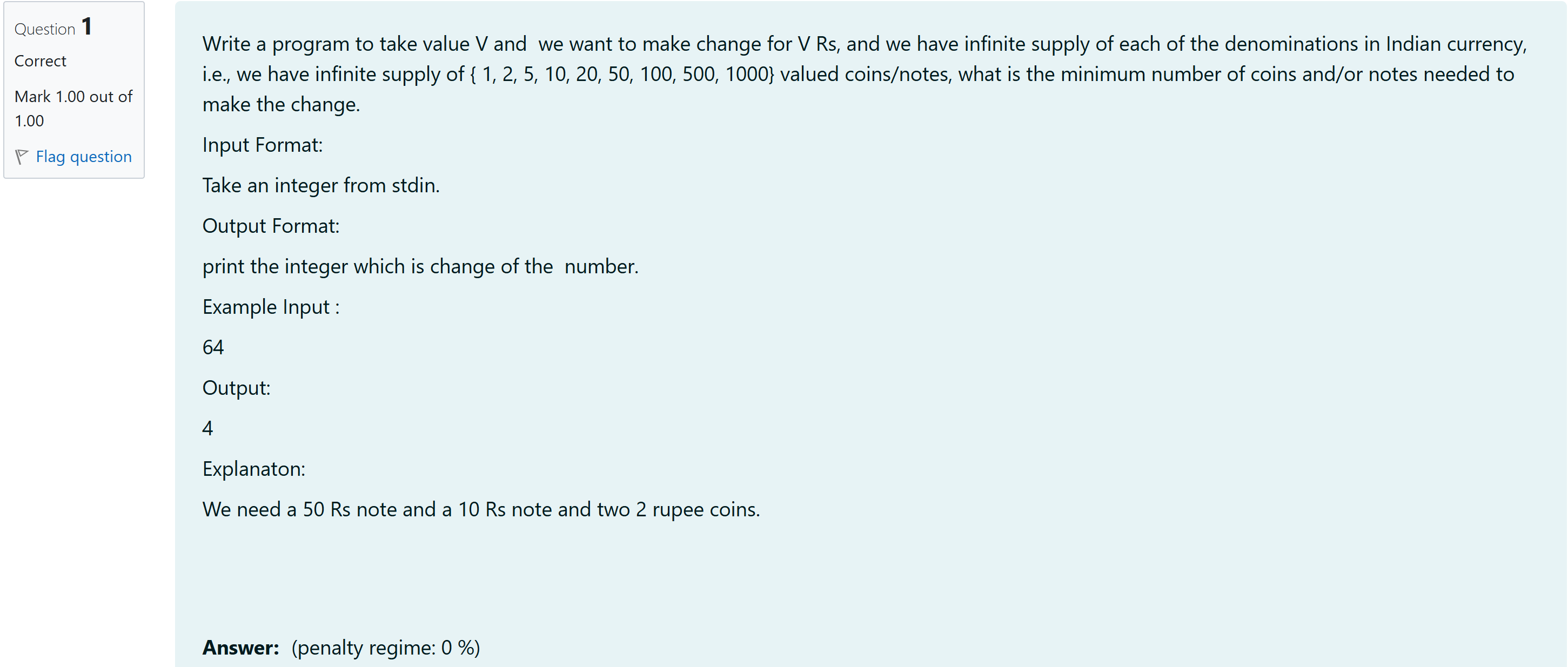
scanf("%d",&n);

reverse(n);

}

Output:

## **1-G-COIN PROBLEM**



#include<stdio.h>

int main() {

int V;

scanf("%d", &V);

int denominations[] = {1000, 500, 100, 50, 20, 10, 5, 2, 1};

int count = 0;

for (int i = 0; i < 9; i++) {

if (V >= denominations[i]) {

count += V / denominations[i];

V = V % denominations[i];

}

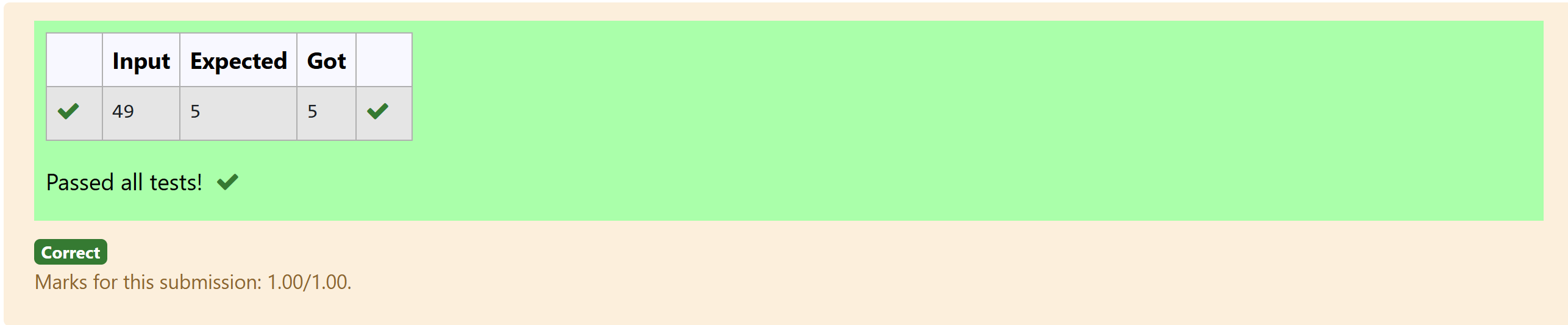
}

printf("%d\n", count);

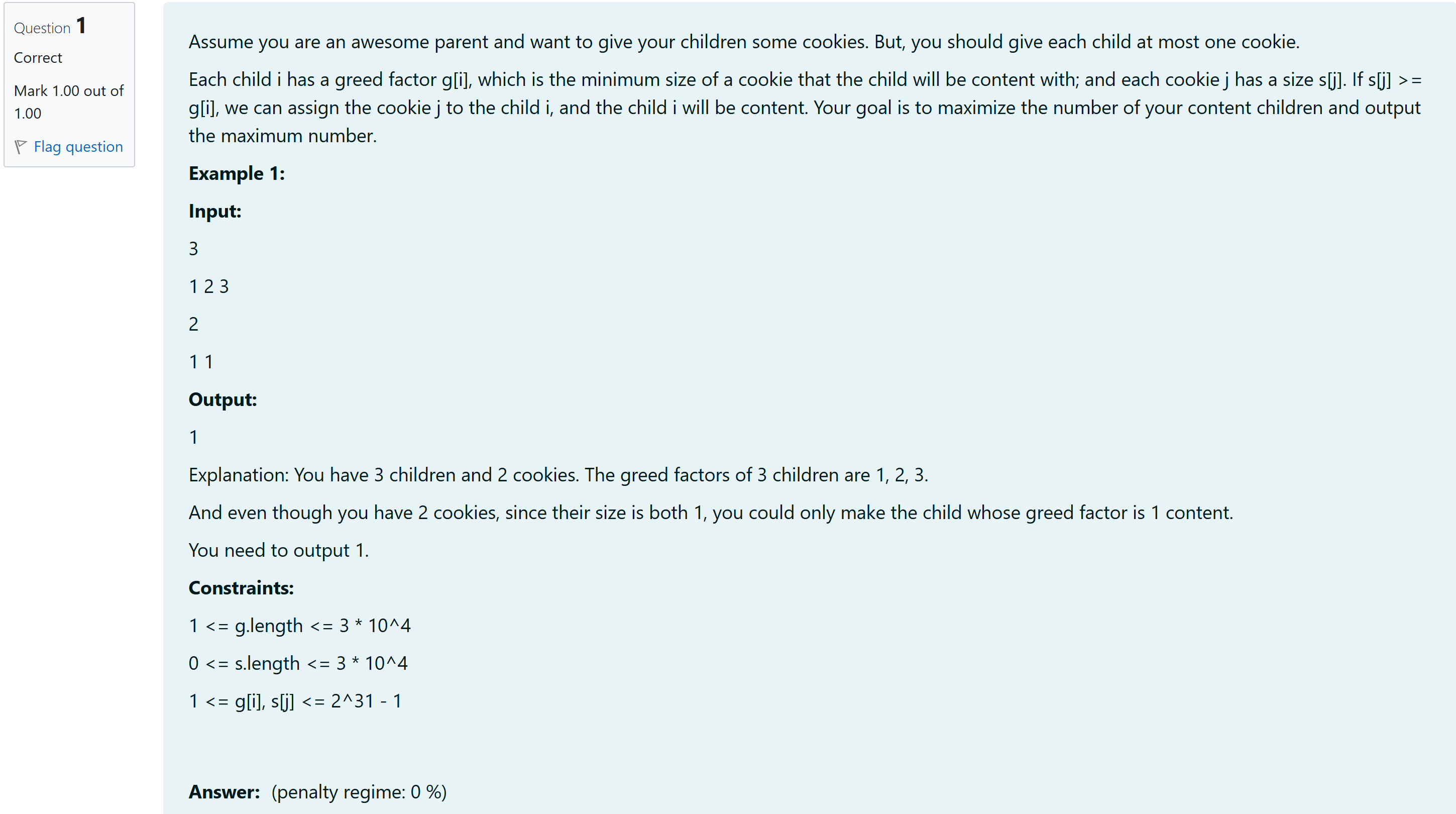
return 0;

}

Output:



**2-G-COOKIES PROBLEM**



#include<stdio.h>

int main()

{

int n;

scanf("%d",&n);

int arr[n];

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

{

scanf("%d",&arr[i]);

}

for (int i=0; i < n - 1; i++)

{

for (int j=0; j < n - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

int sum=0;

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

{

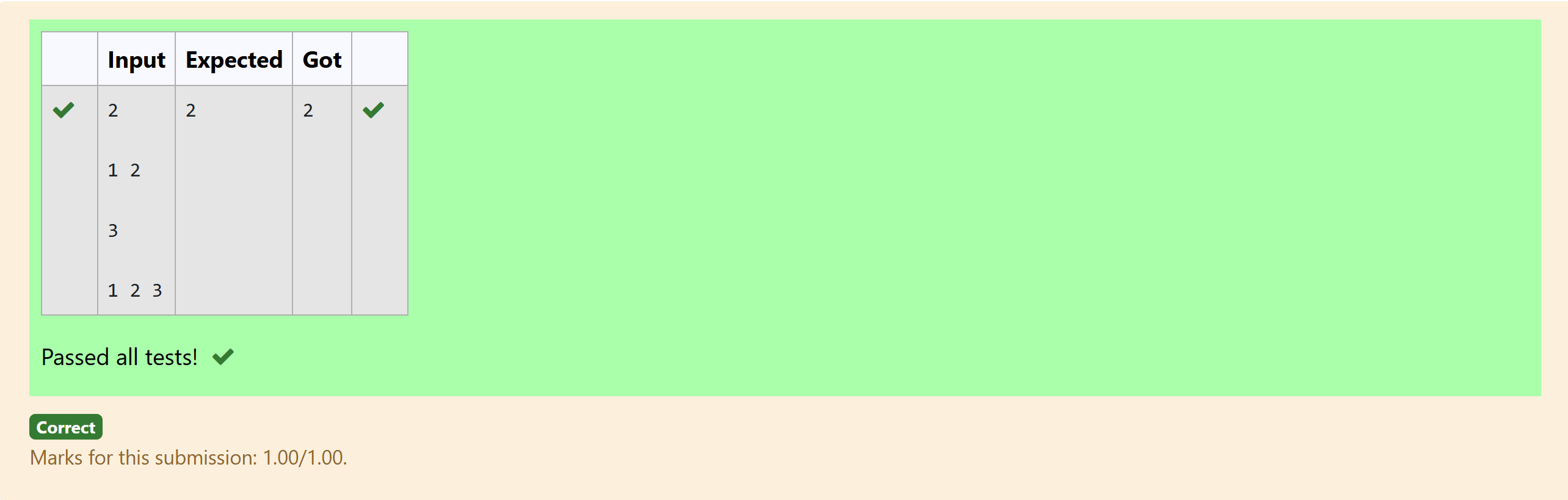
sum+=arr[i]\*i;

}

printf("%d",sum);

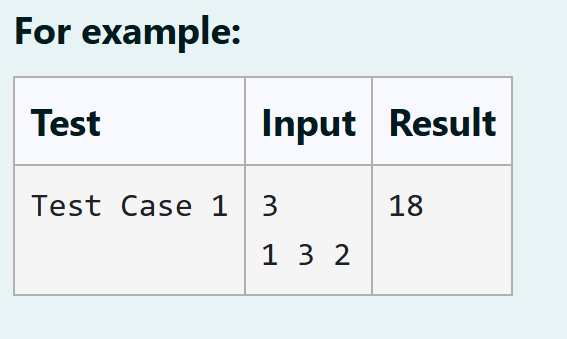
}

Output:



**3-G-BURGER PROBLEM**

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories.   
If he has eaten *i* burgers with c calories each, then he has to run at least *3i \* c* kilometers to burn out the calories. For example, if he ate 3  
 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are (30 \* 1) + (31 \* 3) + (32 \* 2) = 1 + 9 + 18 = 28.  
 But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance  
 he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve the problem.  
**Input Format**  
First Line contains the number of burgers  
Second line contains calories of each burger which is n space-separate integers   
   
**Output Format**  
  
Print: Minimum number of kilometers needed to run to burn out the calories  
   
**Sample Input**  
   
3  
5 10 7  
  
**Sample Output**  
76



#include<stdio.h>

#include<math.h>

int main()

{

int n;

scanf("%d",&n);

int a[n];

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

scanf("%d",&a[i]);

for (int i = 0; i < n-1; i++) {

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

int t;

if (a[i] < a[j]) {

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

}

int sum=0,h;

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

{

h=pow(n,i);

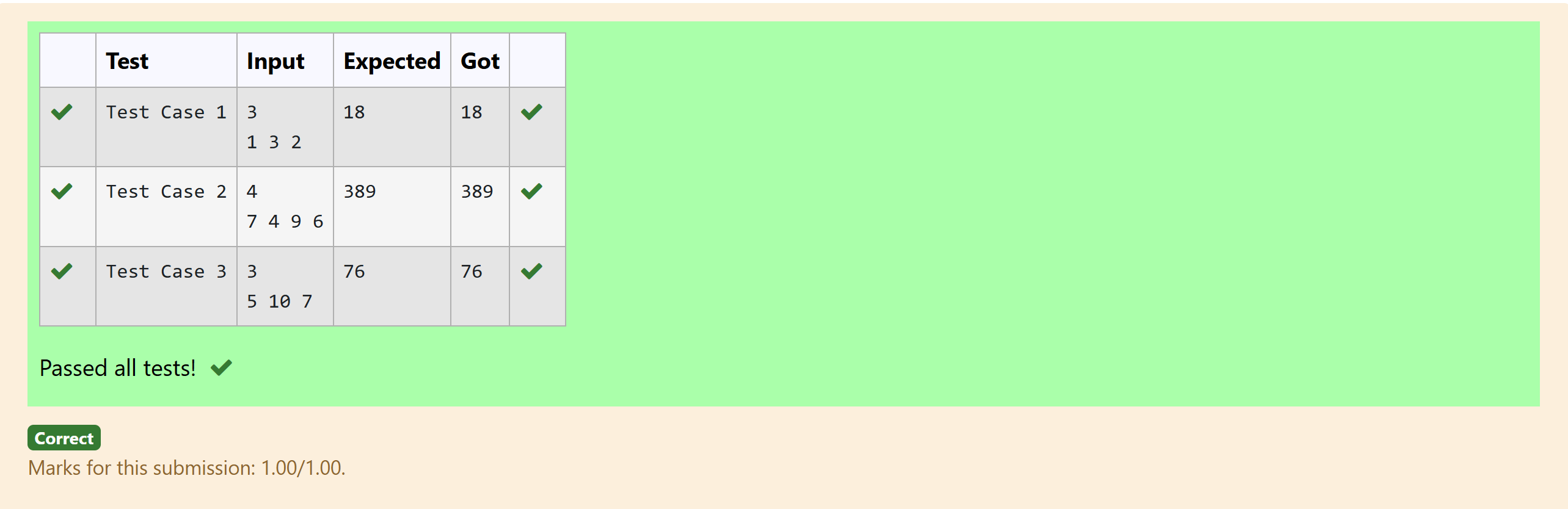
sum+=h\*a[i];

}

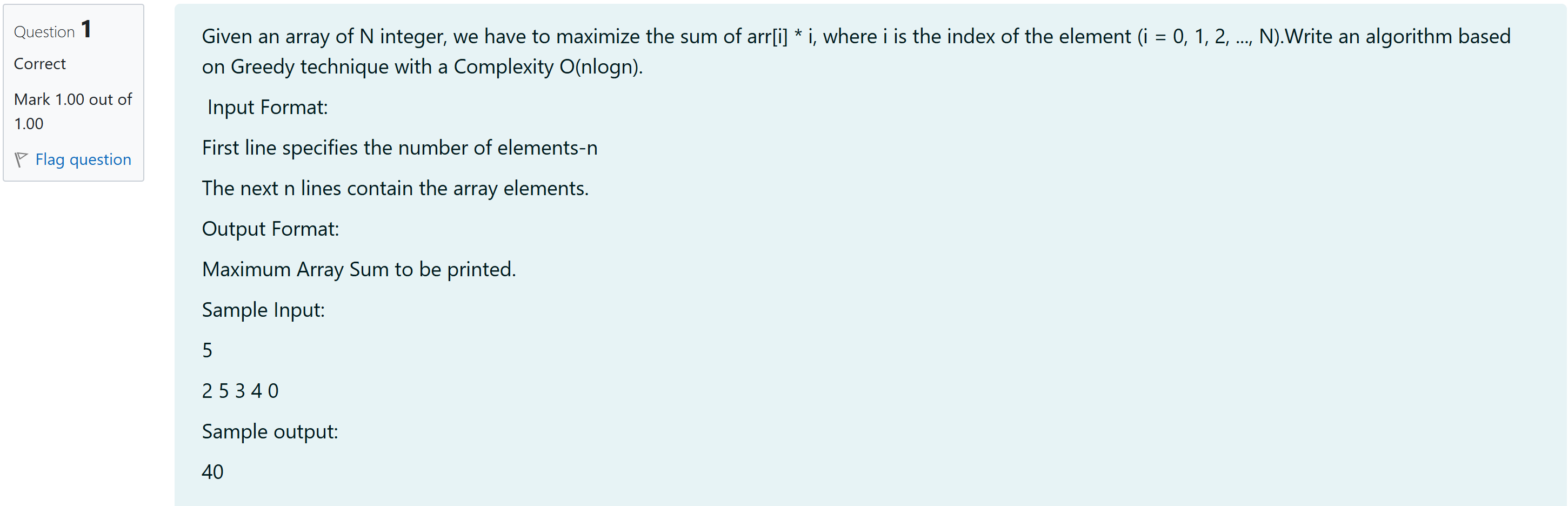
printf("%d",sum);

}

Output:



**4-G-Array Sum max problem**



#include<stdio.h>

int main(){

int n;

scanf("%d", &n);

int a[n];

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

scanf("%d",&a[i]);

}

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

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

int t;

if(a[i]>a[j]){

t=a[j];

a[j]=a[i];

a[i]=t;

}

}

}

int sum=0;

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

sum+=a[i]\*i;

}

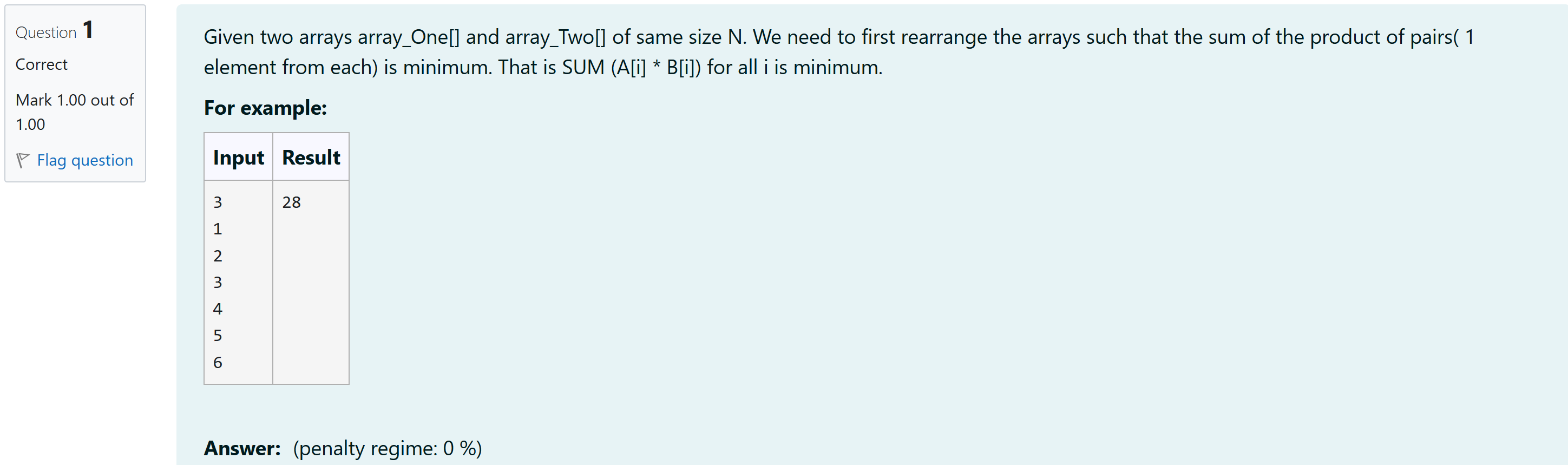
printf("%d",sum);

}

Output:

## 

## **1-Number of Zeros in a Given Array**



#include <stdio.h>

int main() {

int N;

scanf("%d", &N);

int array\_One[N];

int array\_Two[N];

int temp;

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

scanf("%d", &array\_One[i]);

}

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

scanf("%d", &array\_Two[i]);

}

for (int i = 0; i < N - 1; i++) {

for (int j = i + 1; j < N; j++) {

if (array\_One[i] > array\_One[j]) {

temp = array\_One[i];

array\_One[i] = array\_One[j];

array\_One[j] = temp;

}

}

}

for (int i = 0; i < N - 1; i++) {

for (int j = i + 1; j < N; j++) {

if (array\_Two[i] > array\_Two[j]) {

temp = array\_Two[i];

array\_Two[i] = array\_Two[j];

array\_Two[j] = temp;

}

}

}

for (int i = 0; i < N / 2; i++) {

temp = array\_Two[i];

array\_Two[i] = array\_Two[N - i - 1];

array\_Two[N - i - 1] = temp;

}

int min\_sum = 0;

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

min\_sum += array\_One[i] \* array\_Two[i];

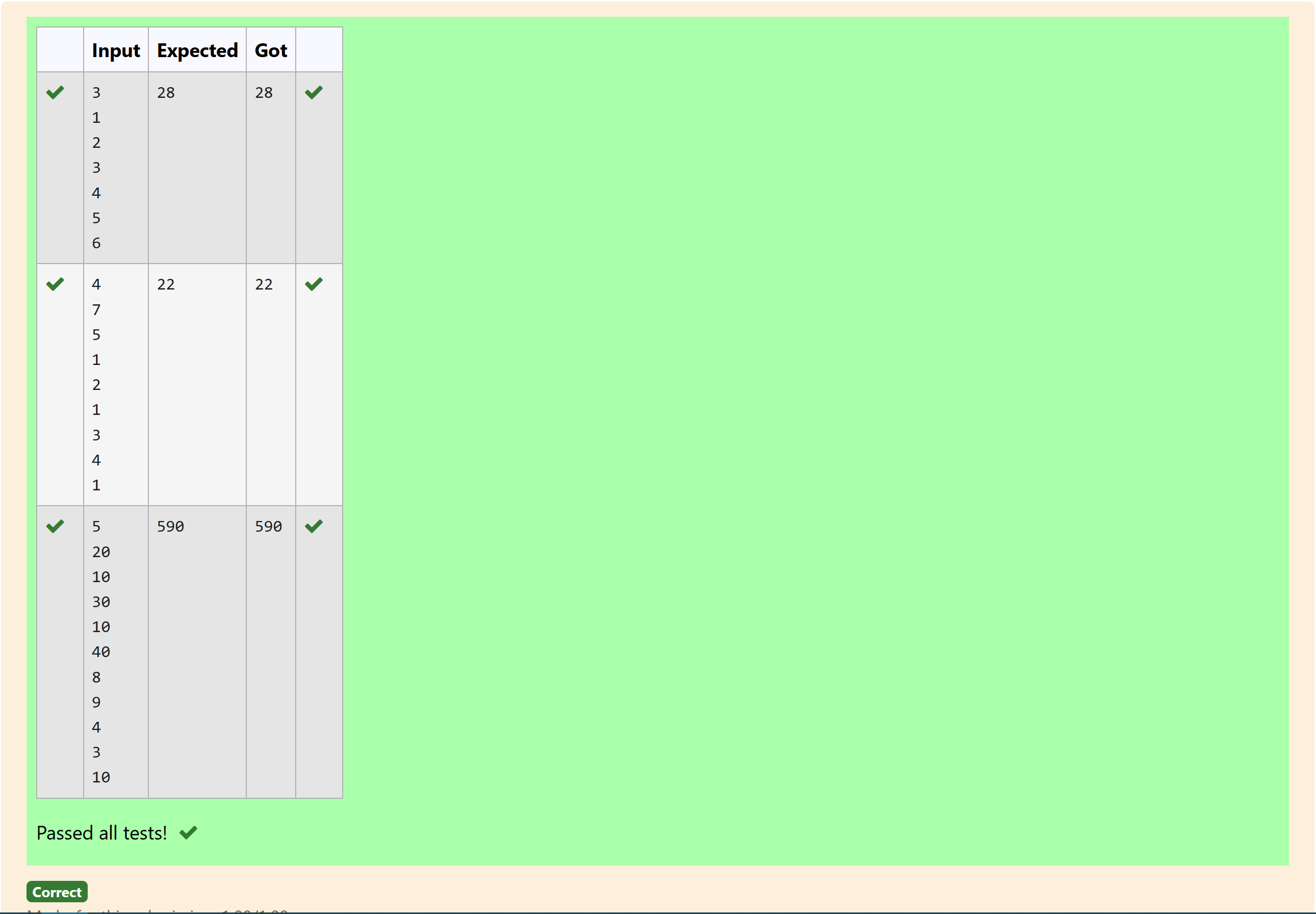
}

printf("%d\n", min\_sum);

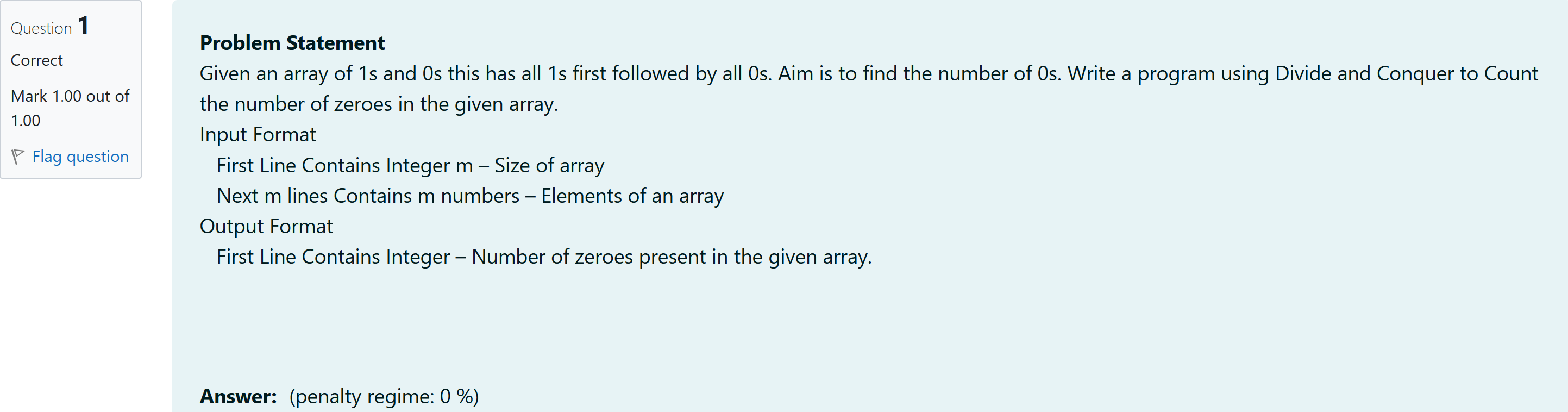
return 0;

}

Output:



## **2-Majority Element**

#include<stdio.h>

int main()

{

int n,temp1=-1,temp2=-1;

scanf("%d",&n);

int arr[n];

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

{

scanf("%d",&arr[i]);

}

int x=(n-1)/2;

for(int i=0;i<x;i++)

{

if(arr[i]==0)

{

temp1=i;

break;

}

else if(arr[x-i]!=0 && arr[x+i]==0)

{

temp2=x+i;

break;

}

}

if(temp1!=-1)

{

printf("%d",n-temp1);

}

else if(temp2!=-1)

{

printf("%d",n-temp2);

}

else

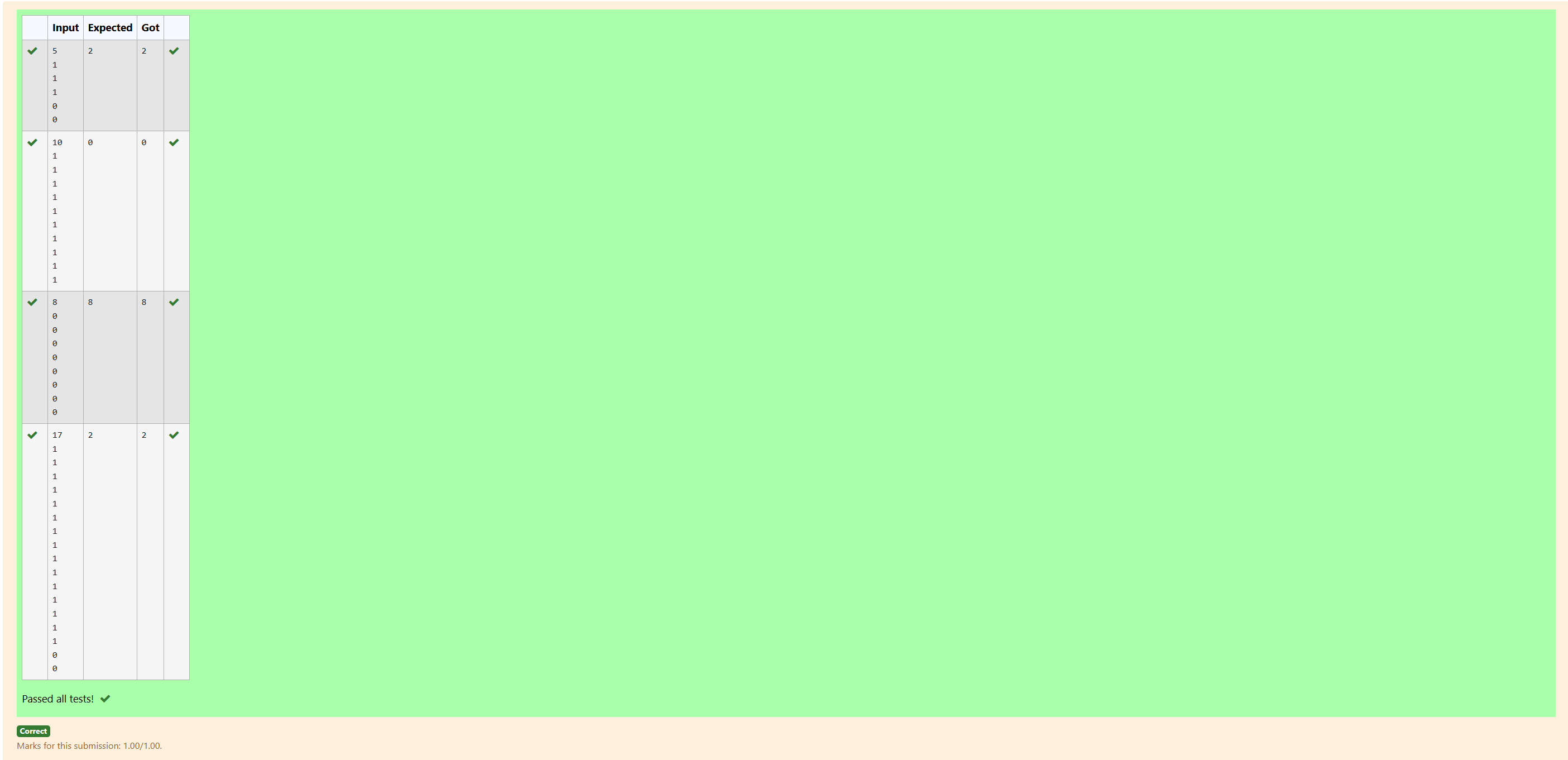
{

printf("%d",temp1+1);

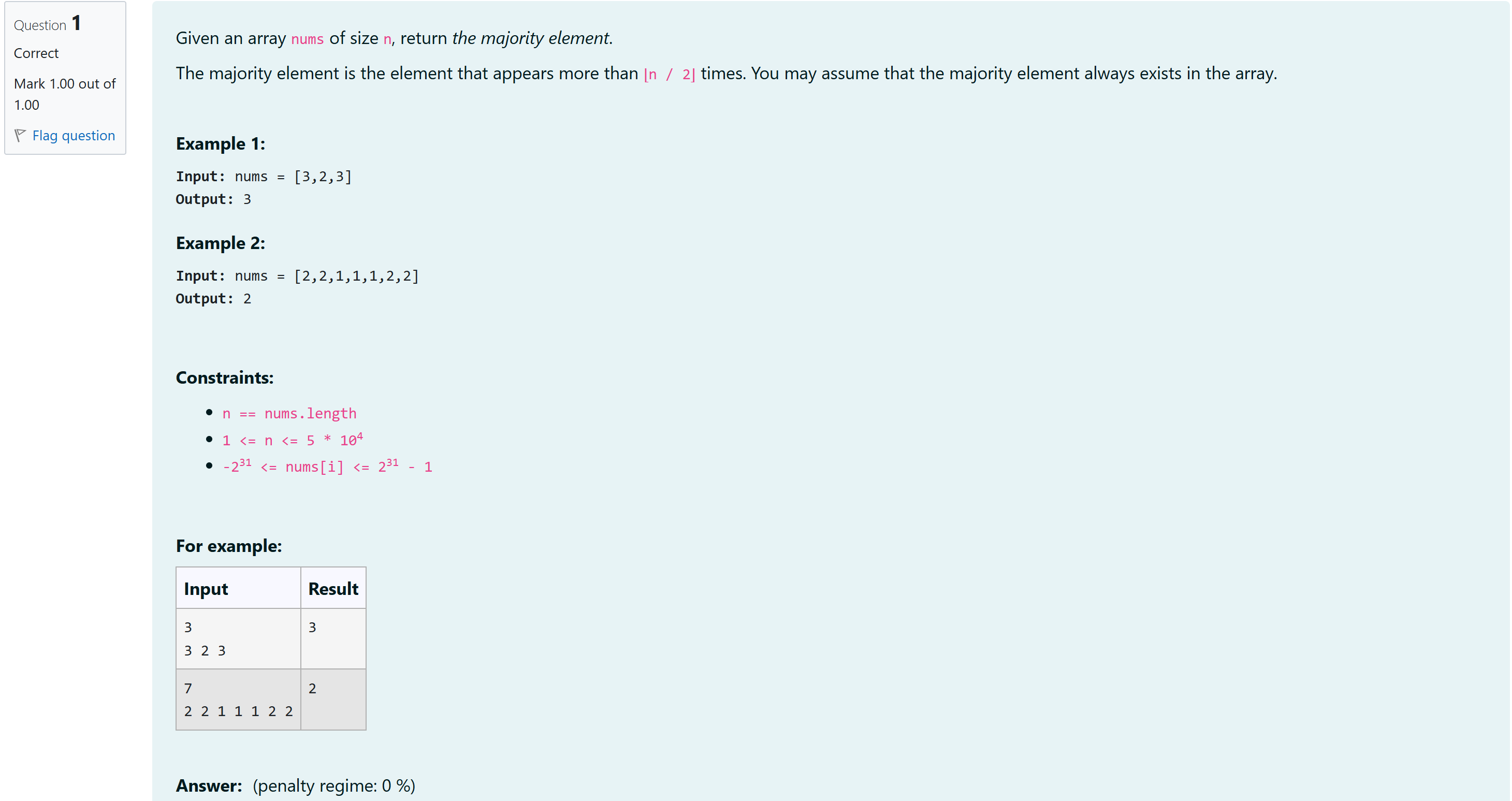
}

}

Output:



**3-Finding Floor Value**



#include <stdio.h>

#include <stdlib.h>

int Majority(int nums[], int size)

{

int count = 0;

int candidate = nums[0];

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

{

if (count == 0)

{

candidate = nums[i];

}

if (nums[i] == candidate)

{

count++;

}

else

{

count--;

}

}

count = 0;

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

{

if (nums[i] == candidate)

{

count++;

}

}

if (count > size / 2)

{

return candidate;

}

return -1;

}

int main()

{

int n;

scanf("%d", &n);

int \*nums = (int \*)malloc(n \* sizeof(int));

if (nums == NULL)

{

return 1;

}

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

{

scanf("%d", &nums[i]);

}

int mElement = Majority(nums, n);

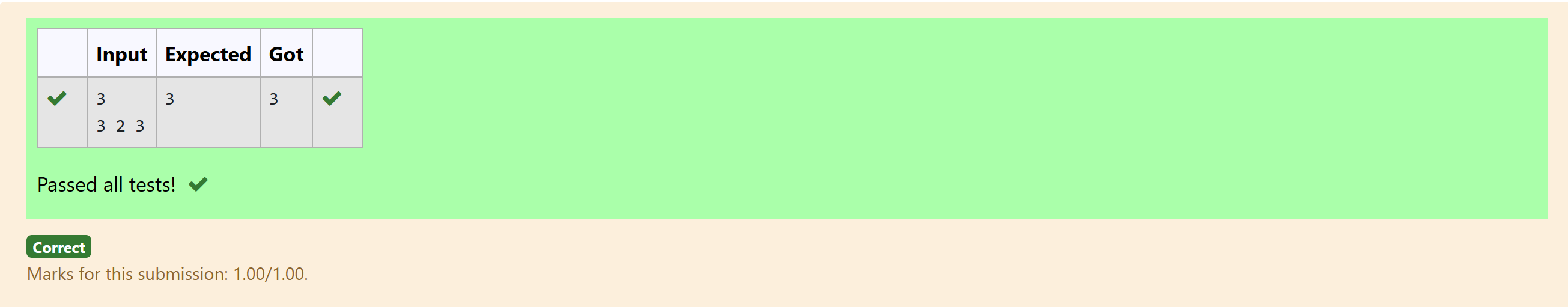
printf("%d\n", mElement);

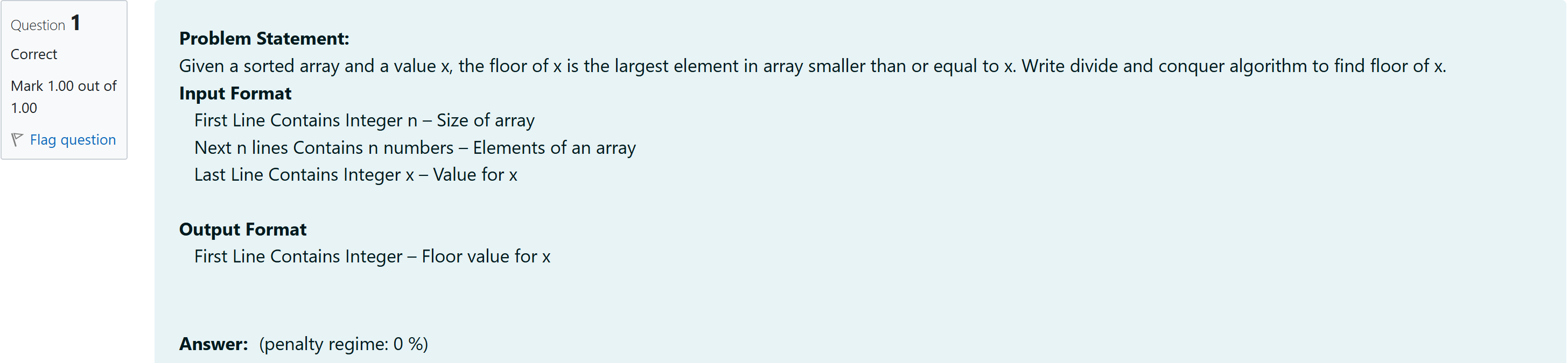
free(nums);

return 0;

}

Output:





#include <stdio.h>

int findFloor(int arr[], int low, int high, int x) {

if (low > high) {

return -1;

}

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

return arr[mid];

} else if (arr[mid] < x) {

int floorValue = findFloor(arr, mid + 1, high, x);

return (floorValue != -1) ? floorValue : arr[mid];

} else {

return findFloor(arr, low, mid - 1, x);

}

}

int main() {

int n, x;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

scanf("%d", &x);

int result = findFloor(arr, 0, n - 1, x);

if (result != -1) {

printf("%d\n", result);

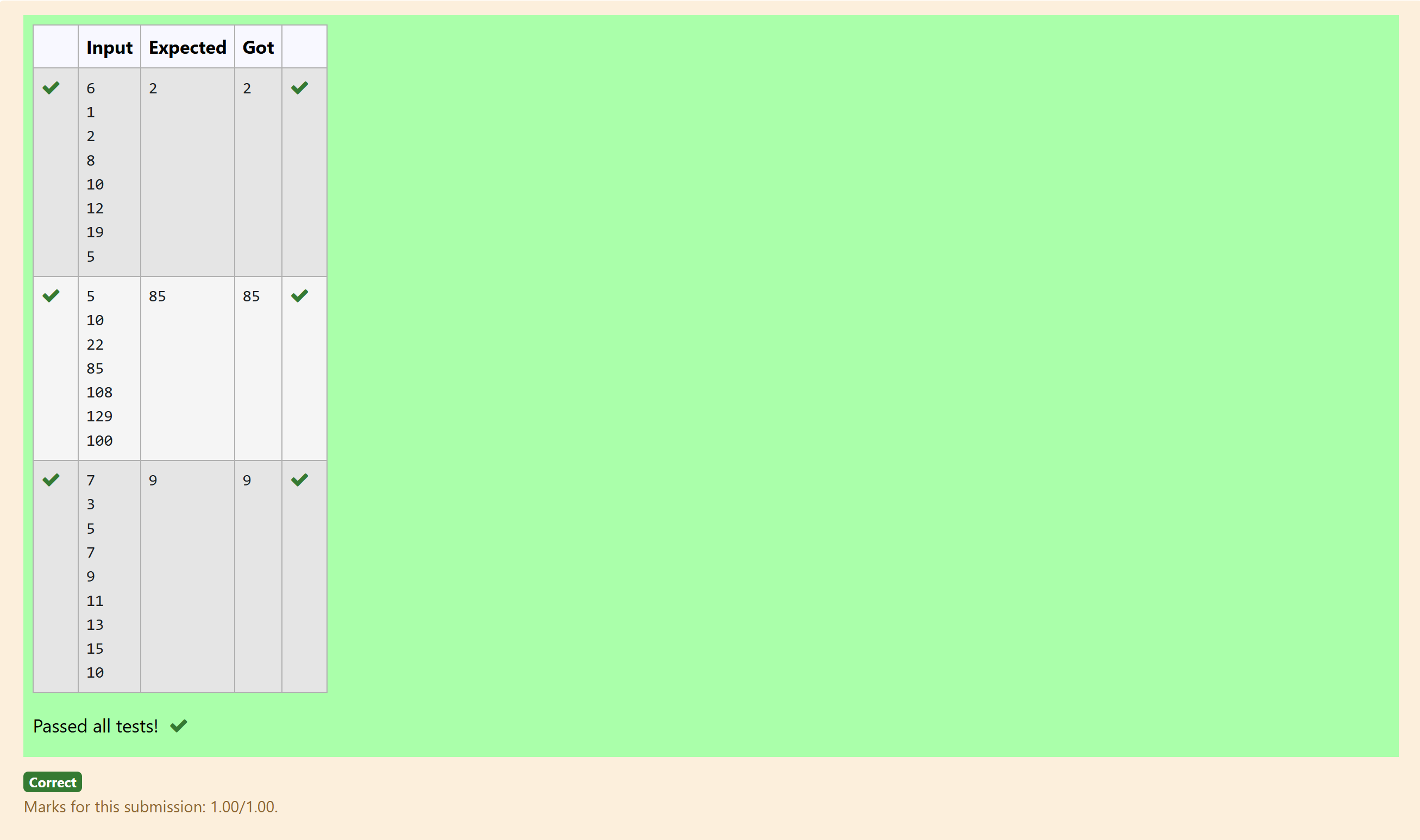
} else {

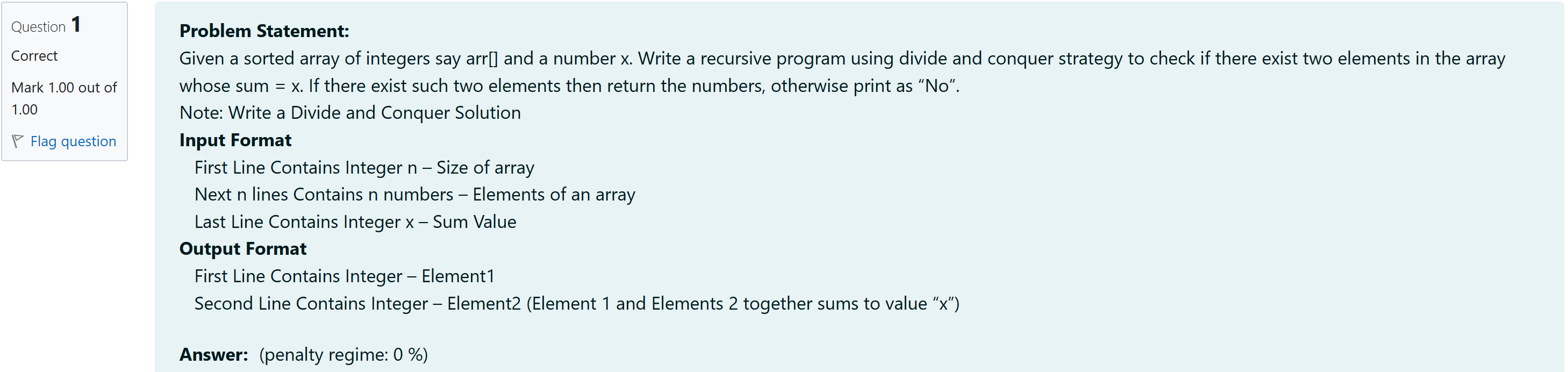
printf("No floor value found\n");

}

return 0;

}





#include <stdio.h>

int findPair(int arr[], int low, int high, int x) {

int left = low;

int right = high;

while (left < right) {

int sum = arr[left] + arr[right];

if (sum == x) {

printf("%d\n", arr[left]);

printf("%d\n", arr[right]);

return 1;

} else if (sum < x) {

left++;

} else {

right--;

}

}

return 0;

}

int main() {

int n, x;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

scanf("%d", &x);

if (!findPair(arr, 0, n - 1, x)) {

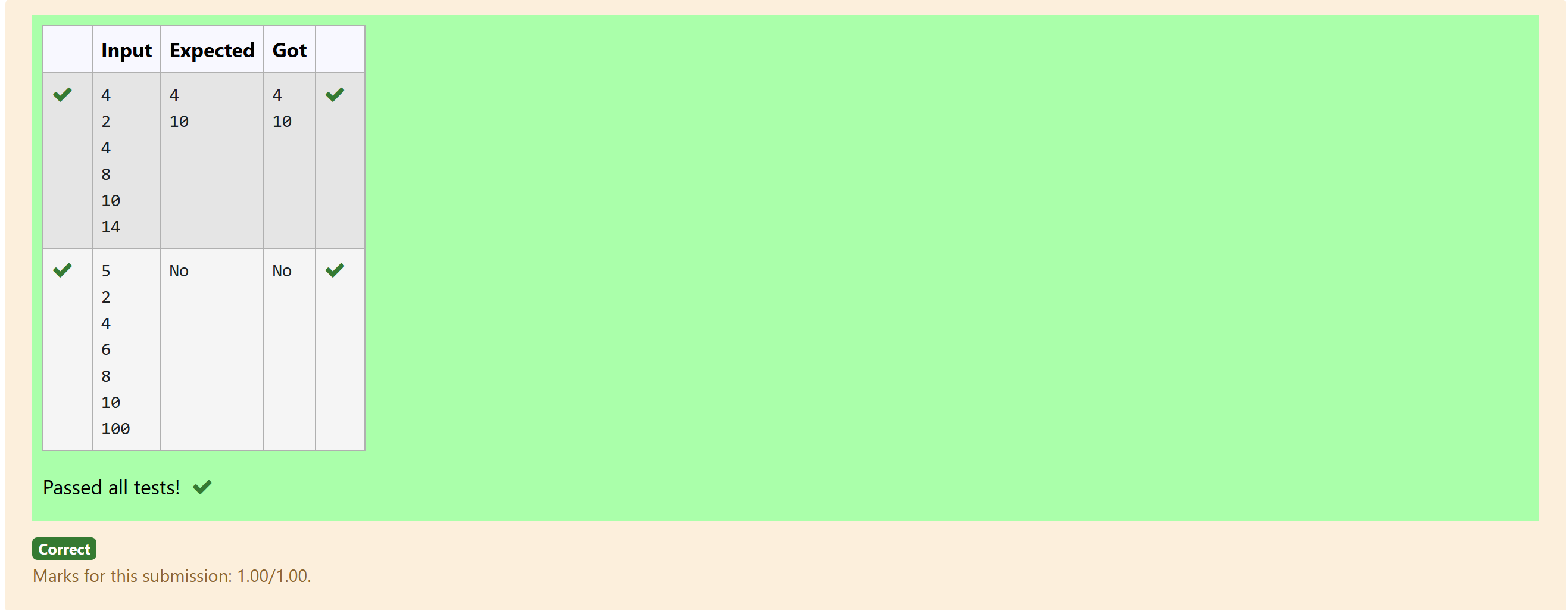
printf("No\n");

}

return 0;

}

Output:





#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return i + 1;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int n;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

quickSort(arr, 0, n - 1);

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

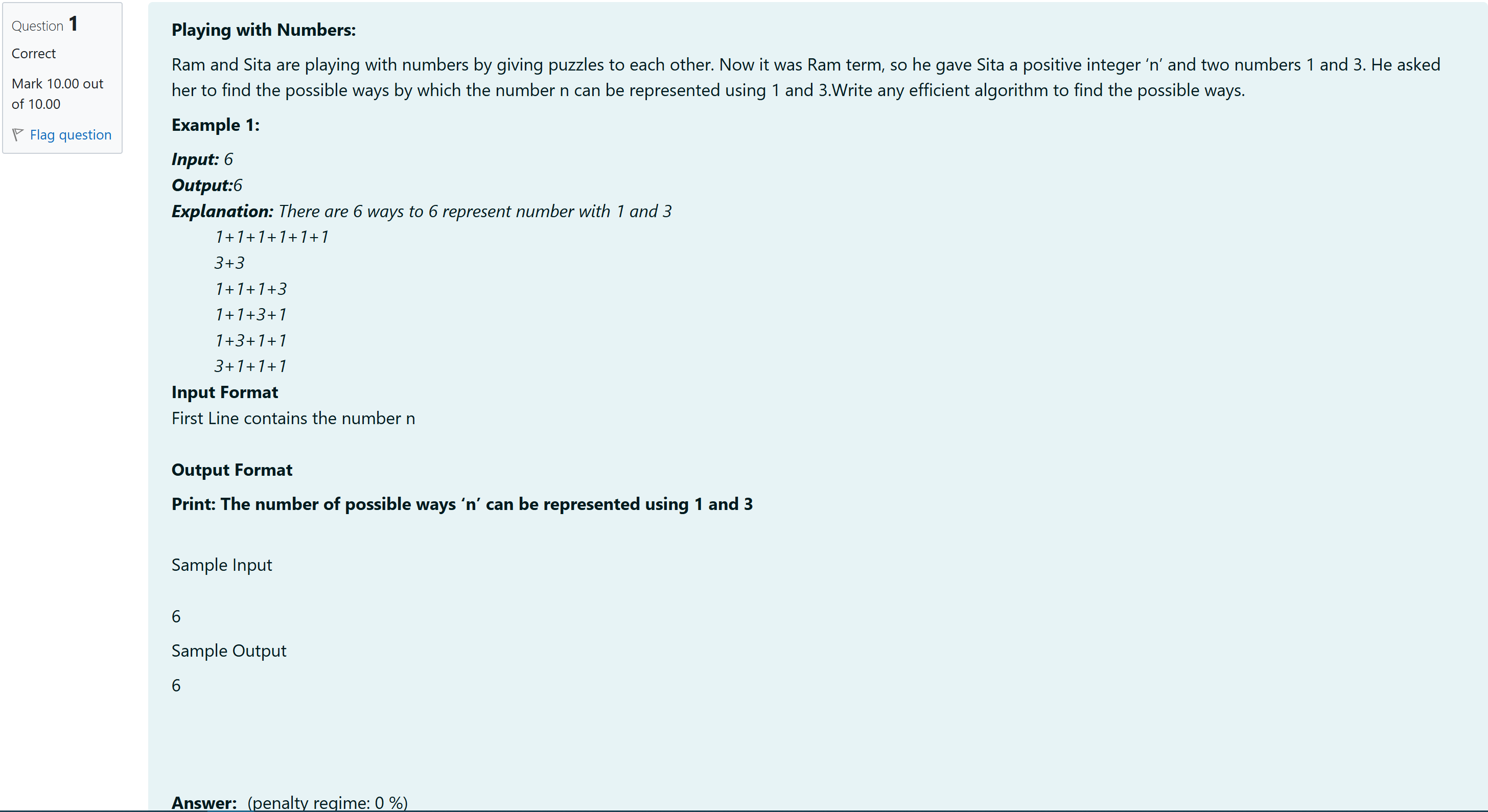
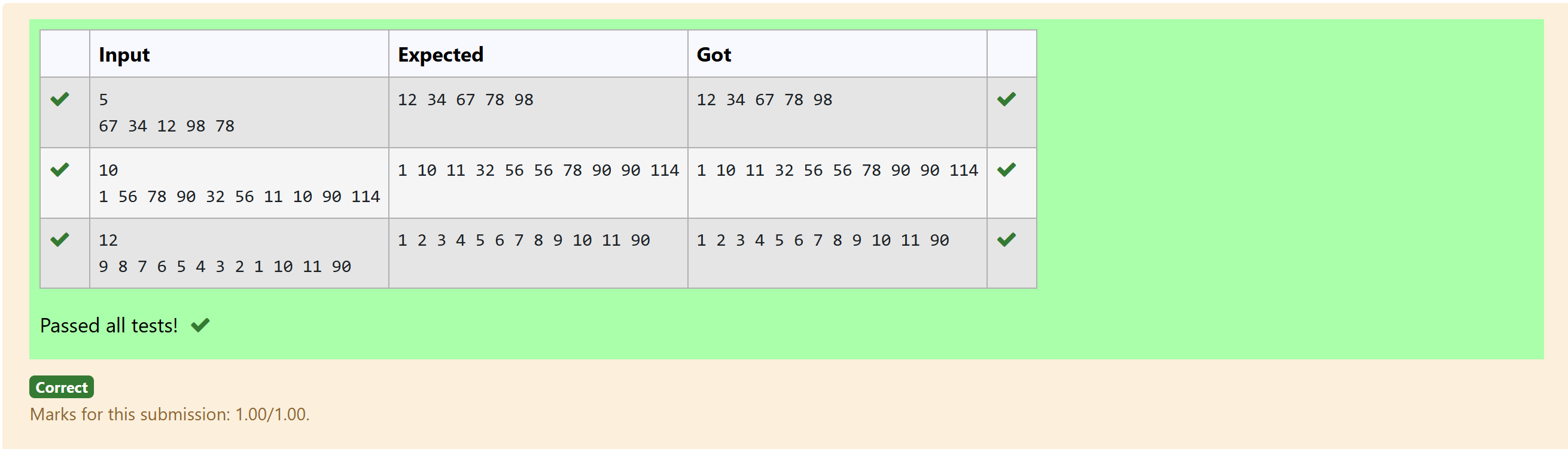
printf("%d ", arr[i]);

}

return 0;

}

Output:



#include <stdio.h>

long long int count\_ways\_to\_sum(int n)

{

if (n == 0) return 1;

if (n == 1) return 1;

if (n == 2) return 1;

long long int dp[n + 1];

dp[0] = 1;

dp[1] = 1;

dp[2] = 1;

for (int i = 3; i <= n; i++)

{

dp[i] = dp[i - 1] + dp[i - 3];

}

return dp[n];

}

int main()

{

int n;

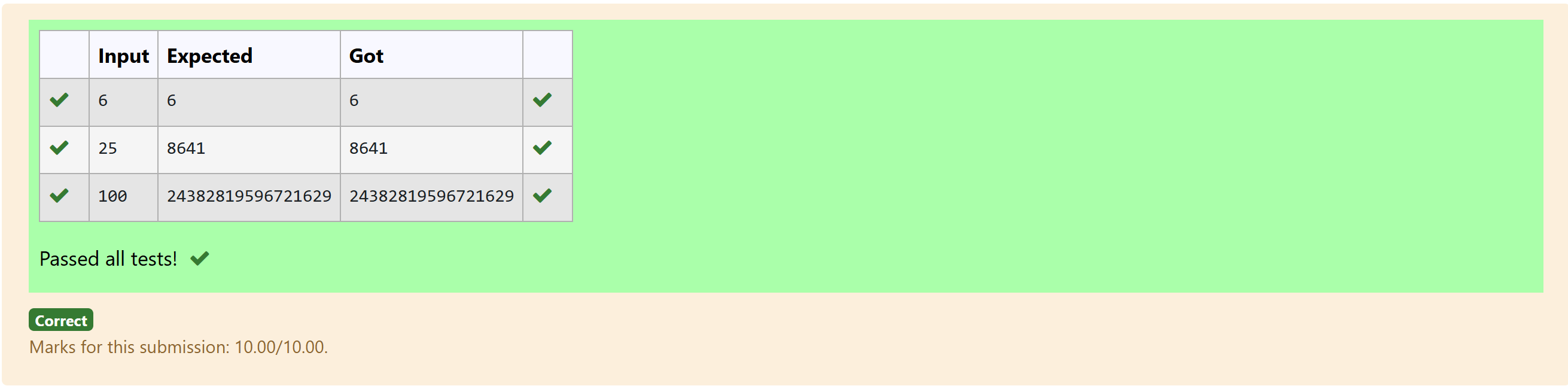
scanf("%d", &n);

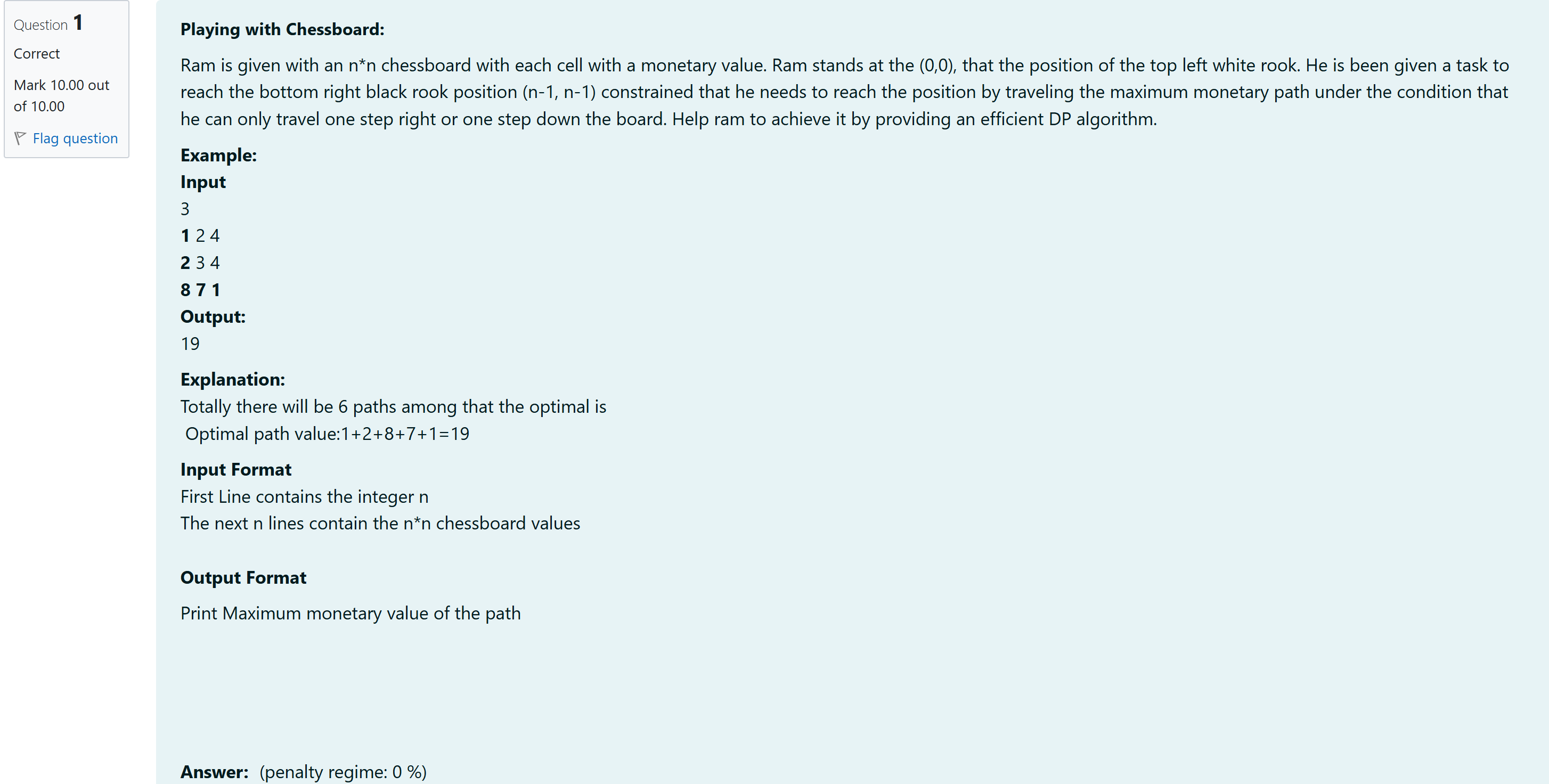
printf("%lld\n", count\_ways\_to\_sum(n));

return 0;

}

Output:





#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int maxMonetaryPath(int n, int chessboard[n][n]) {

int dp[n][n];

// Initialize the starting point

dp[0][0] = chessboard[0][0];

// Fill the first row

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

dp[0][j] = dp[0][j - 1] + chessboard[0][j];

}

// Fill the first column

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

dp[i][0] = dp[i - 1][0] + chessboard[i][0];

}

// Fill the rest of the dp array

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

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

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + chessboard[i][j];

}

}

// The result is in the bottom-right corner

return dp[n - 1][n - 1];

}

int main() {

int n;

//printf("Enter the size of the chessboard (n): ");

scanf("%d", &n);

int chessboard[n][n];

//printf("Enter the values of the chessboard:\n");

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

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

scanf("%d", &chessboard[i][j]);

}

}

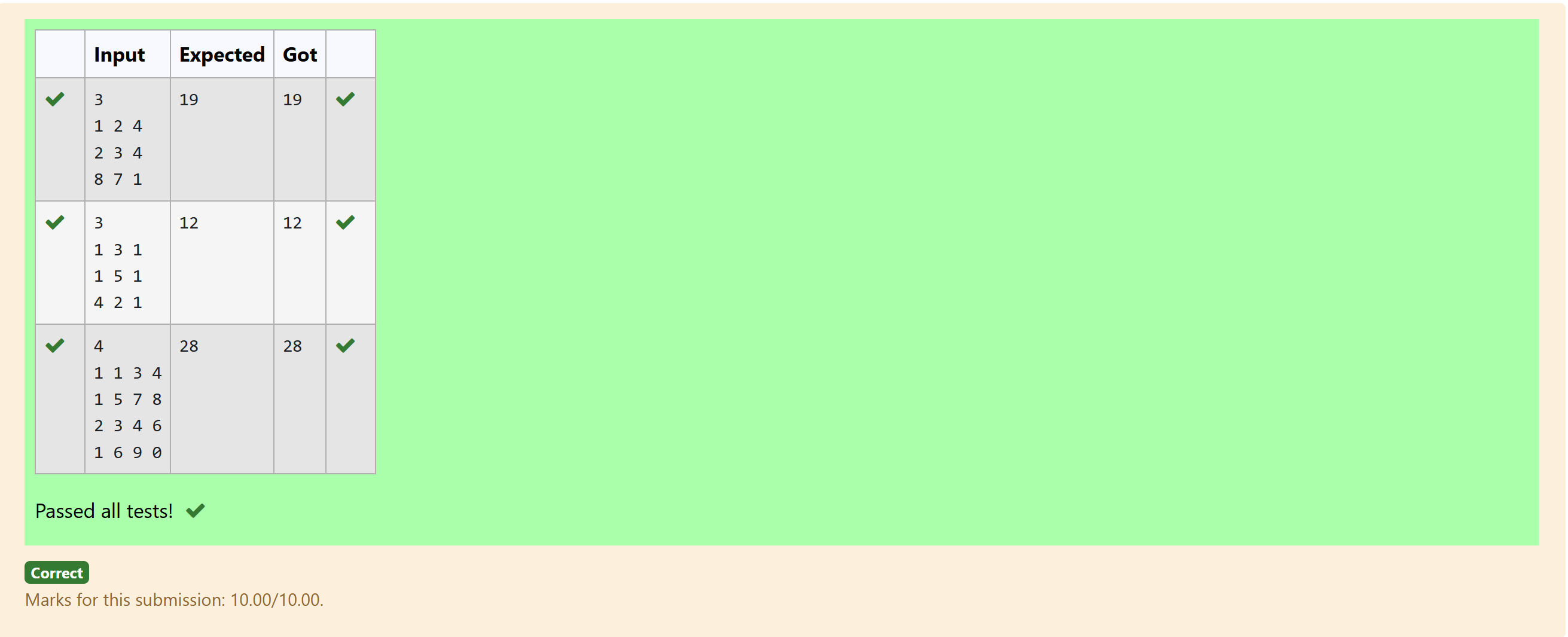
int result = maxMonetaryPath(n, chessboard);

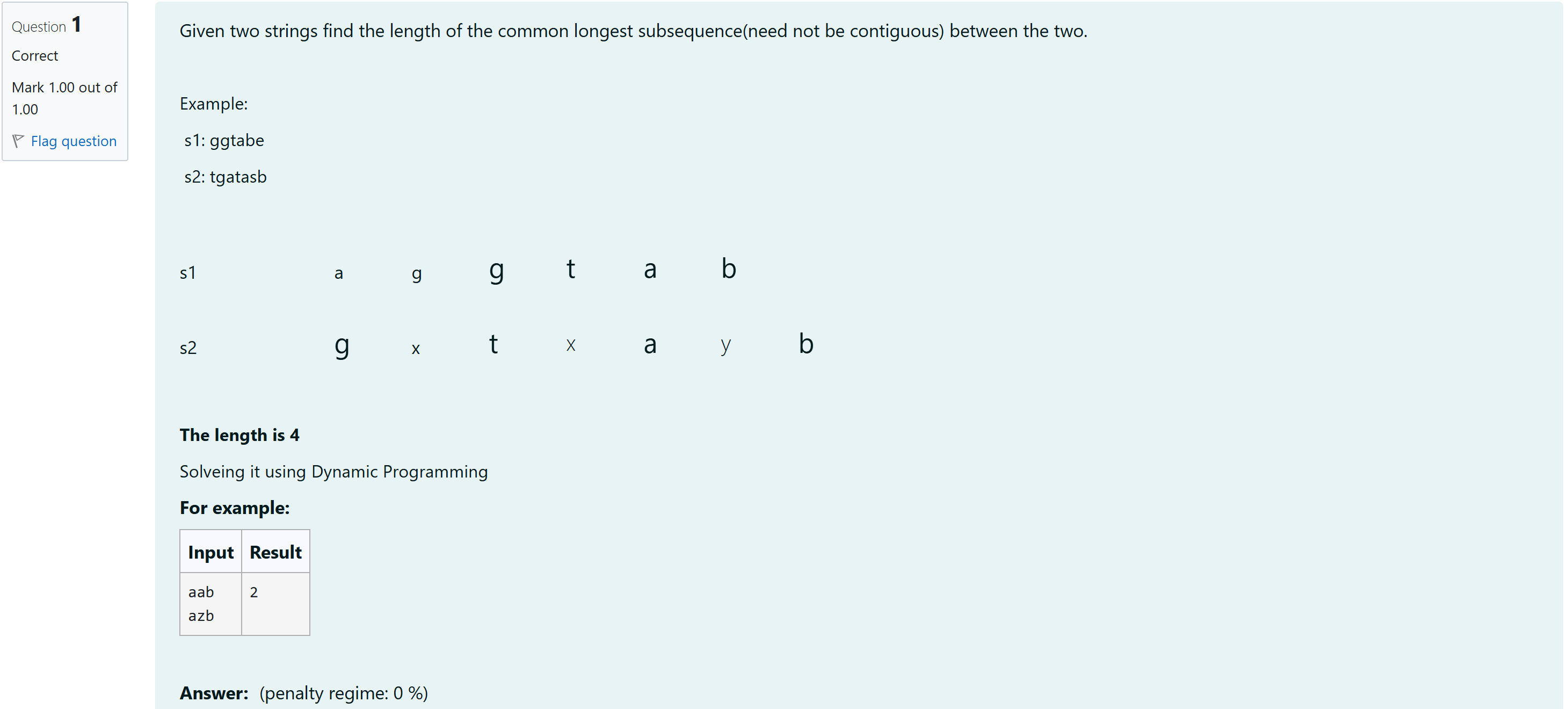
printf("%d\n", result);

return 0;

}

Output:





#include <stdio.h>

#include <string.h>

int longestCommonSubsequence(char \*s1, char \*s2) {

int len1 = strlen(s1);

int len2 = strlen(s2);

int dp[len1 + 1][len2 + 1];

for (int i = 0; i <= len1; i++) {

for (int j = 0; j <= len2; j++) {

if (i == 0 || j == 0) {

dp[i][j] = 0;

} else if (s1[i - 1] == s2[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];

}

}

}

return dp[len1][len2];

}

int main() {

char s1[100], s2[100];

scanf("%s", s1);

scanf("%s", s2);

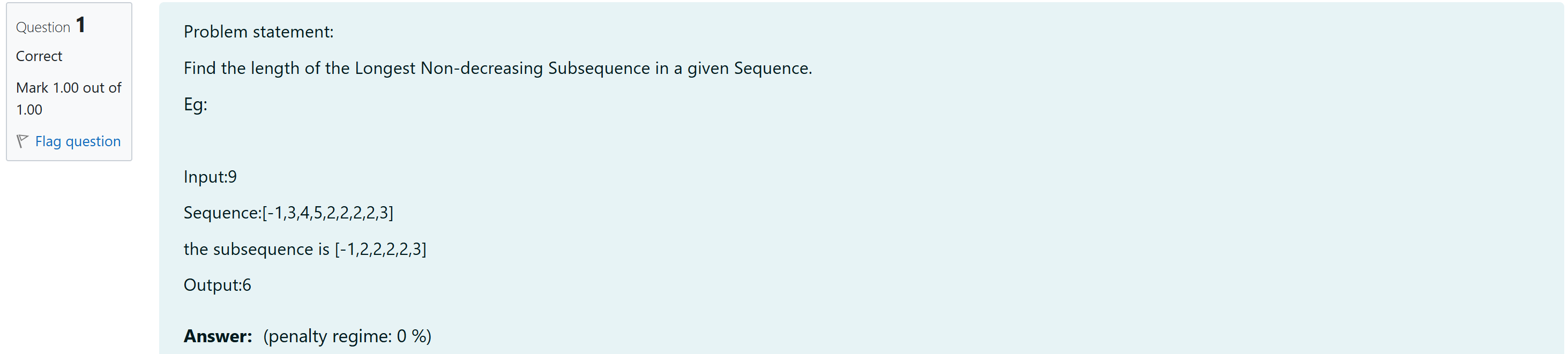
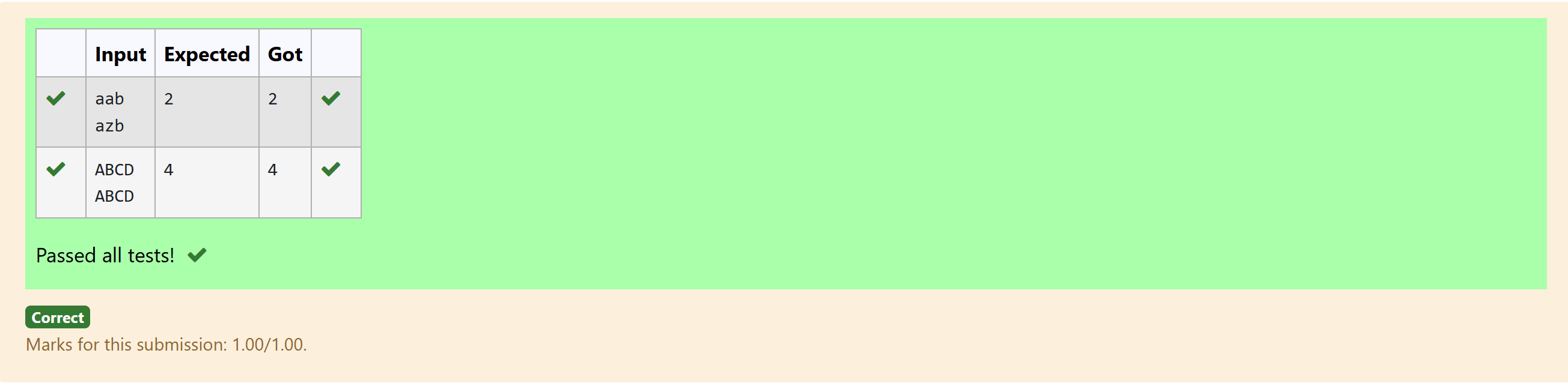
int result = longestCommonSubsequence(s1, s2);

printf("%d\n", result);

return 0;

}

Output:



#include<stdio.h>

int longestNonDecreasingSubsequence(int arr[],int n){

int dp[n];

int maxLength = 1;

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

{

dp[i] = 1;

}

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

{

for (int j = 0; j < i; j++)

{

if (arr[j] <= arr[i])

{

dp[i] = dp[i] > dp[j] + 1 ? dp[i] : dp[j] + 1;

}

}

if (dp[i] > maxLength)

{

maxLength = dp[i];

}

}

return maxLength;

}

int main()

{

int n;

scanf("%d", &n);

int arr[n];

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

{

scanf("%d", &arr[i]);

}

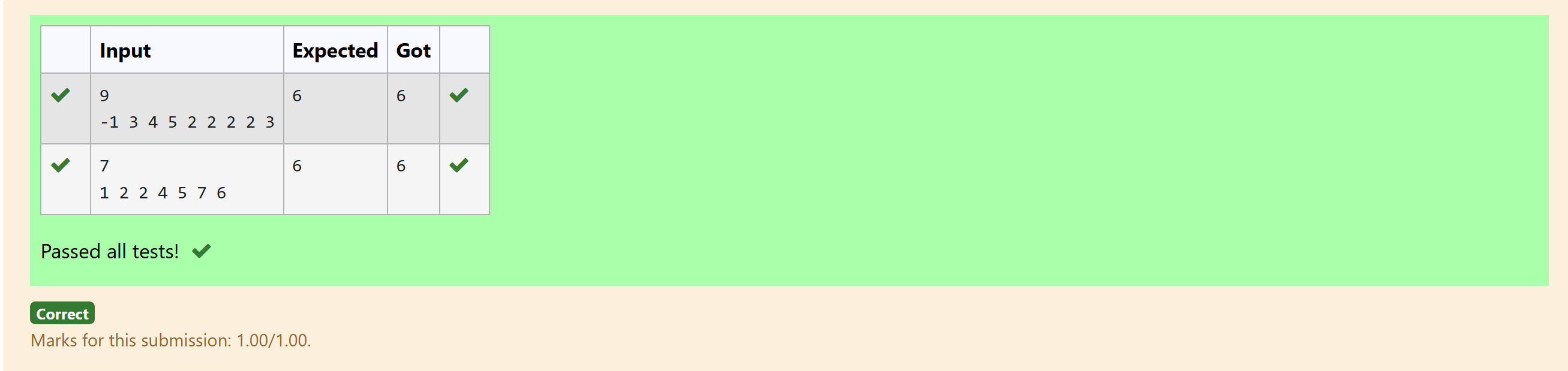
int result = longestNonDecreasingSubsequence(arr, n);

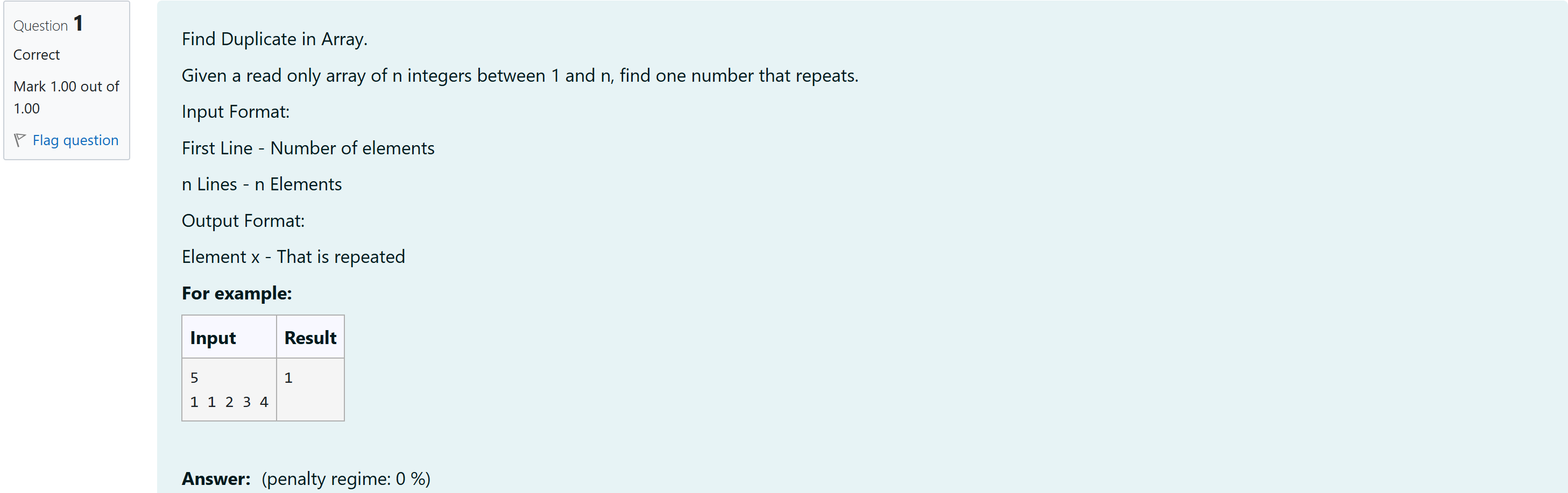
printf("%d\n", result);

return 0;

}

Output:





#include <stdio.h>

#include <stdbool.h>

int main()

{

int n;

scanf("%d", &n);

int arr[n];

bool seen[n + 1];

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

{

seen[i] = false;

}

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

scanf("%d", &arr[i]);

}

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

int num =arr[i];

if (seen[num]) {

printf("%d\n", num);

return 0;

}

seen[num] = true;

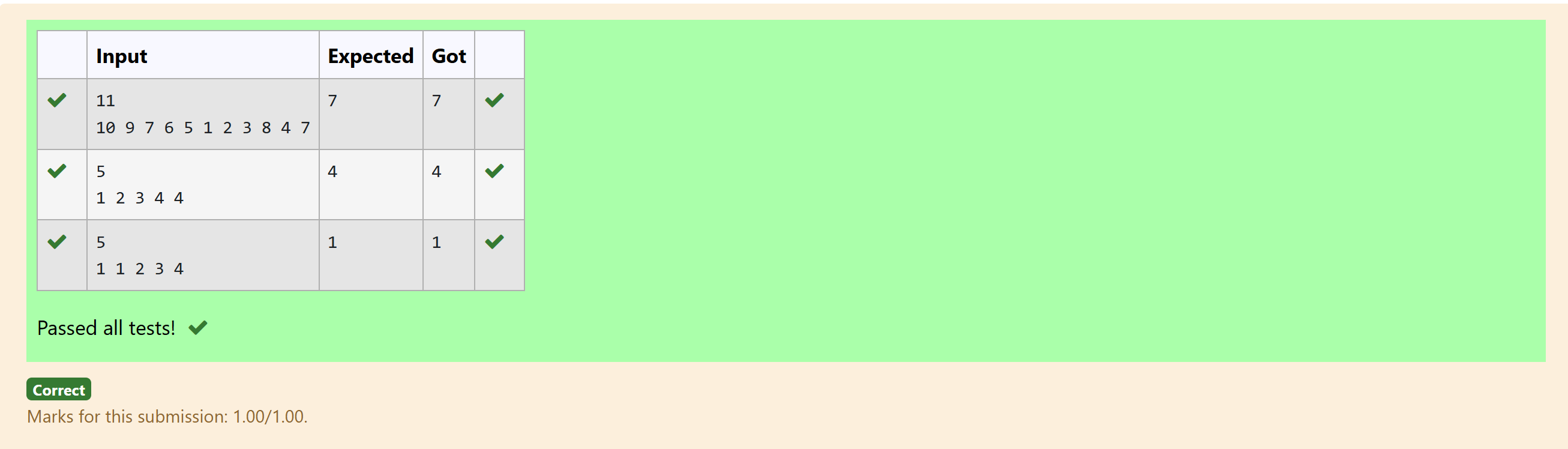
}

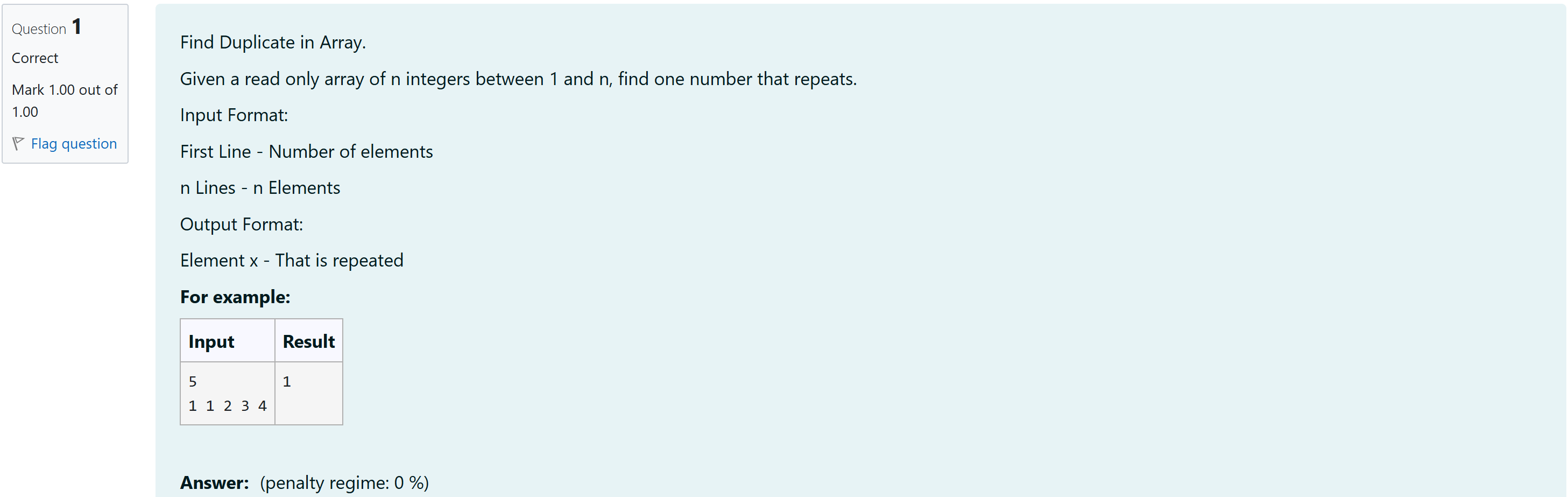
printf("No duplicate found\n");

return 0;

}

Output:





#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

int main() {

int n;

scanf("%d", &n);

if (n <= 0) {

printf("Invalid array size\n");

return 1;

}

int \*arr = malloc(n \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed\n");

return 1;

}

int max\_value = 0;

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

scanf("%d", &arr[i]);

if (arr[i] > max\_value) {

max\_value = arr[i];

}

}

bool \*seen = calloc(max\_value + 1, sizeof(bool));

if (seen == NULL) {

printf("Memory allocation failed\n");

free(arr);

return 1;

}

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

int num = arr[i];

if (num < 0 || num > max\_value) {

printf("Value out of range\n");

free(arr);

free(seen);

return 1;

}

if (seen[num]) {

printf("%d\n", num);

free(arr);

free(seen);

return 0;

}

seen[num] = true;

}

printf("No duplicate found\n");

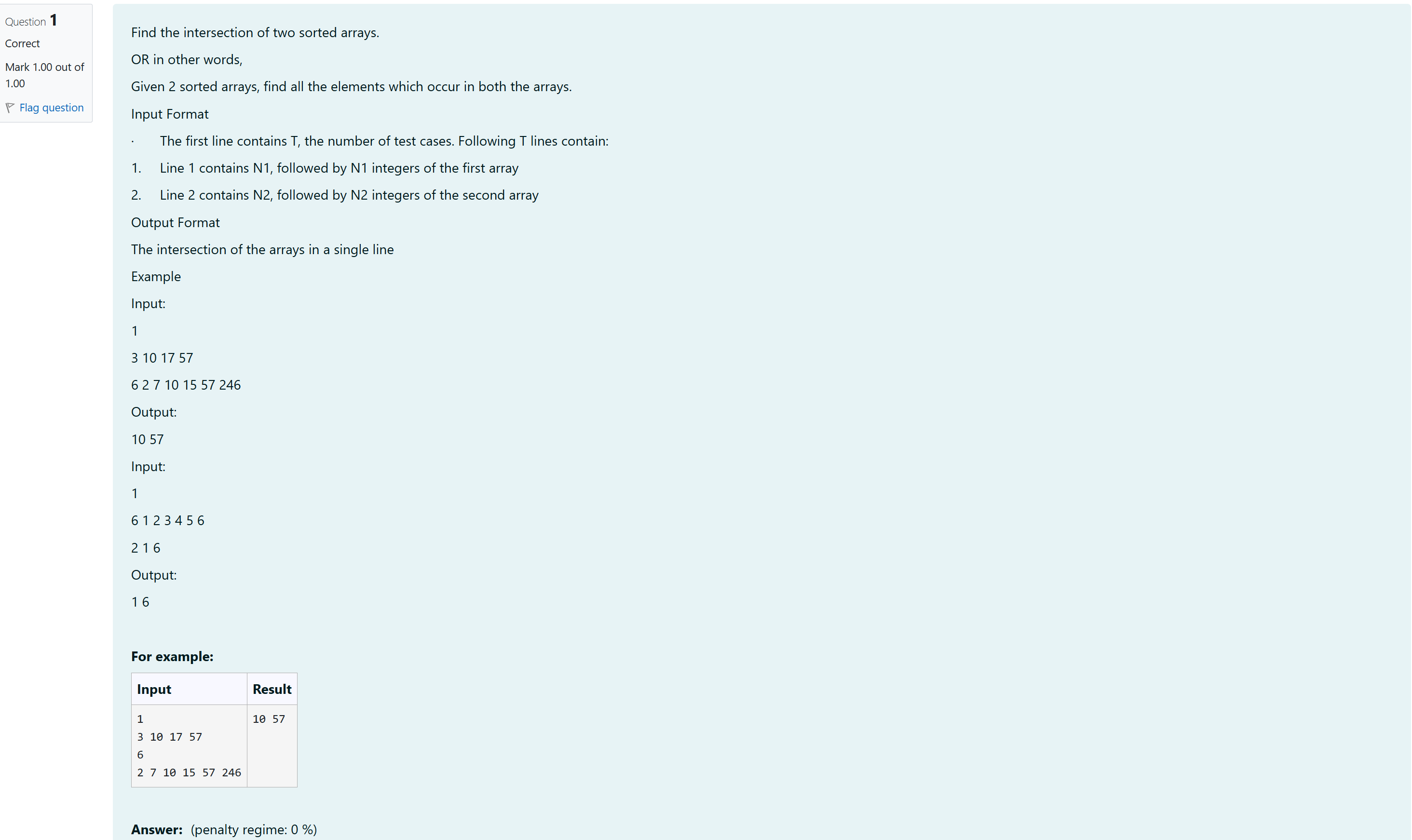
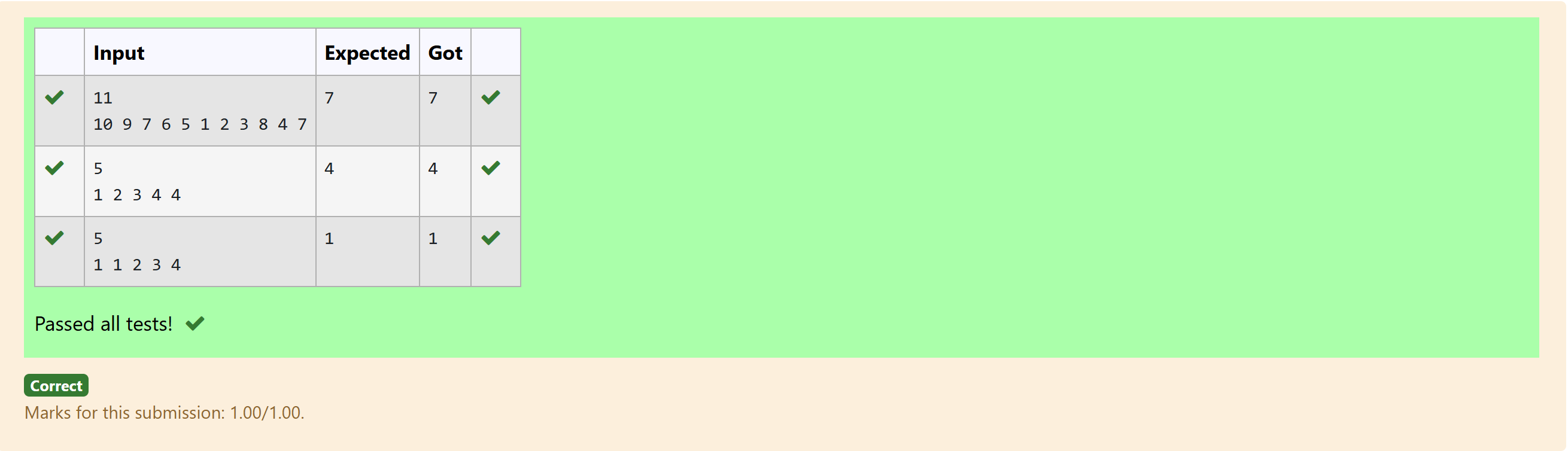
free(arr);

free(seen);

return 0;

}

Output:



#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {

int i = 0, j = 0;

int isFirst = 1;

while (i < n1 && j < n2) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr2[j] < arr1[i]) {

j++;

} else {

if (!isFirst) {

printf(" ");

}

printf("%d", arr1[i]);

isFirst = 0;

i++;

j++;

}

}

if (isFirst) {

printf("No common elements");

}

printf("\n");

}

int main() {

int T;

scanf("%d", &T);

while (T--) {

int n1, n2;

scanf("%d", &n1);

int arr1[n1];

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]);

}

scanf("%d", &n2);

int arr2[n2];

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]);

}

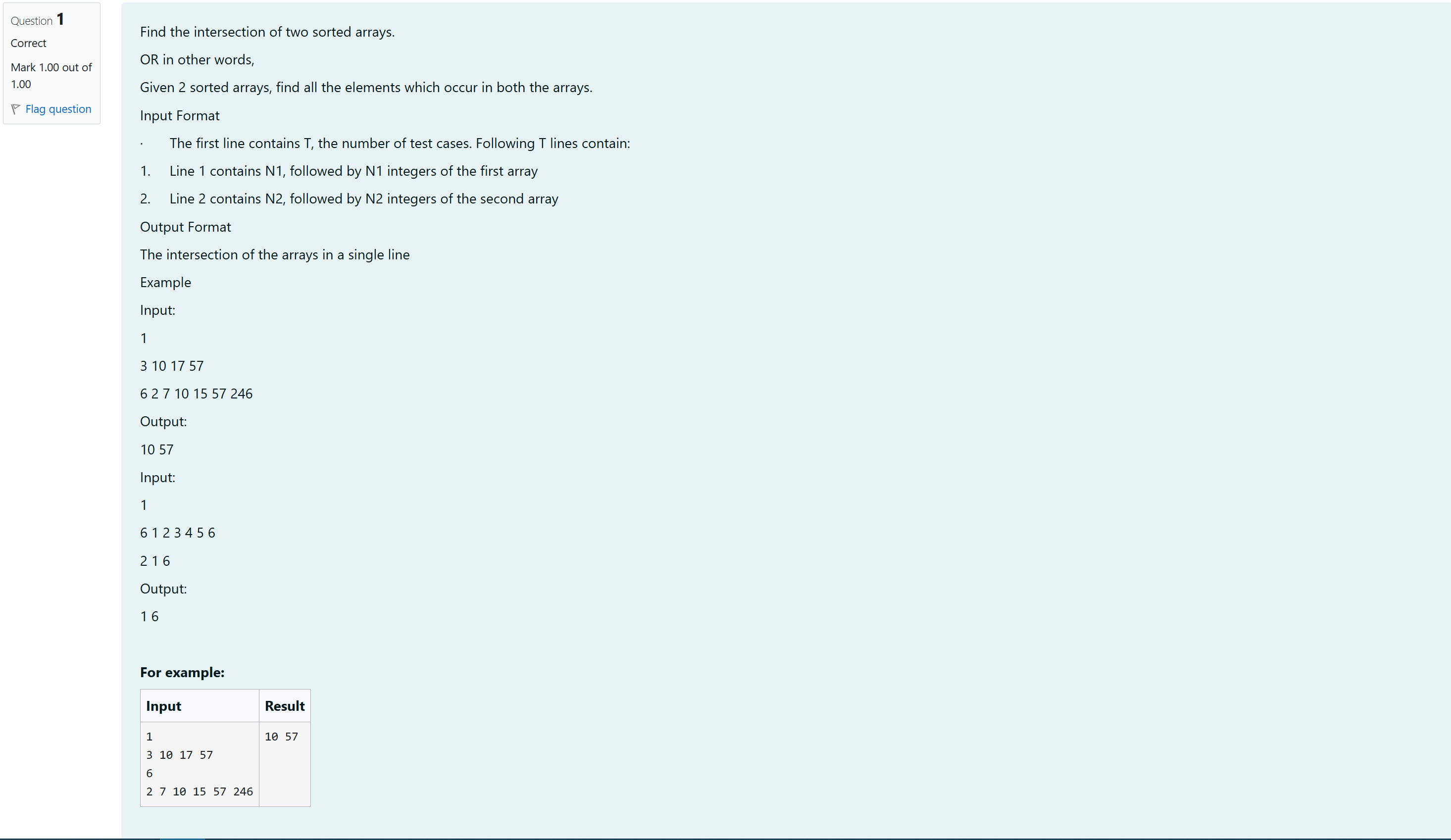
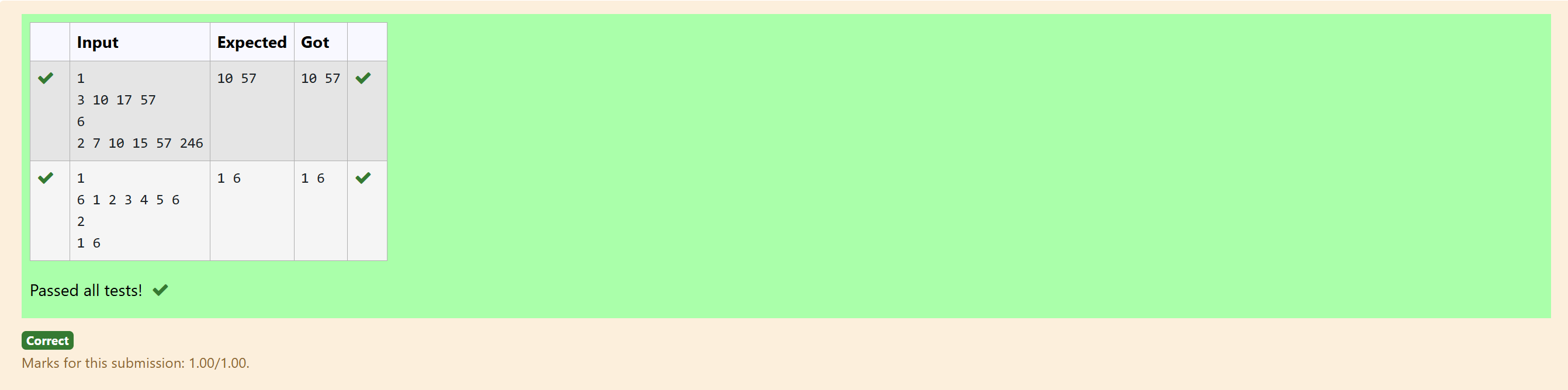
findIntersection(arr1, n1, arr2, n2);

}

return 0;

}

Output:



#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {

int i = 0, j = 0;

int isFirst = 1;

while (i < n1 && j < n2) {

if (arr1[i] < arr2[j]) {

i++;

} else if (arr2[j] < arr1[i]) {

j++;

} else {

if (!isFirst) {

printf(" ");

}

printf("%d", arr1[i]);

isFirst = 0;

i++;

j++;

}

}

if (isFirst) {

printf("No common elements");

}

printf("\n");

}

int main() {

int T;

scanf("%d", &T);

while (T--) {

int n1, n2;

scanf("%d", &n1);

int arr1[n1];

for (int i = 0; i < n1; i++) {

scanf("%d", &arr1[i]);

}

scanf("%d", &n2);

int arr2[n2];

for (int i = 0; i < n2; i++) {

scanf("%d", &arr2[i]);

}

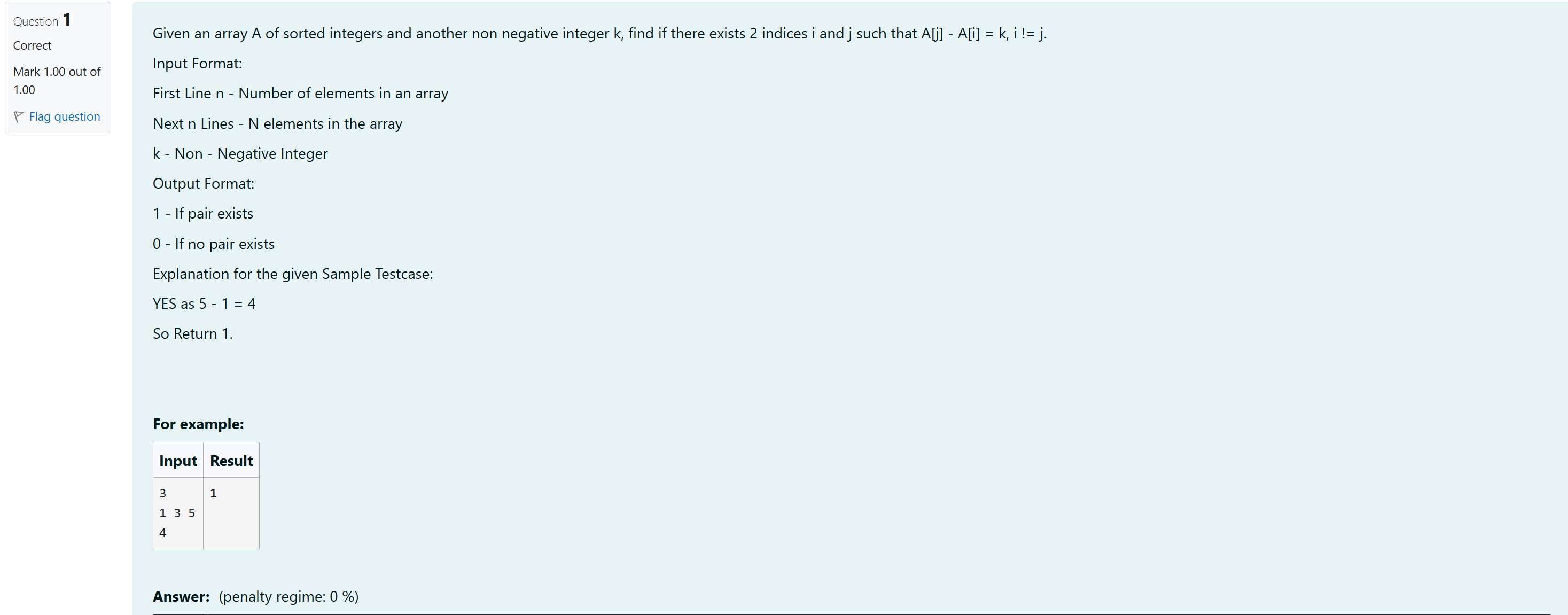
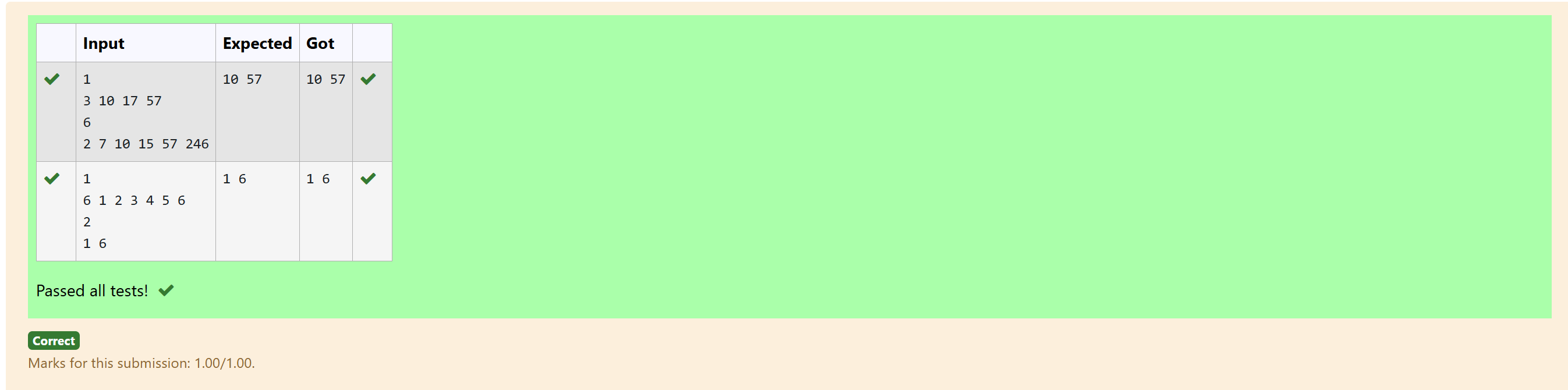
findIntersection(arr1, n1, arr2, n2);

}

return 0;

}

Output:



#include <stdio.h>

int findPairWithDifference(int arr[], int n, int k) {

int i = 0, j = 1;

while (j < n) {

int diff = arr[j] - arr[i];

if (i != j && diff == k) {

return 1;

} else if (diff < k) {

j++;

} else {

i++;

if (i == j) {

j++;

}

}

}

return 0;

}

int main() {

int n, k;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

scanf("%d", &k);

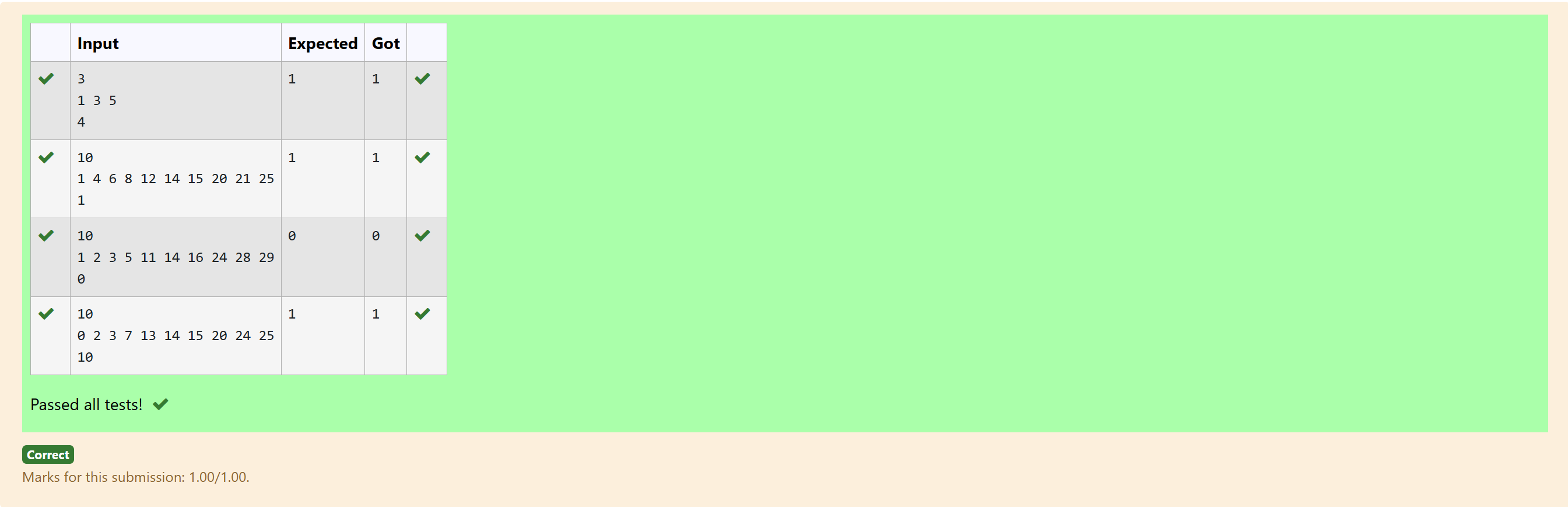
int result = findPairWithDifference(arr, n, k);

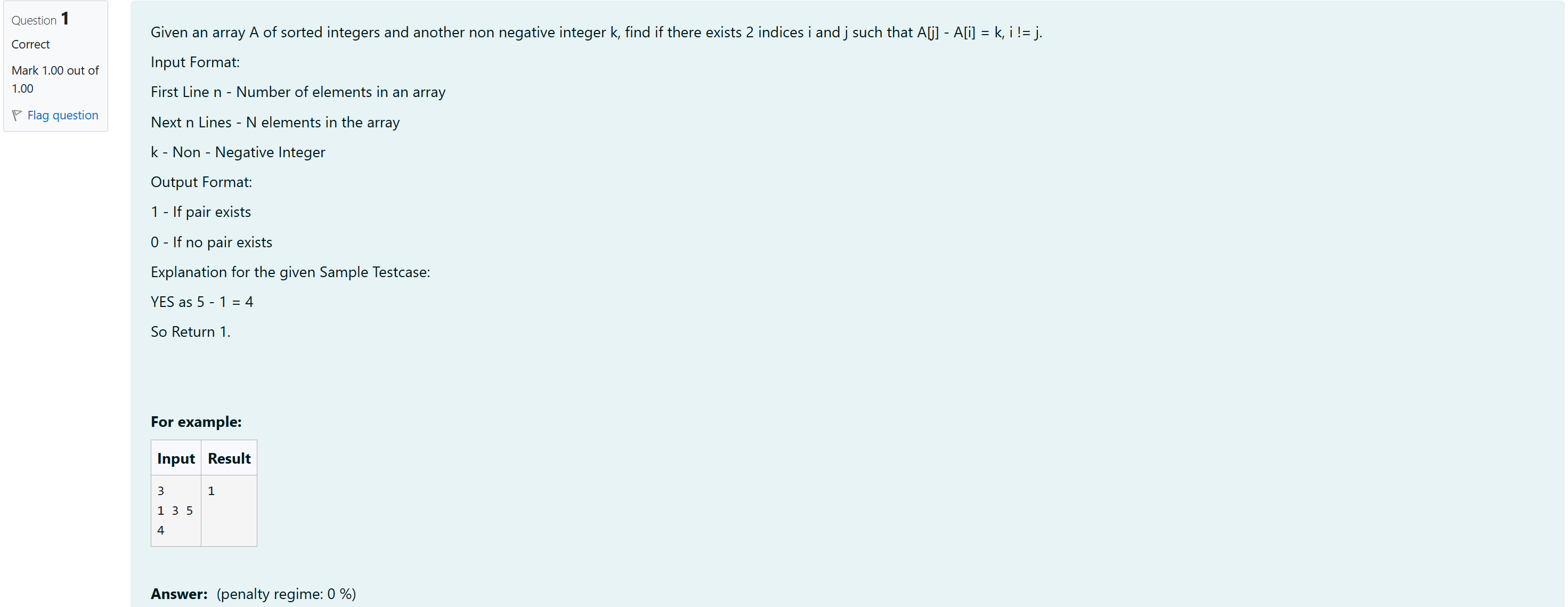
printf("%d\n", result);

return 0;

}

Output:





#include <stdio.h>

int findPairWithDifference(int arr[], int n, int k) {

int i = 0, j = 1;

while (j < n) {

int diff = arr[j] - arr[i];

if (i != j && diff == k) {

return 1;

} else if (diff < k) {

j++;

} else {

i++;

if (i == j) {

j++;

}

}

}

return 0;

}

int main() {

int n, k;

scanf("%d", &n);

int arr[n];

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

scanf("%d", &arr[i]);

}

scanf("%d", &k);

int result = findPairWithDifference(arr, n, k);

printf("%d\n", result);

return 0;

}

