RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM - 602 105

RAJALAKSHMI

ENGINEERING COLLEGE

An AUTONOMOUS Institution

Affiliated to ANNA UNIVERSITY, Chennai

CS23331 DESIGN AND ANALYSIS OF ALGORITHM

Laboratory Record Notebook

Name : IRSHAN M

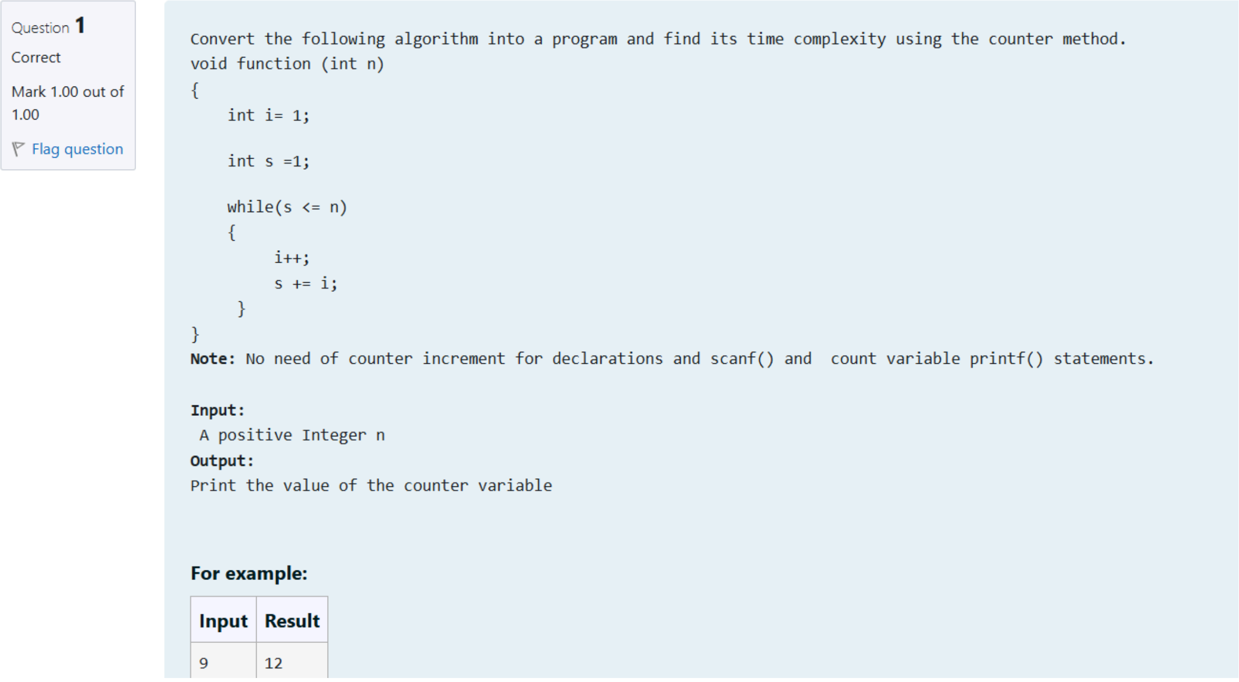
Year / Branch / Section : II / AIML / B

University Register No. : 2116231501505

College Roll No. : 231501505

Semester : 3rd SEMESTER

Academic Year : 2024-2025



Question **1**

Correct

Mark 1.00 out of 1.00

Flag question

```
Convert the following algorithm into a program and find its time complexity using the counter method.
void function (int n)
{
    int i= 1;

    int s =1;

    while(s <= n)
    {
         i++;
         s += i;
     }
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

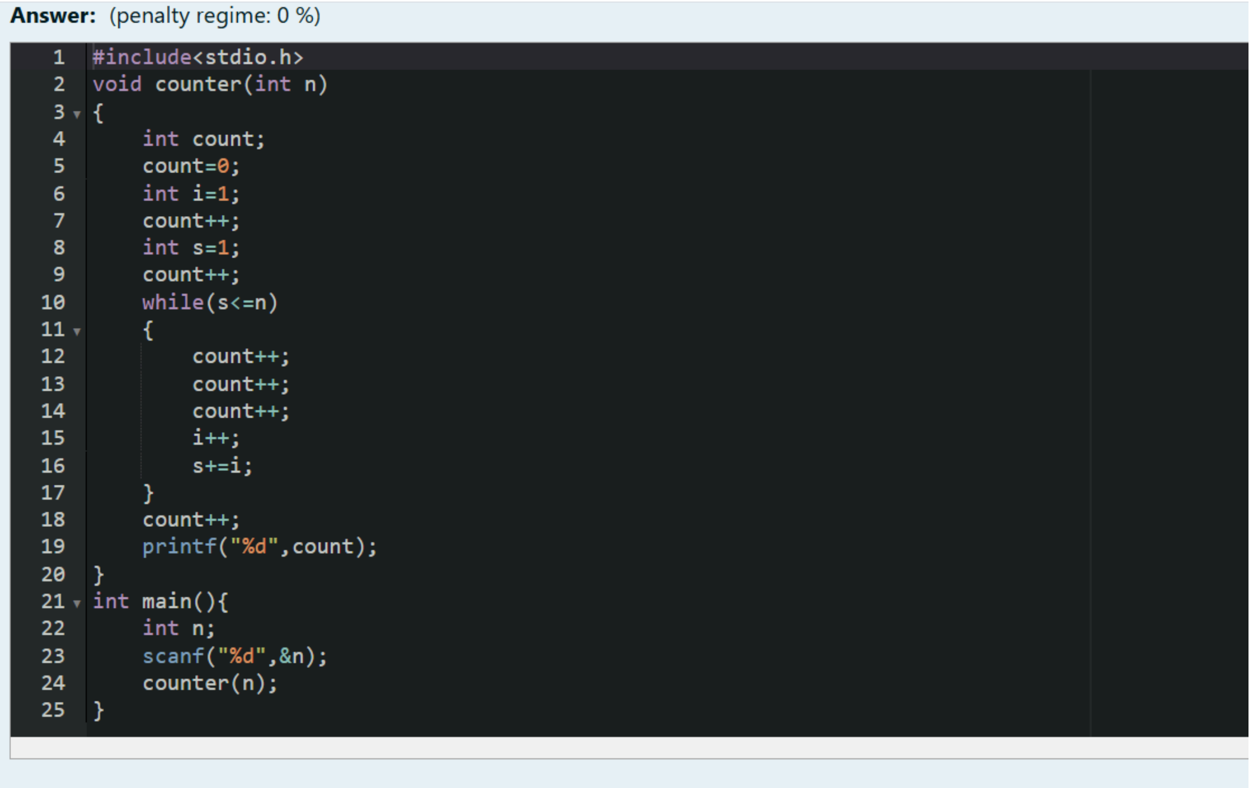
A positive Integer n

Output:

Print the value of the counter variable

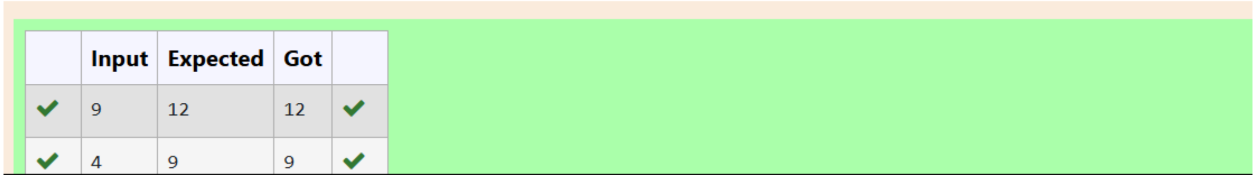
For example:

Input	Result
9	12

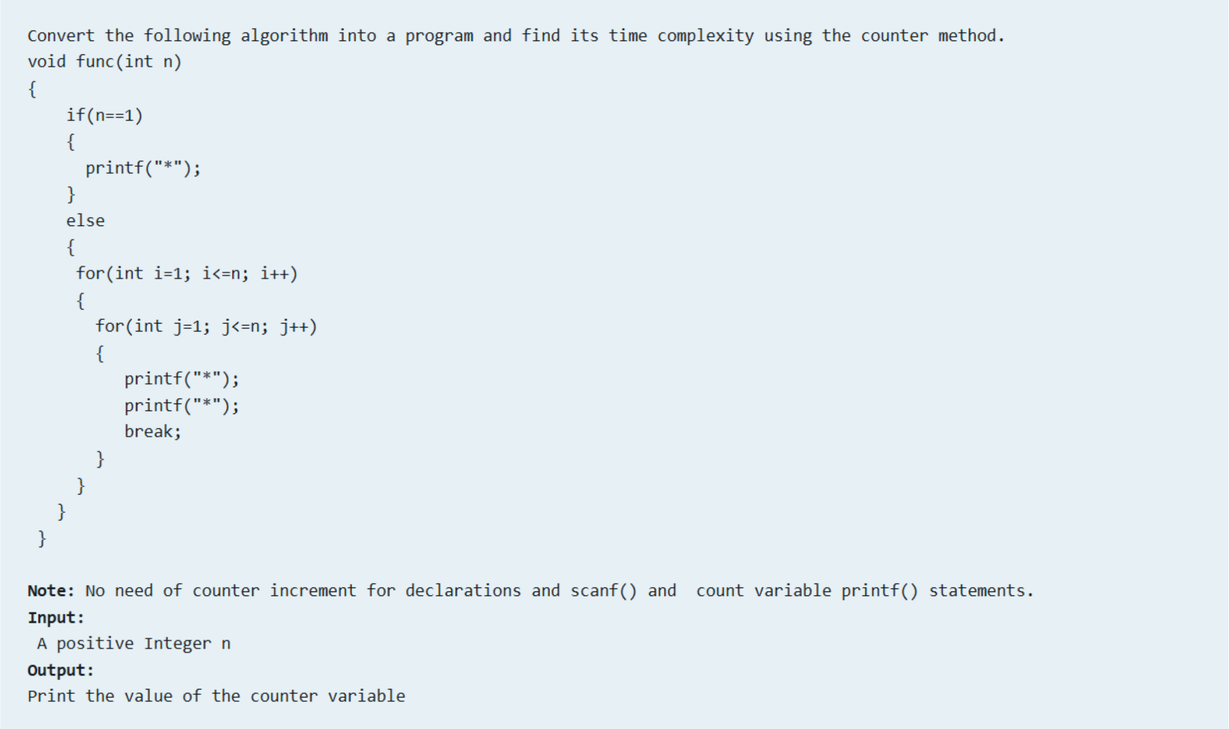


Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
void counter(int n)
{
    int count;
    count=0;
    int i=1;
    count++;
    int s=1;
    count++;
    while(s<=n)
    {
        count++;
        count++;
        count++;
        i++;
        s+=i;
    }
    count++;
    printf("%d",count);
}
int main(){
    int n;
    scanf("%d",&n);
    counter(n);
}
```



	Input	Expected	Got	
✓	9	12	12	✓
✓	4	9	9	✓



Convert the following algorithm into a program and find its time complexity using the counter method.

```
void func(int n)
{
    if(n==1)
    {
      printf("*");
    }
    else
    {
     for(int i=1; i<=n; i++)
     {
       for(int j=1; j<=n; j++)
       {
          printf("*");
          printf("*");
          break;
       }
     }
   }
 }
```

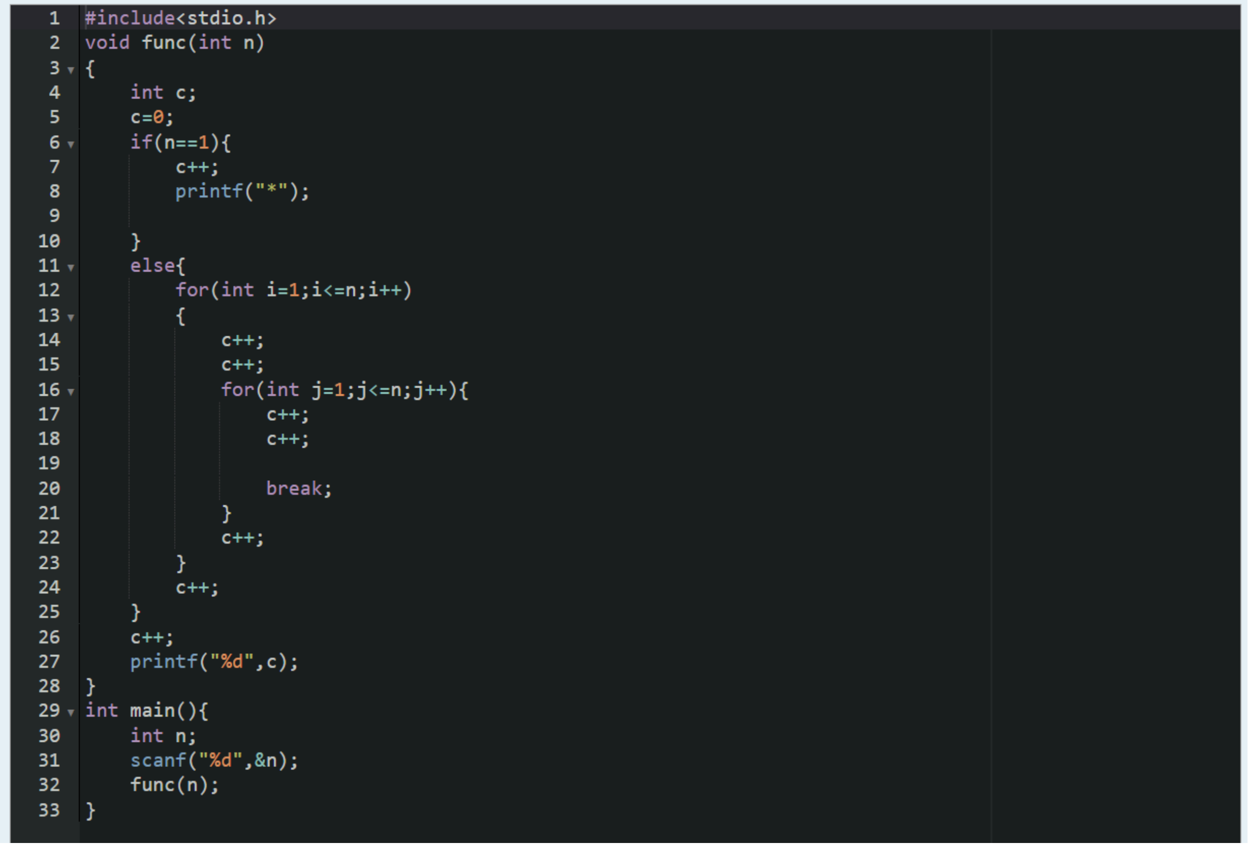
Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

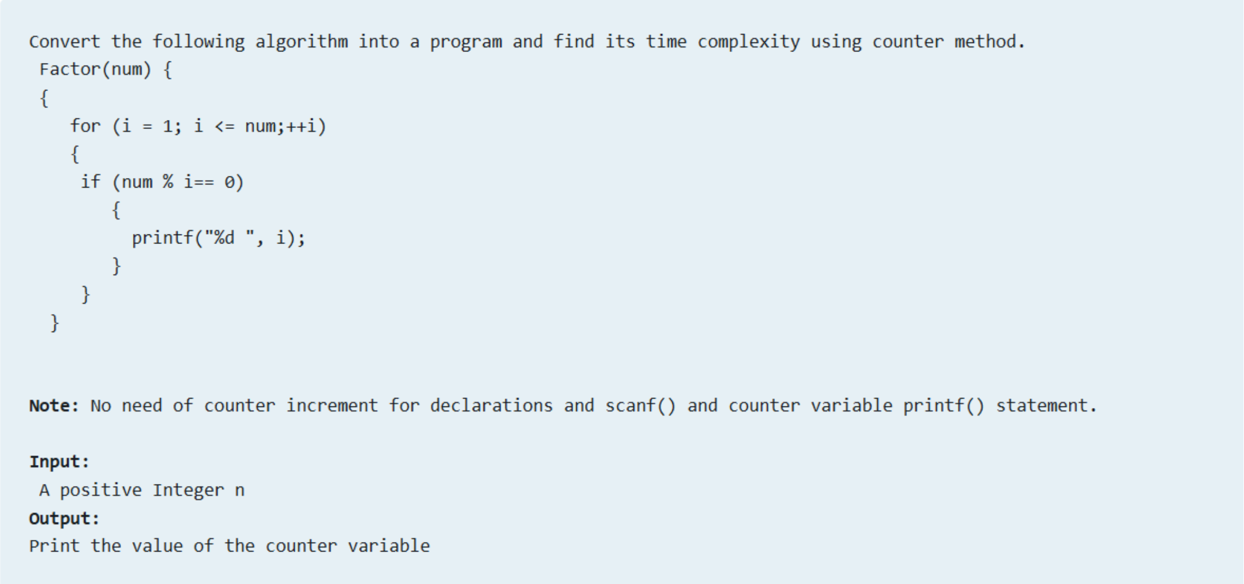
A positive Integer n

Output:

Print the value of the counter variable



```
#include<stdio.h>
void func(int n)
{
    int c;
    c=0;
    if(n==1){
        c++;
        printf("*");

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

                break;
            }
            c++;
        }
        c++;
    }
    c++;
    printf("%d",c);
}
int main(){
    int n;
    scanf("%d",&n);
    func(n);
}
```



Convert the following algorithm into a program and find its time complexity using counter method.

```
 Factor(num) {
 {
    for (i = 1; i <= num;++i)
    {
     if (num % i== 0)
        {
          printf("%d ", i);
        }
     }
  }
```

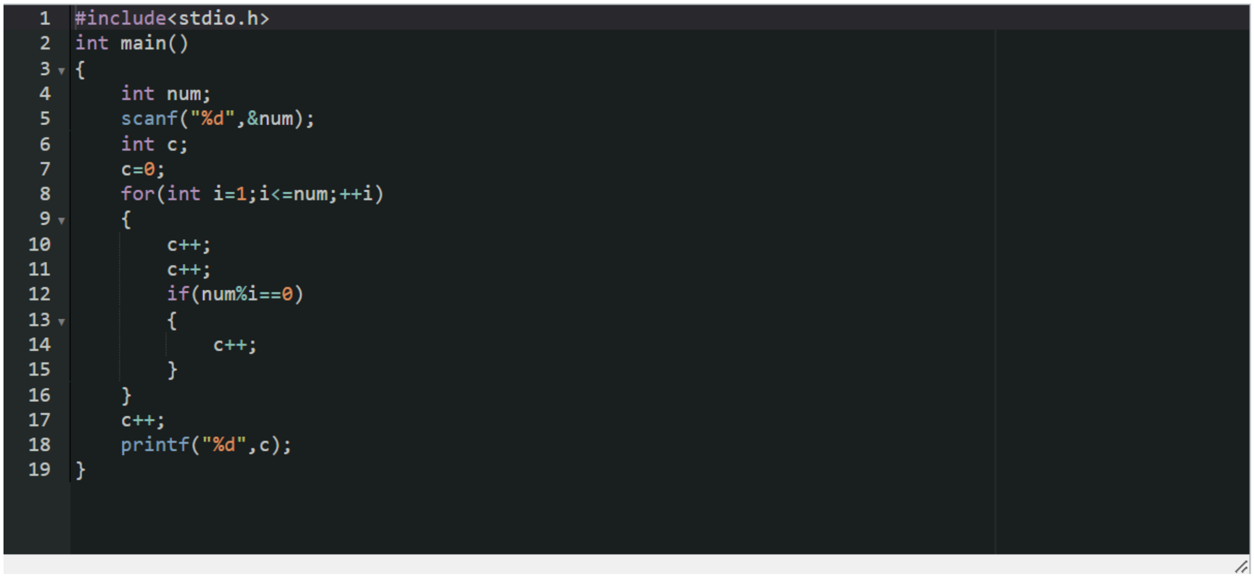
Note: No need of counter increment for declarations and scanf() and counter variable printf() statement.

Input:

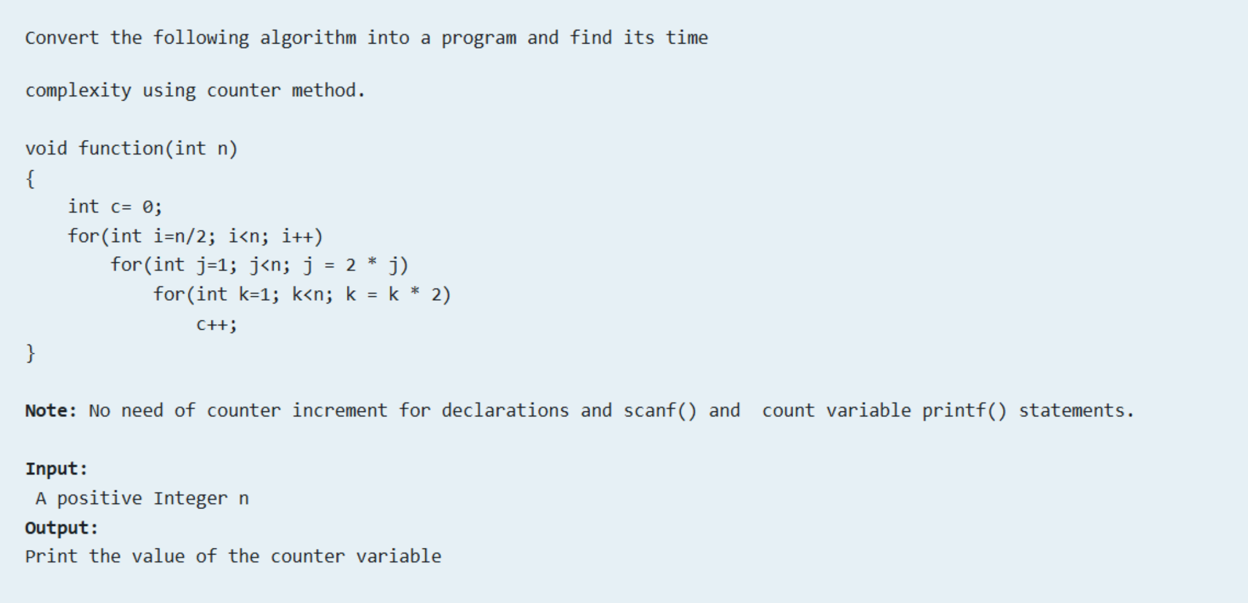
A positive Integer n

Output:

Print the value of the counter variable



```c
#include<stdio.h>
int main()
{
    int num;
    scanf("%d",&num);
    int c;
    c=0;
    for(int i=1;i<=num;++i)
    {
        c++;
        c++;
        if(num%i==0)
        {
            c++;
        }
    }
    c++;
    printf("%d",c);
}
```



Convert the following algorithm into a program and find its time

complexity using counter method.

```
void function(int n)
{
    int c= 0;
    for(int i=n/2; i<n; i++)
        for(int j=1; j<n; j = 2 * j)
            for(int k=1; k<n; k = k * 2)
                c++;
}
```

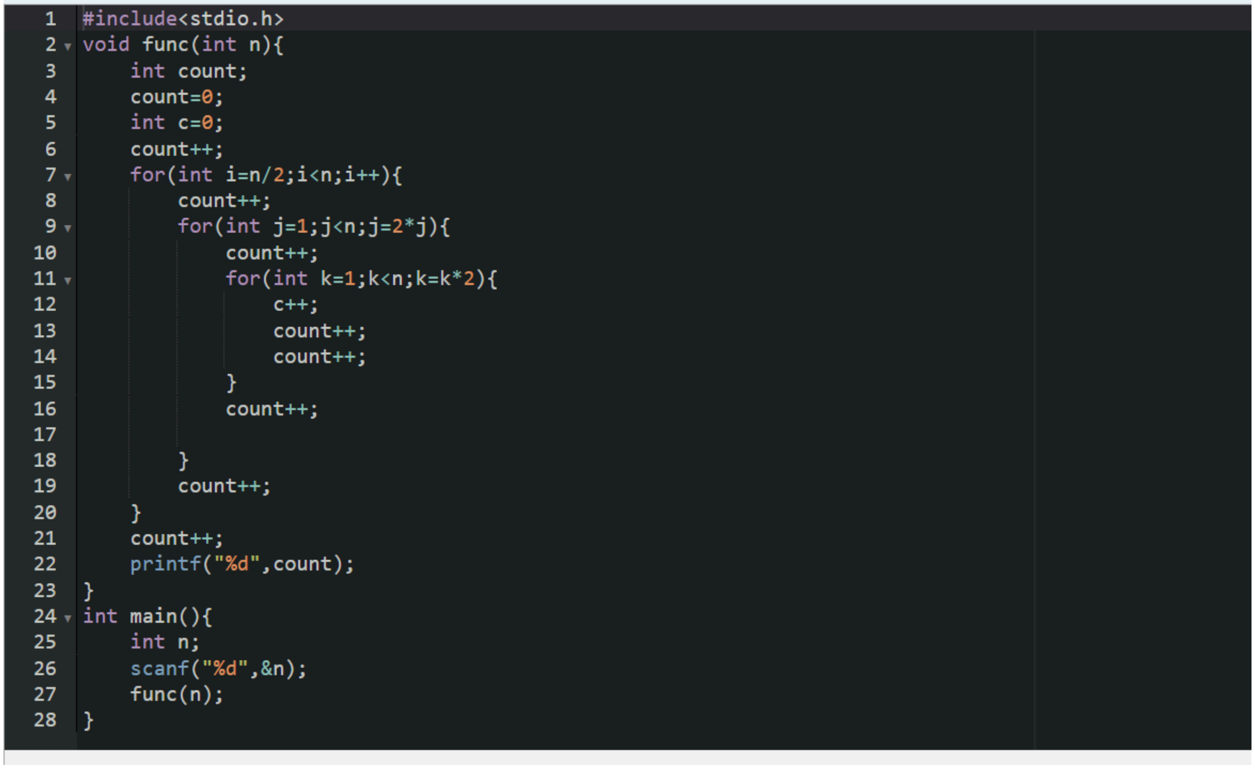
Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

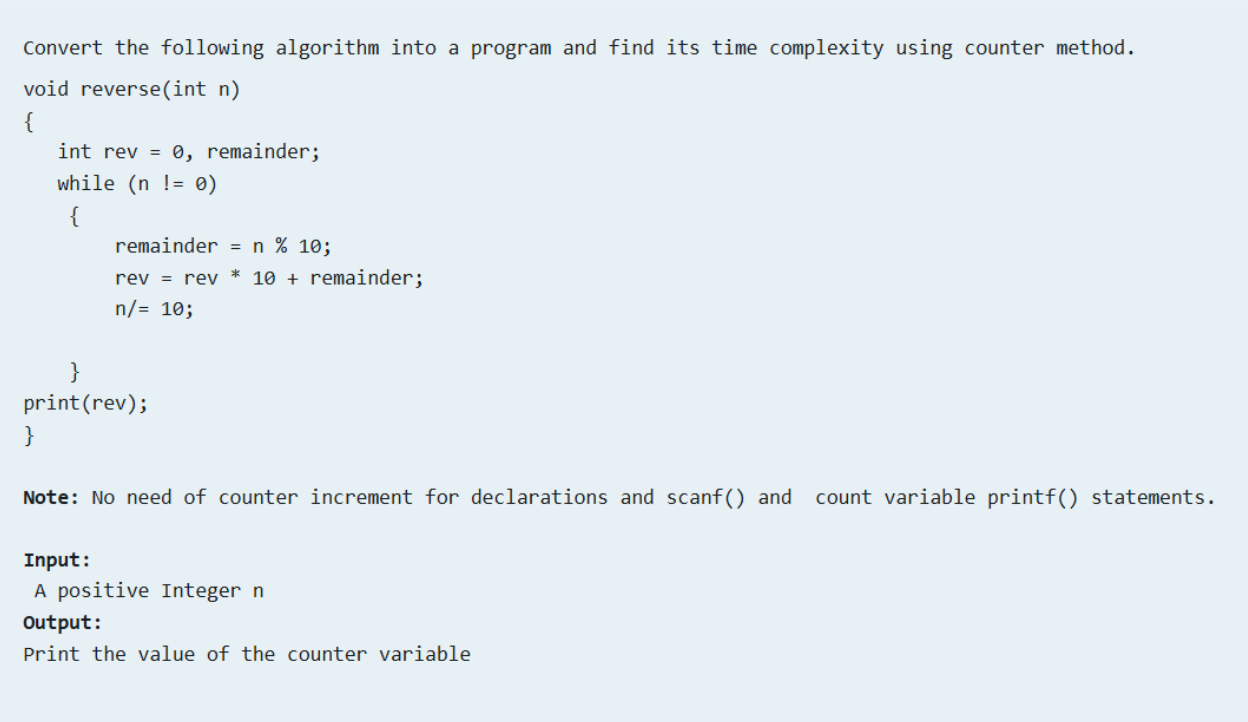
A positive Integer n

Output:

Print the value of the counter variable



```
#include<stdio.h>
void func(int n){
    int count;
    count=0;
    int c=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){
                c++;
                count++;
                count++;
            }
            count++;

        }
        count++;
    }
    count++;
    printf("%d",count);
}
int main(){
    int n;
    scanf("%d",&n);
    func(n);
}
```



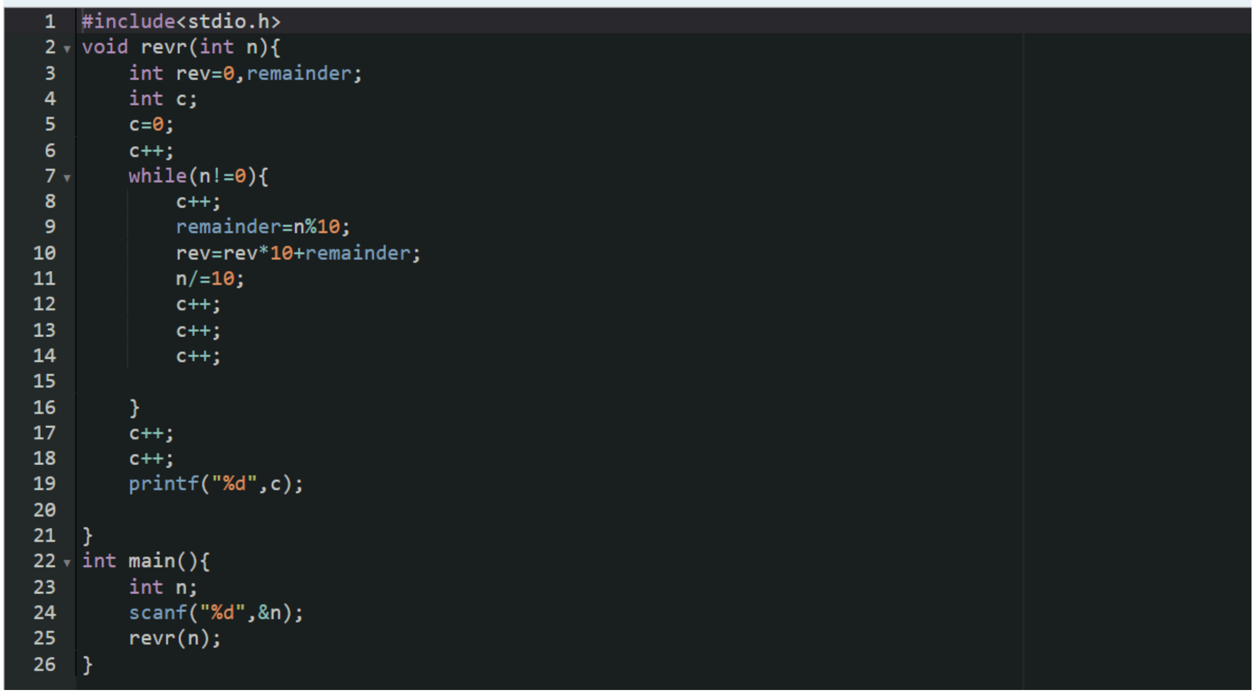
Convert the following algorithm into a program and find its time complexity using counter method.

```
void reverse(int n)
{
   int rev = 0, remainder;
   while (n != 0)
    {
        remainder = n % 10;
        rev = rev * 10 + remainder;
        n/= 10;

    }
print(rev);
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:
A positive Integer n

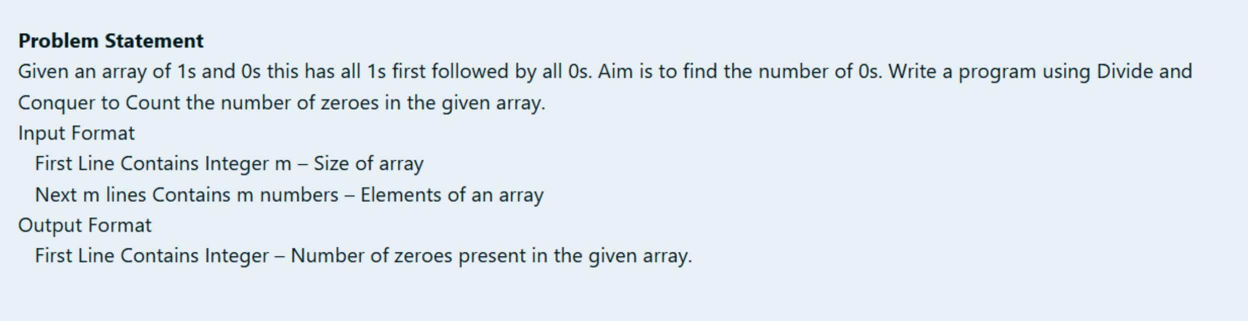
Output:
Print the value of the counter variable



```c
#include<stdio.h>
void revr(int n){
    int rev=0,remainder;
    int c;
    c=0;
    c++;
    while(n!=0){
        c++;
        remainder=n%10;
        rev=rev*10+remainder;
        n/=10;
        c++;
        c++;
        c++;

    }
    c++;
    c++;
    printf("%d",c);

}
int main(){
    int n;
    scanf("%d",&n);
    revr(n);
}
```

Divide and conquer

1. Number of zeros in given array



Problem Statement

Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

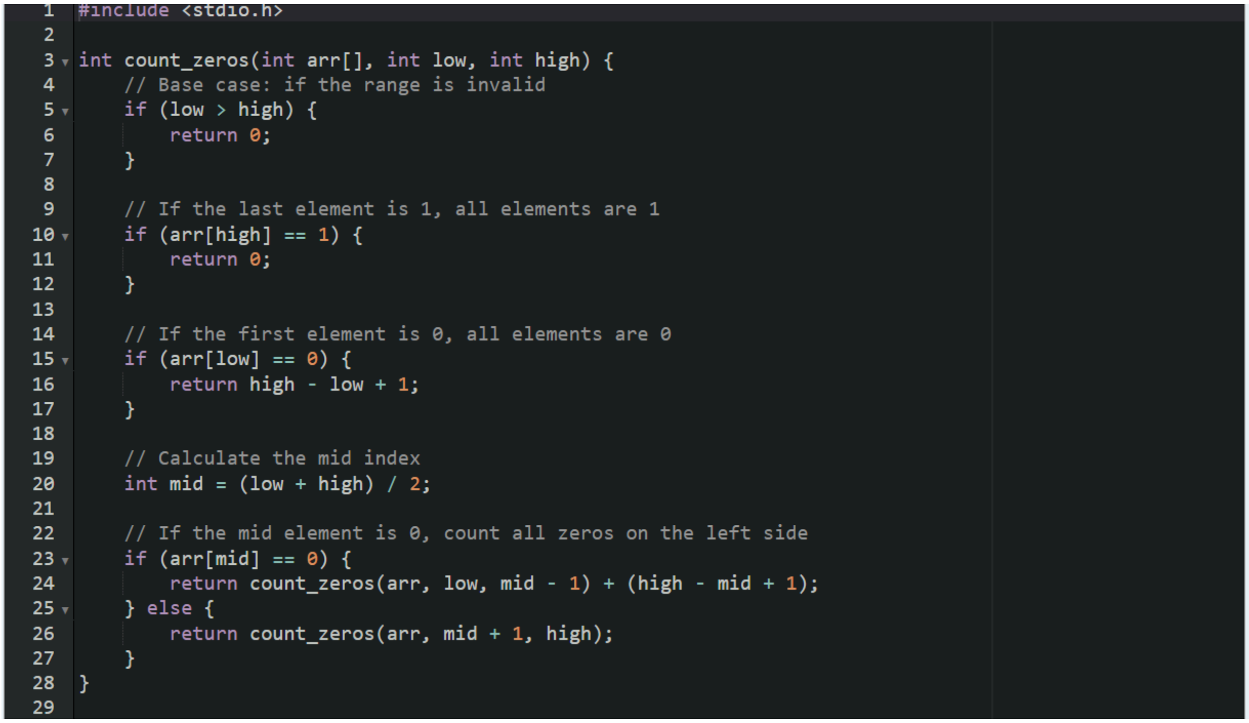
Input Format

First Line Contains Integer m – Size of array

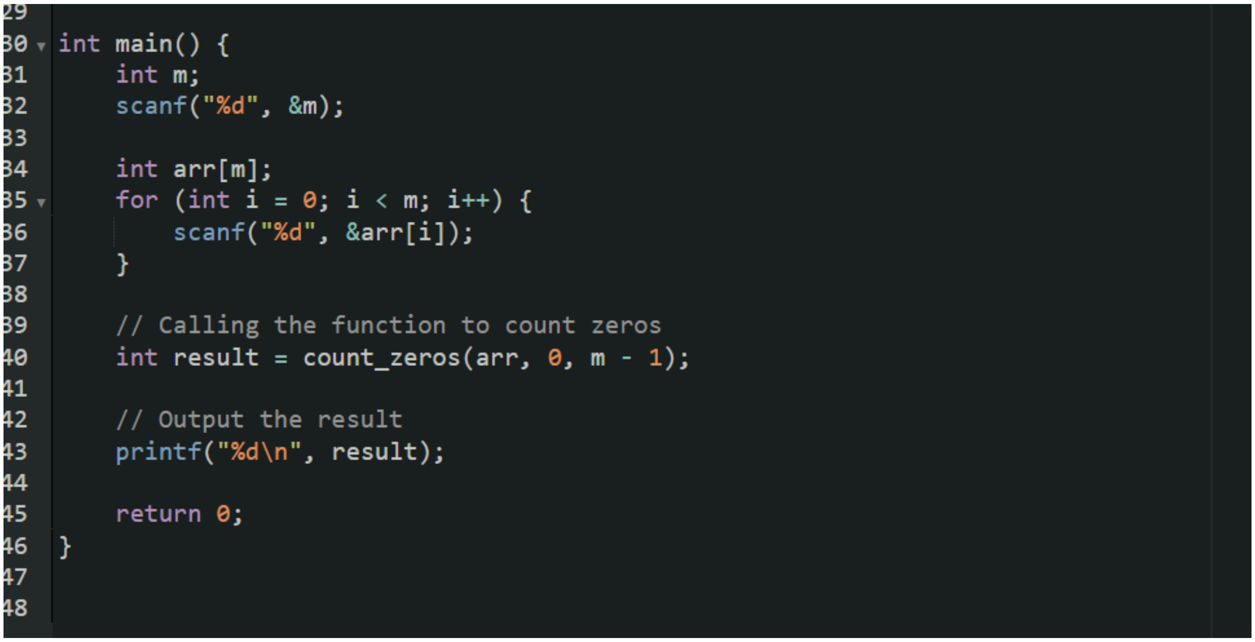
Next m lines Contains m numbers – Elements of an array

Output Format

First Line Contains Integer – Number of zeroes present in the given array.

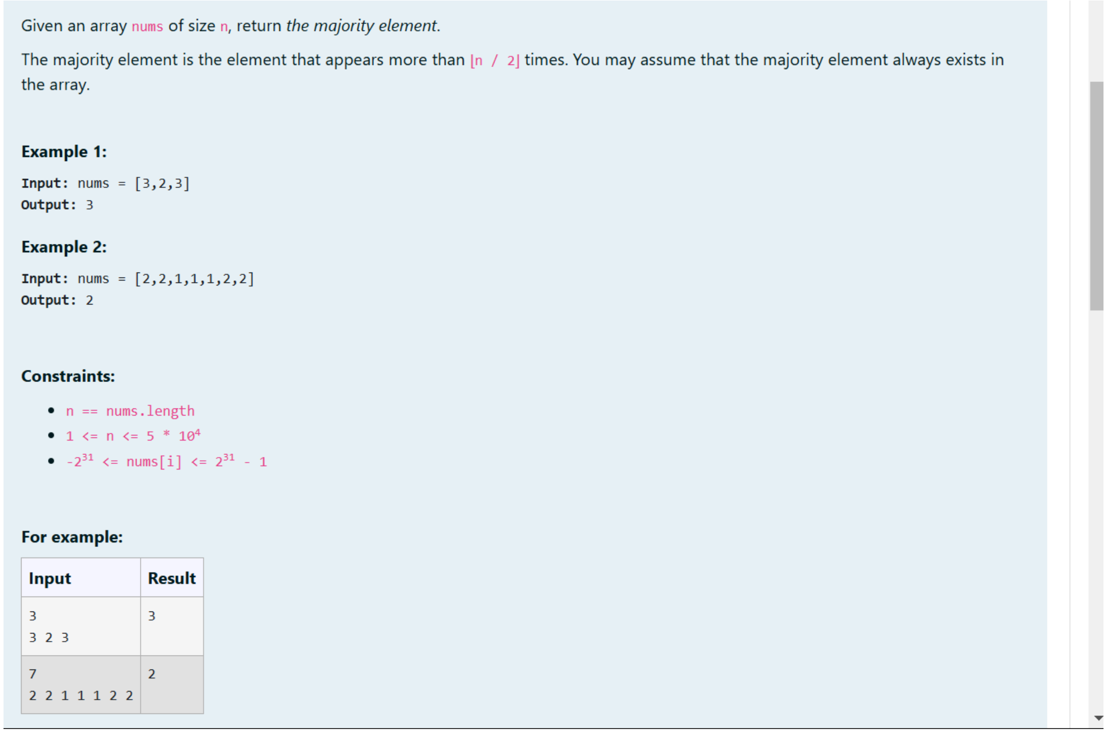


```c
#include <stdio.h>

int count_zeros(int arr[], int low, int high) {
    // Base case: if the range is invalid
    if (low > high) {
        return 0;
    }

    // If the last element is 1, all elements are 1
    if (arr[high] == 1) {
        return 0;
    }

    // If the first element is 0, all elements are 0
    if (arr[low] == 0) {
        return high - low + 1;
    }

    // Calculate the mid index
    int mid = (low + high) / 2;

    // If the mid element is 0, count all zeros on the left side
    if (arr[mid] == 0) {
        return count_zeros(arr, low, mid - 1) + (high - mid + 1);
    } else {
        return count_zeros(arr, mid + 1, high);
    }
}

```



```c
int main() {
    int m;
    scanf("%d", &m);

    int arr[m];
    for (int i = 0; i < m; i++) {
        scanf("%d", &arr[i]);
    }

    // Calling the function to count zeros
    int result = count_zeros(arr, 0, m - 1);

    // Output the result
    printf("%d\n", result);

    return 0;
}
```

2. Majority element



Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

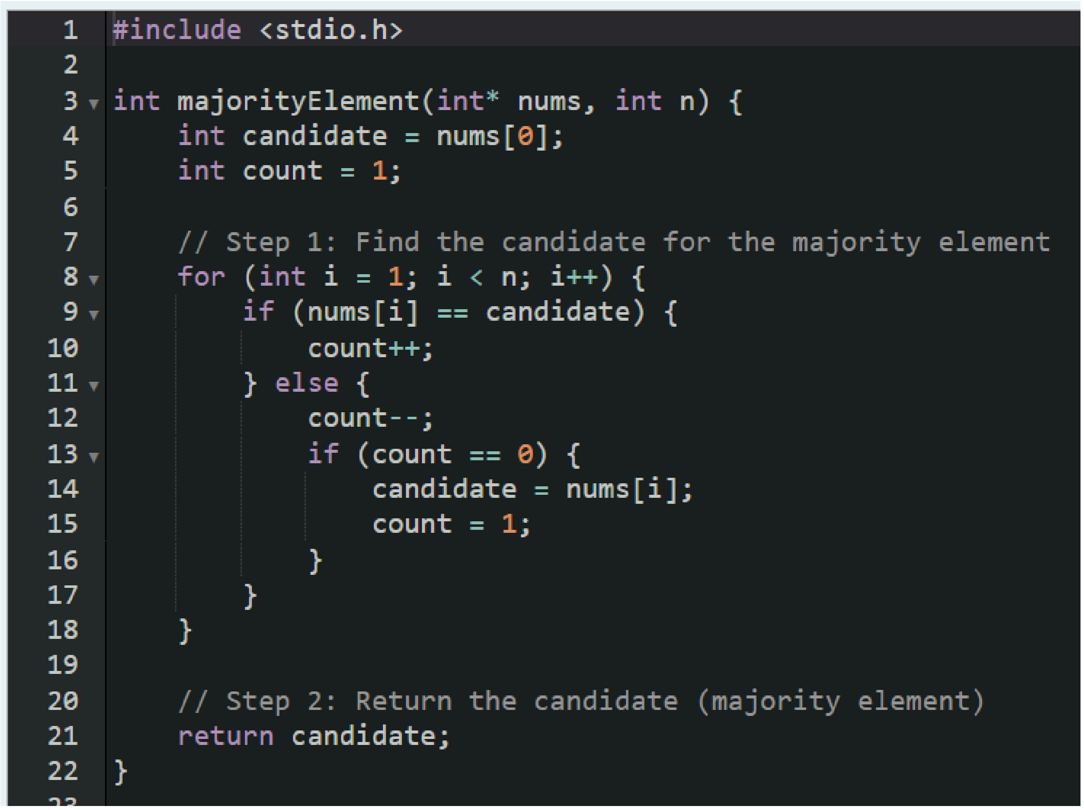
```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

Constraints:

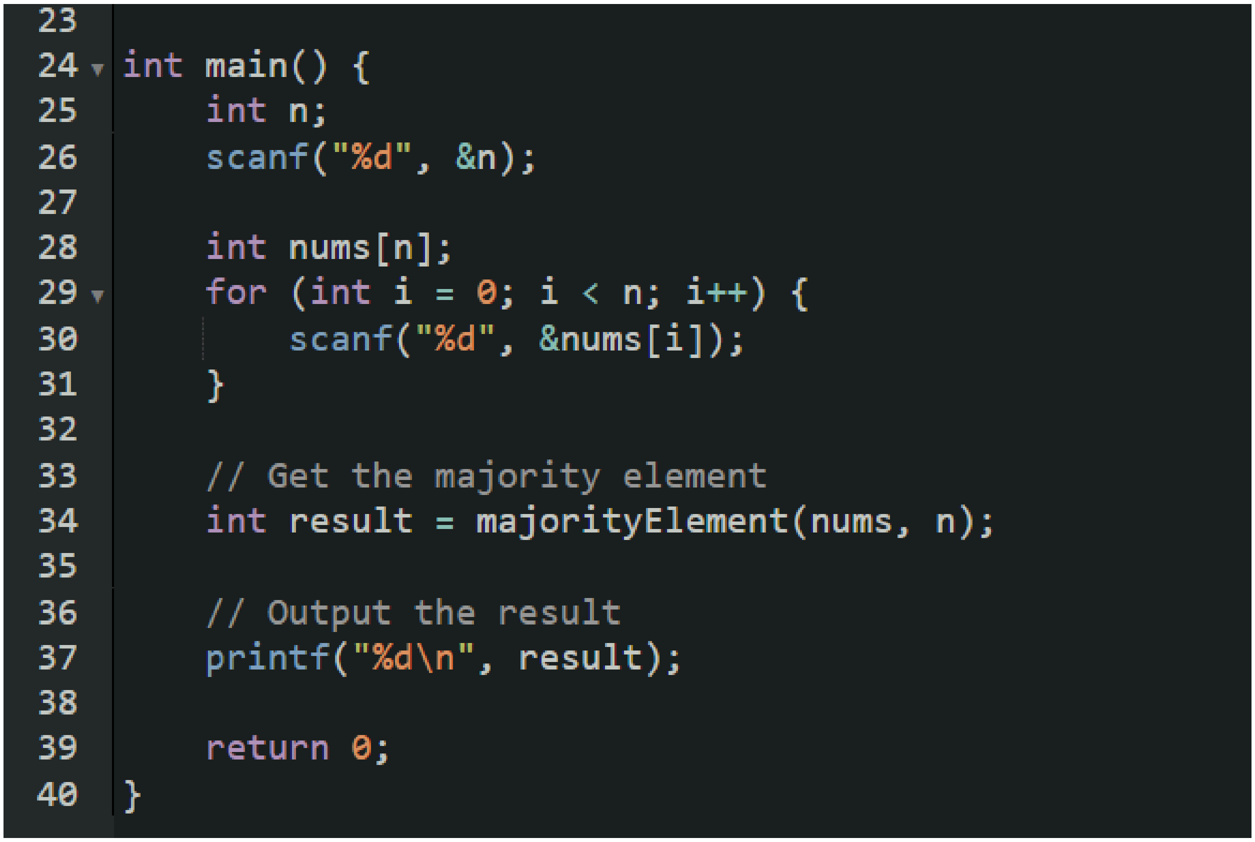
- `n == nums.length`
- $1 <= n <= 5 * 10^4$
- $-2^{31} <= nums[i] <= 2^{31} - 1$

For example:

Input	Result
3 3 2 3	3
7 2 2 1 1 1 2 2	2

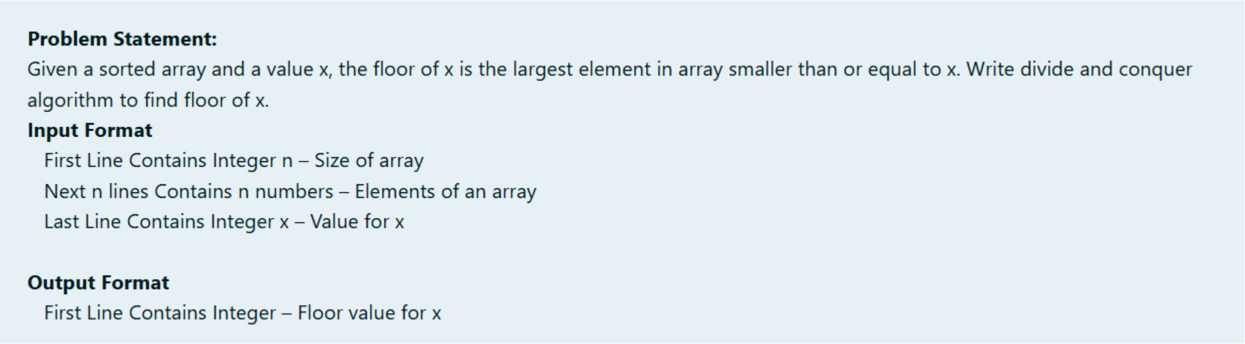


```c
#include <stdio.h>

int majorityElement(int* nums, int n) {
    int candidate = nums[0];
    int count = 1;

    // Step 1: Find the candidate for the majority element
    for (int i = 1; i < n; i++) {
        if (nums[i] == candidate) {
            count++;
        } else {
            count--;
            if (count == 0) {
                candidate = nums[i];
                count = 1;
            }
        }
    }

    // Step 2: Return the candidate (majority element)
    return candidate;
}
```



```c

int main() {
    int n;
    scanf("%d", &n);

    int nums[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &nums[i]);
    }

    // Get the majority element
    int result = majorityElement(nums, n);

    // Output the result
    printf("%d\n", result);

    return 0;
}
```

3. Finding floor value



Problem Statement:

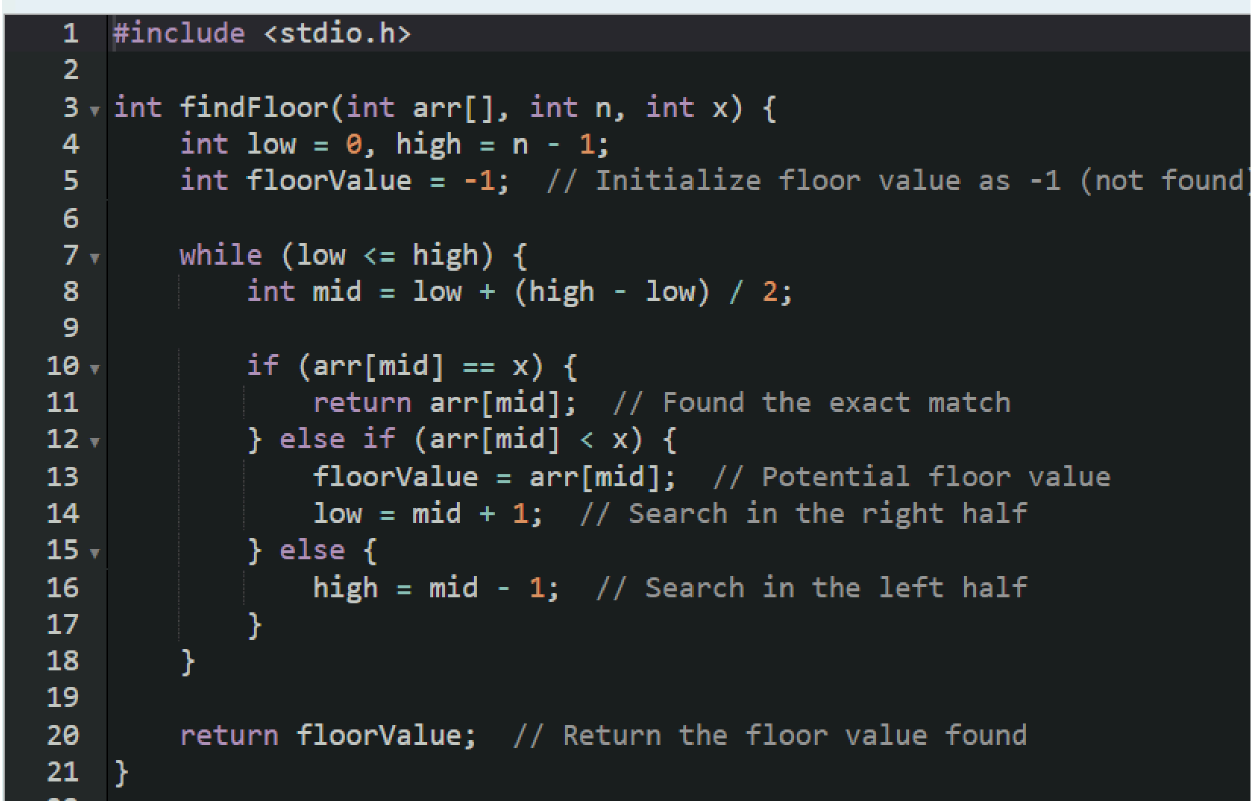
Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

Input Format

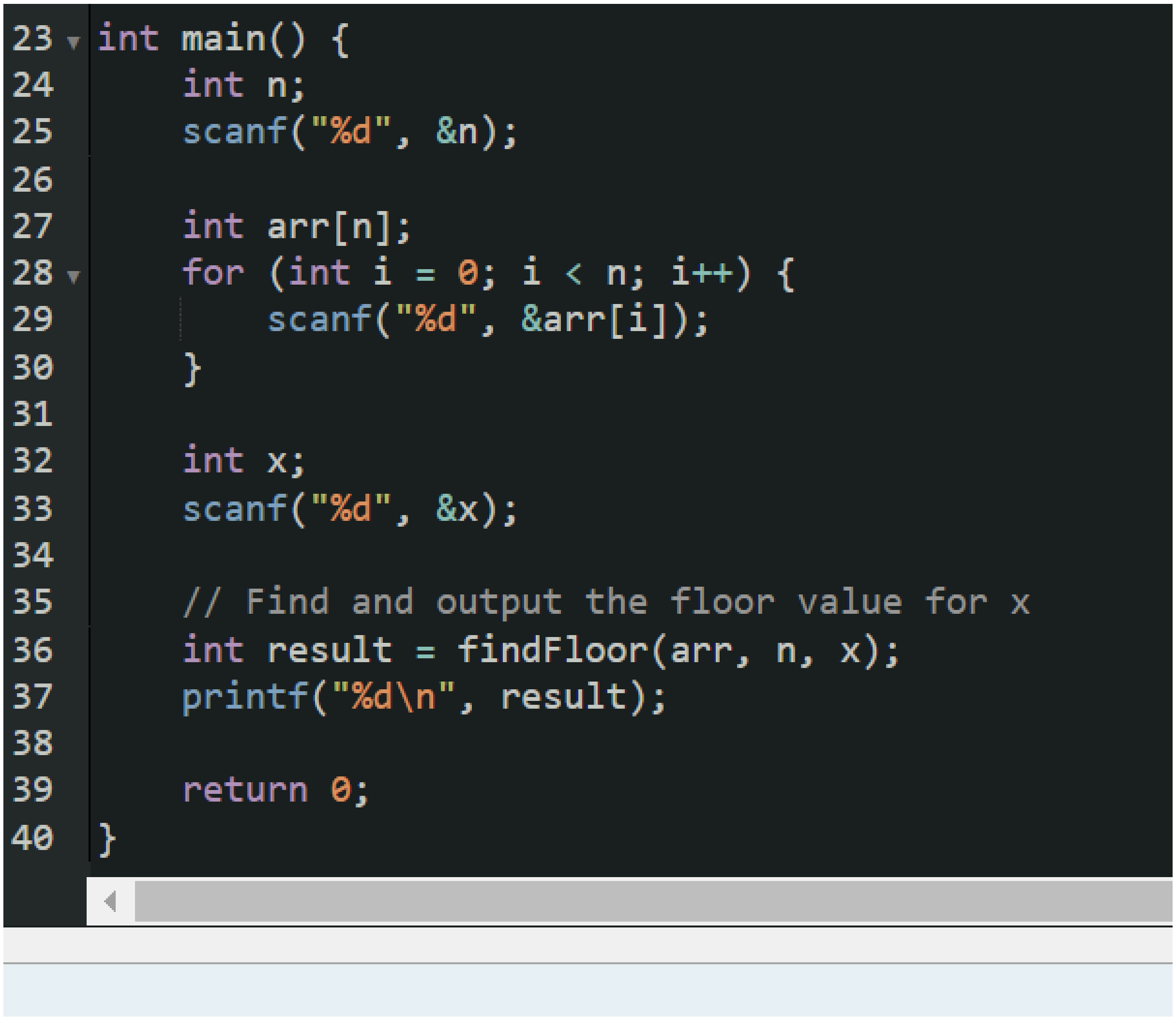
First Line Contains Integer n – Size of array
Next n lines Contains n numbers – Elements of an array
Last Line Contains Integer x – Value for x

Output Format

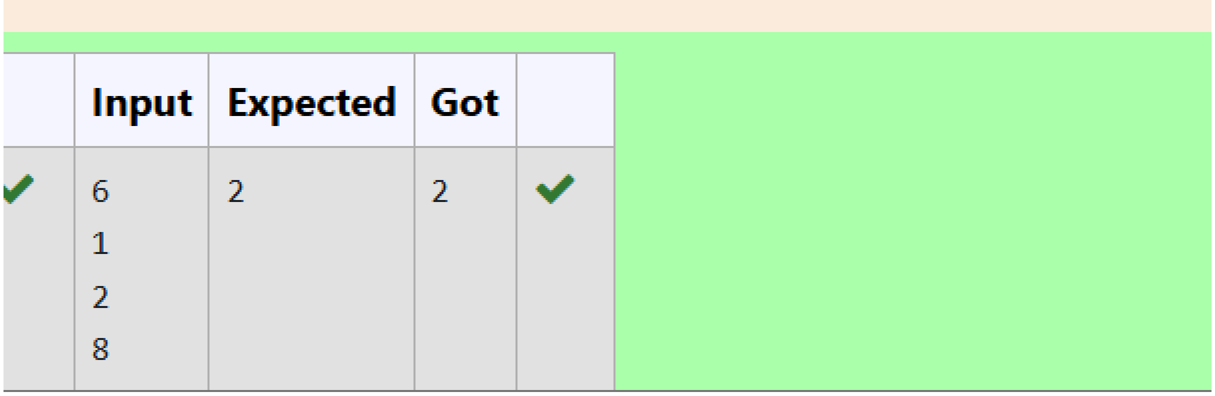
First Line Contains Integer – Floor value for x



```c
#include <stdio.h>

int findFloor(int arr[], int n, int x) {
    int low = 0, high = n - 1;
    int floorValue = -1;  // Initialize floor value as -1 (not found)

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

        if (arr[mid] == x) {
            return arr[mid];  // Found the exact match
        } else if (arr[mid] < x) {
            floorValue = arr[mid];  // Potential floor value
            low = mid + 1;  // Search in the right half
        } else {
            high = mid - 1;  // Search in the left half
        }
    }

    return floorValue;  // Return the floor value found
}
```

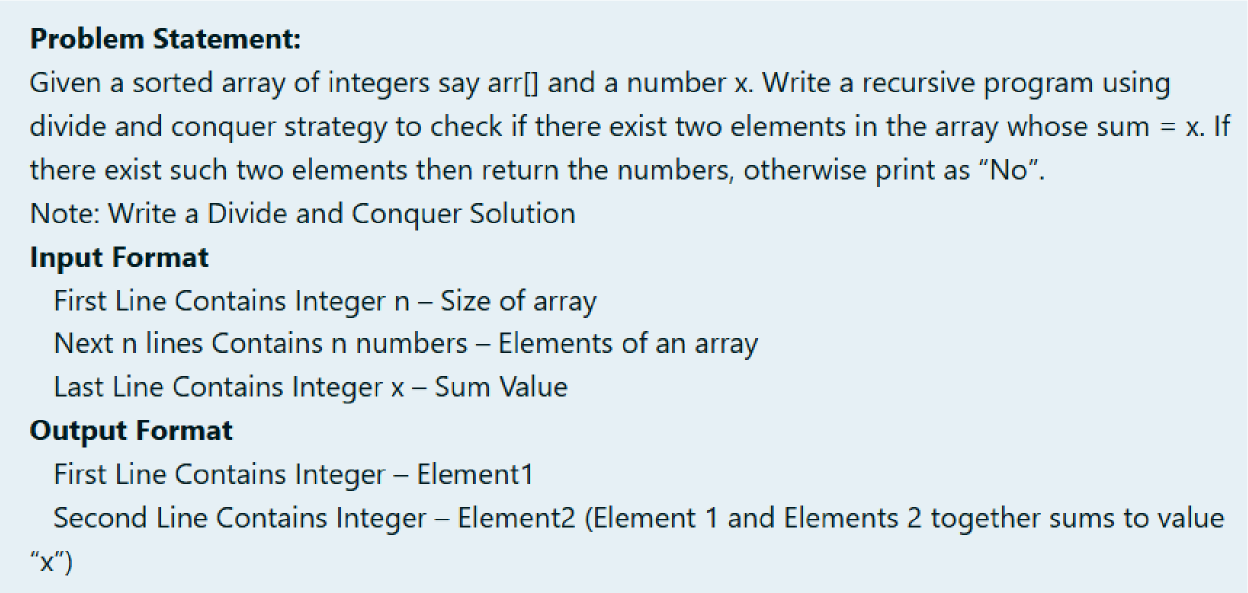


```c
int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int x;
    scanf("%d", &x);

    // Find and output the floor value for x
    int result = findFloor(arr, n, x);
    printf("%d\n", result);

    return 0;
}
```



	Input	Expected	Got	
✓	6 1 2 8	2	2	✓

4. Two elements sum to x



Problem Statement:

Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as “No”.

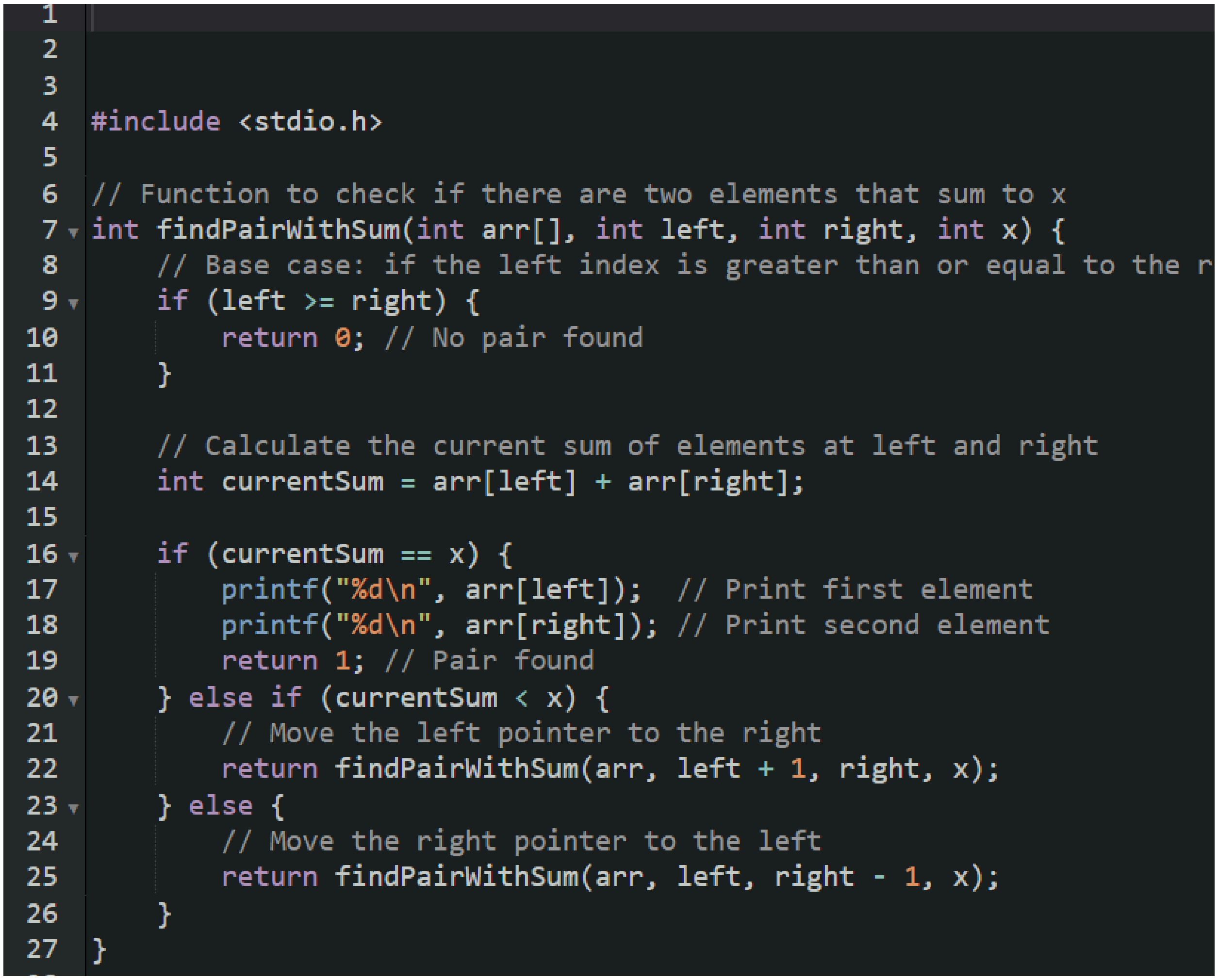
Note: Write a Divide and Conquer Solution

Input Format

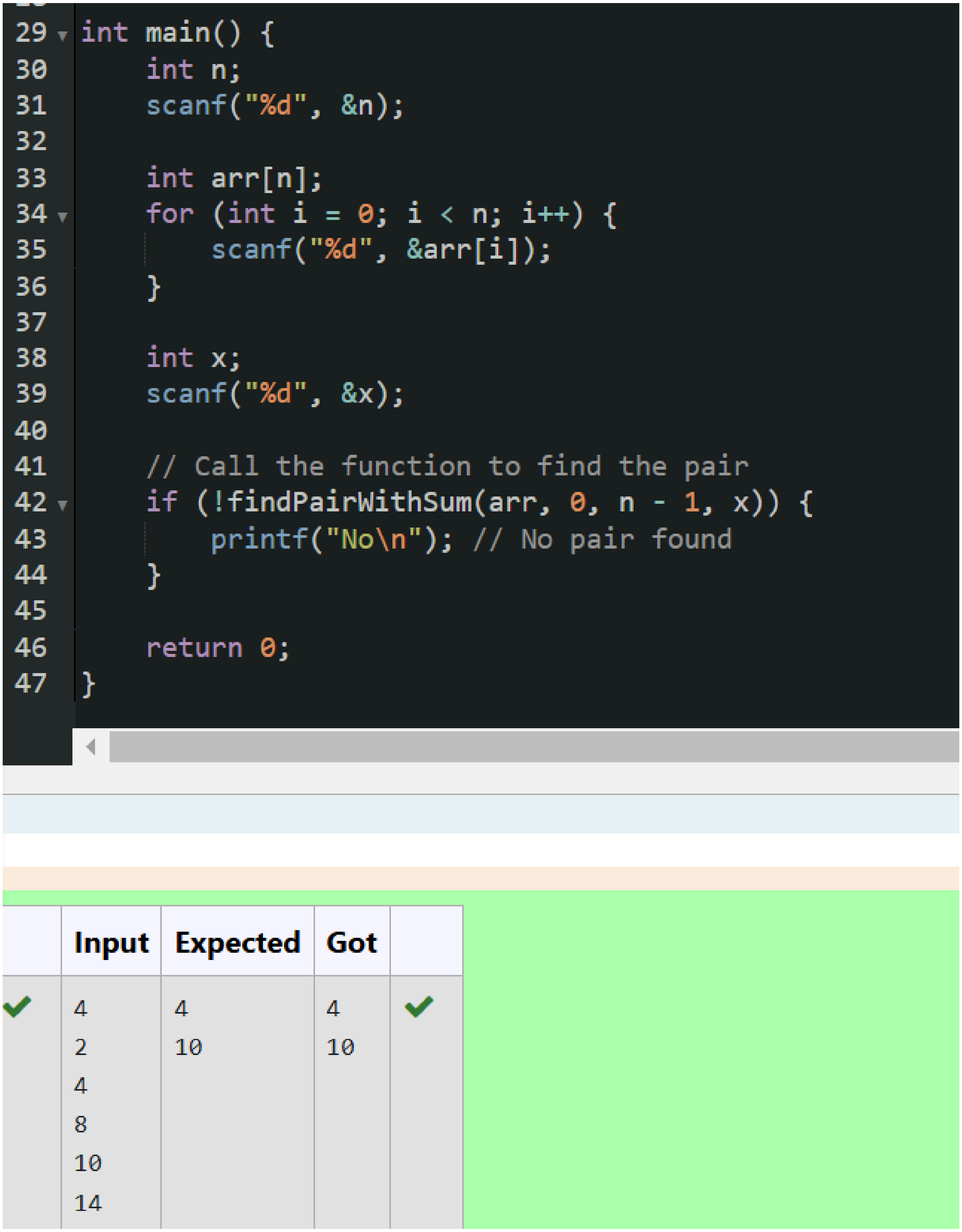
First Line Contains Integer n – Size of array
Next n lines Contains n numbers – Elements of an array
Last Line Contains Integer x – Sum Value

Output Format

First Line Contains Integer – Element1
Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

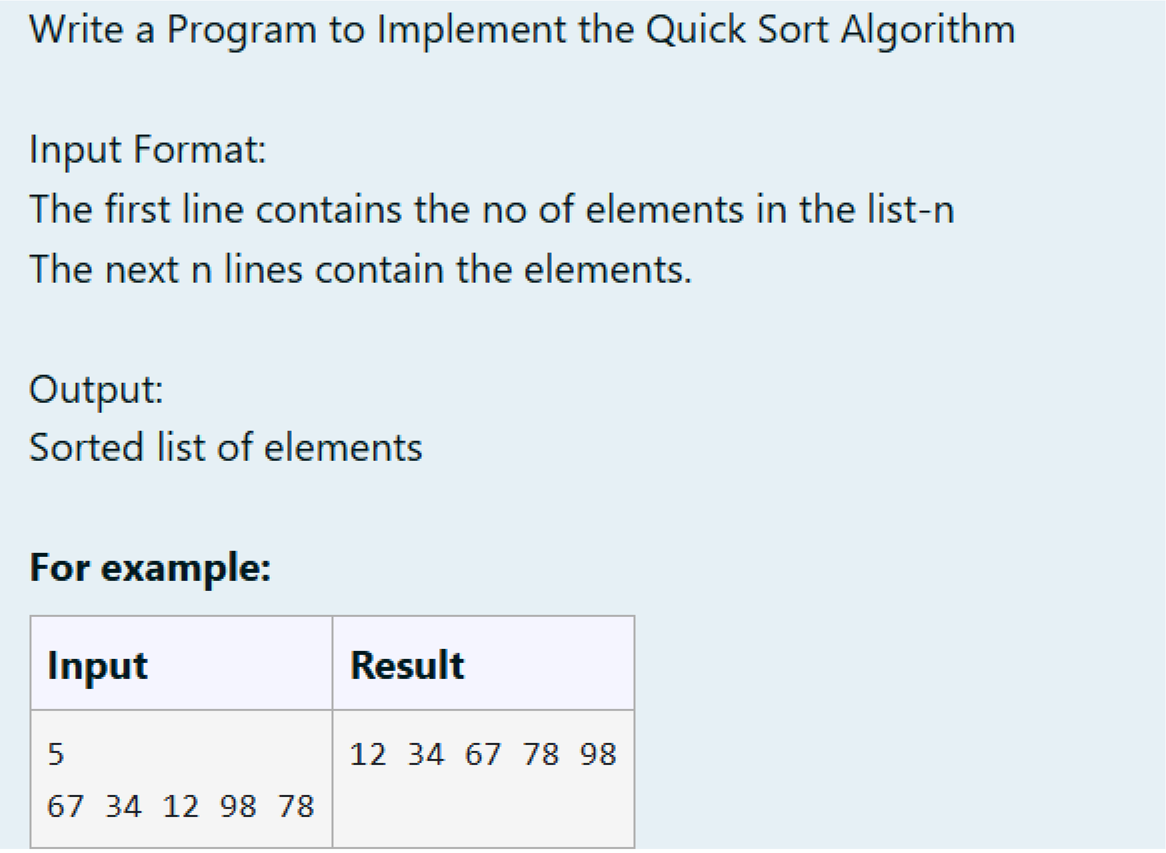


```c
#include <stdio.h>

// Function to check if there are two elements that sum to x
int findPairWithSum(int arr[], int left, int right, int x) {
    // Base case: if the left index is greater than or equal to the r
    if (left >= right) {
        return 0; // No pair found
    }

    // Calculate the current sum of elements at left and right
    int currentSum = arr[left] + arr[right];

    if (currentSum == x) {
        printf("%d\n", arr[left]);  // Print first element
        printf("%d\n", arr[right]); // Print second element
        return 1; // Pair found
    } else if (currentSum < x) {
        // Move the left pointer to the right
        return findPairWithSum(arr, left + 1, right, x);
    } else {
        // Move the right pointer to the left
        return findPairWithSum(arr, left, right - 1, x);
    }
}
```



```c
int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int x;
    scanf("%d", &x);

    // Call the function to find the pair
    if (!findPairWithSum(arr, 0, n - 1, x)) {
        printf("No\n"); // No pair found
    }

    return 0;
}
```

	Input	Expected	Got	
✓	4 2 4 8 10 14	4 10	4 10	✓



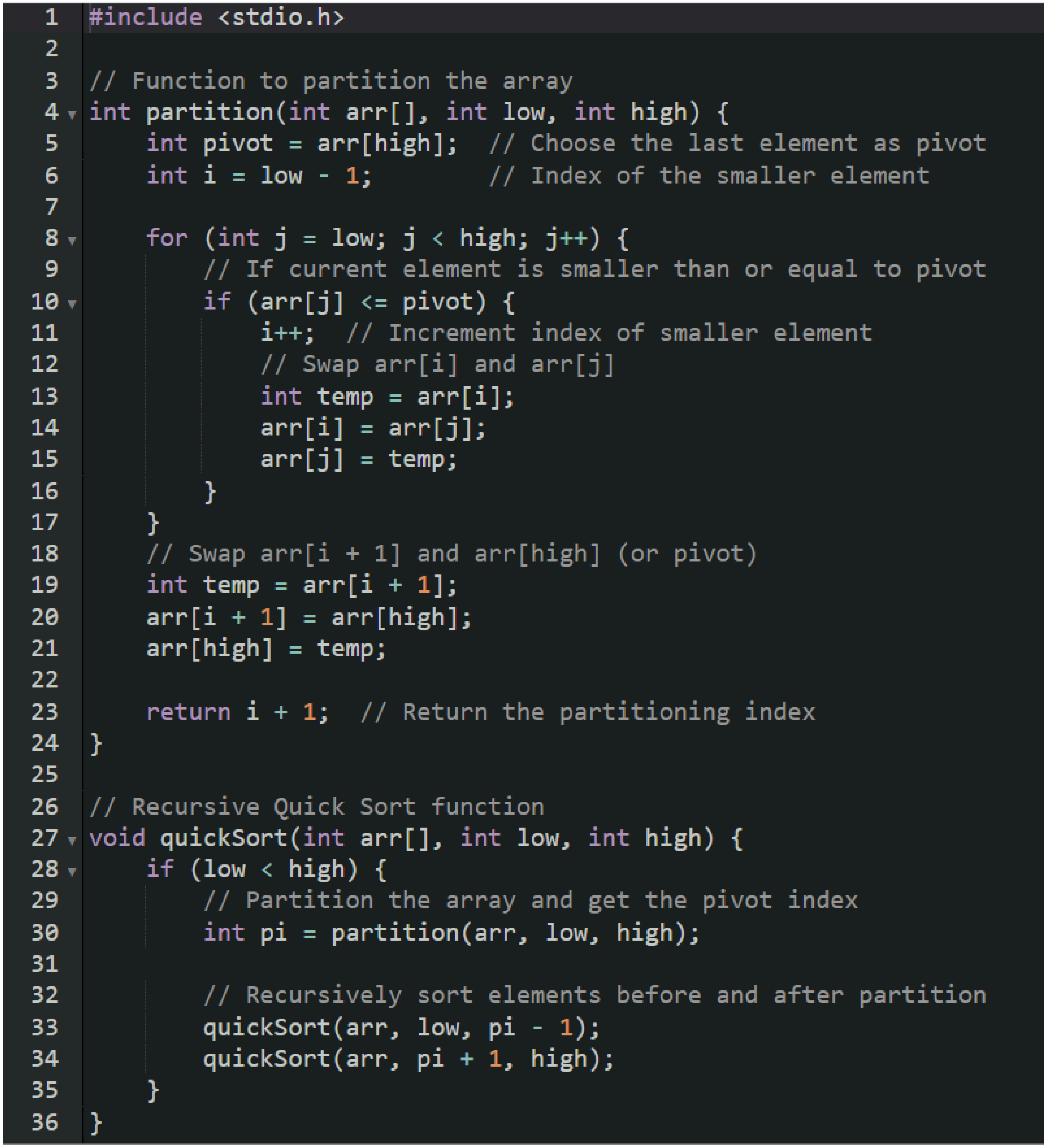
Write a Program to Implement the Quick Sort Algorithm

Input Format:
The first line contains the no of elements in the list-n
The next n lines contain the elements.

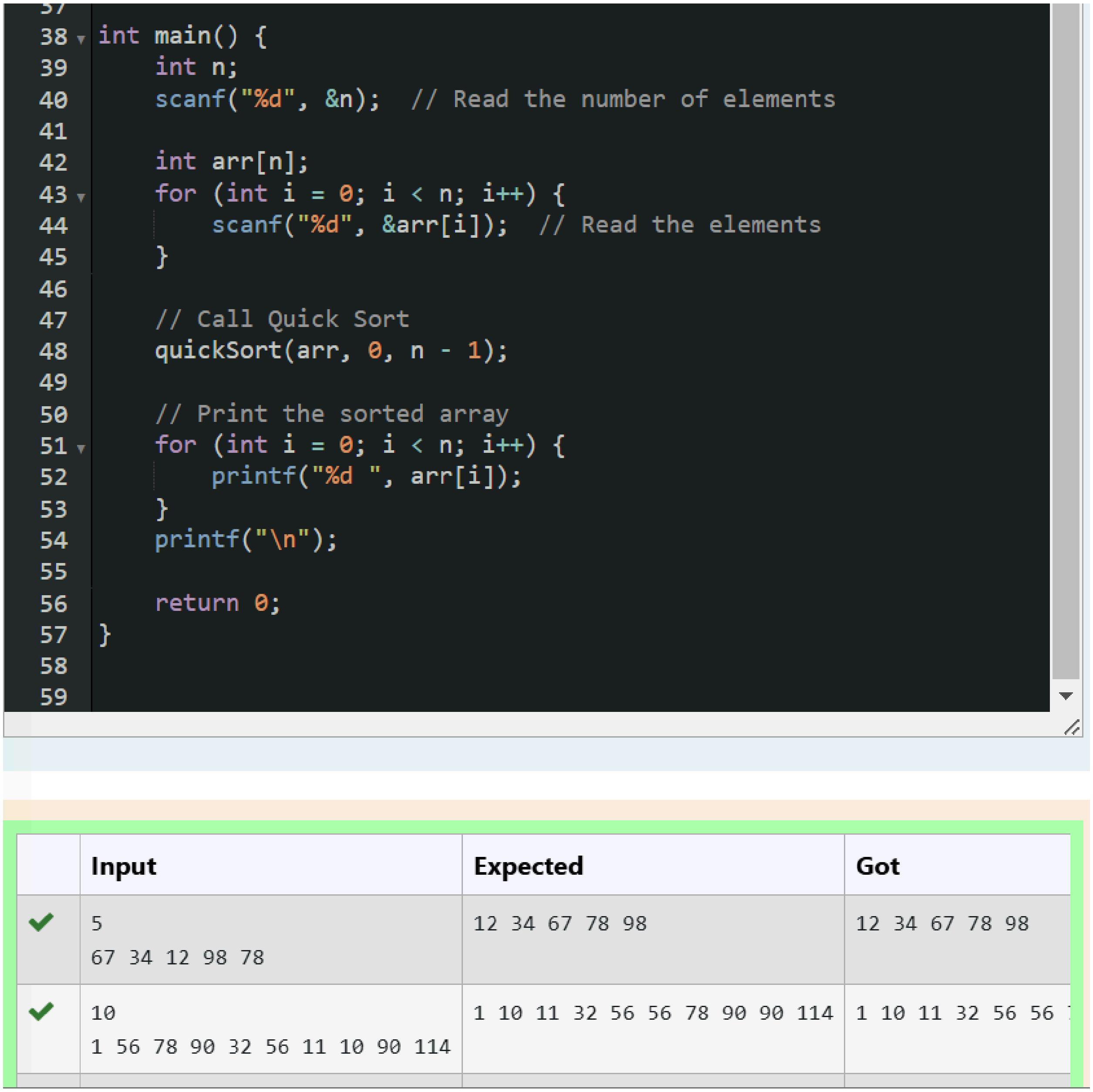
Output:
Sorted list of elements

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

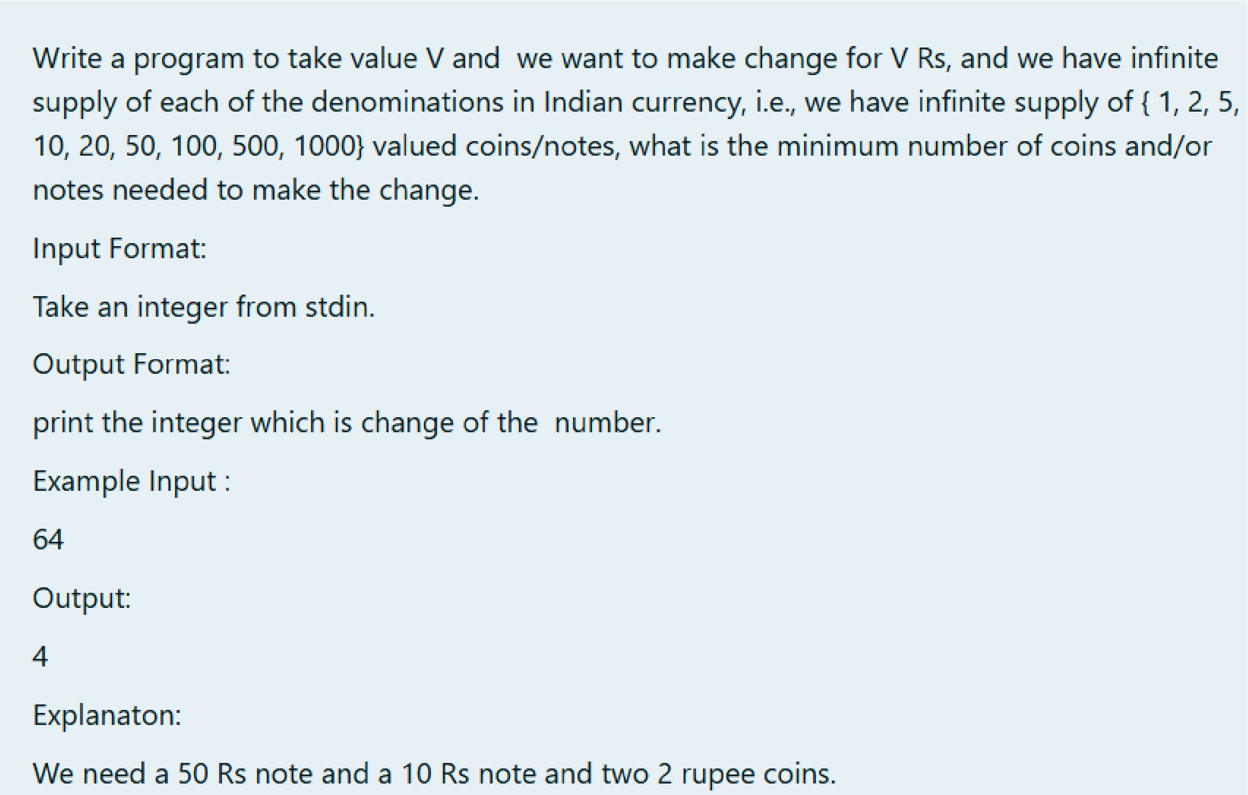


```c
#include <stdio.h>

// Function to partition the array
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // Choose the last element as pivot
    int i = low - 1;        // Index of the smaller element

    for (int j = low; j < high; j++) {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= pivot) {
            i++;  // Increment index of smaller element
            // Swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Swap arr[i + 1] and arr[high] (or pivot)
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;  // Return the partitioning index
}

// Recursive Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array and get the pivot index
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```



```c
int main() {
    int n;
    scanf("%d", &n);  // Read the number of elements

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);  // Read the elements
    }

    // Call Quick Sort
    quickSort(arr, 0, n - 1);

    // Print the sorted array
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

	Input	Expected	Got
✓	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98
✓	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56



Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

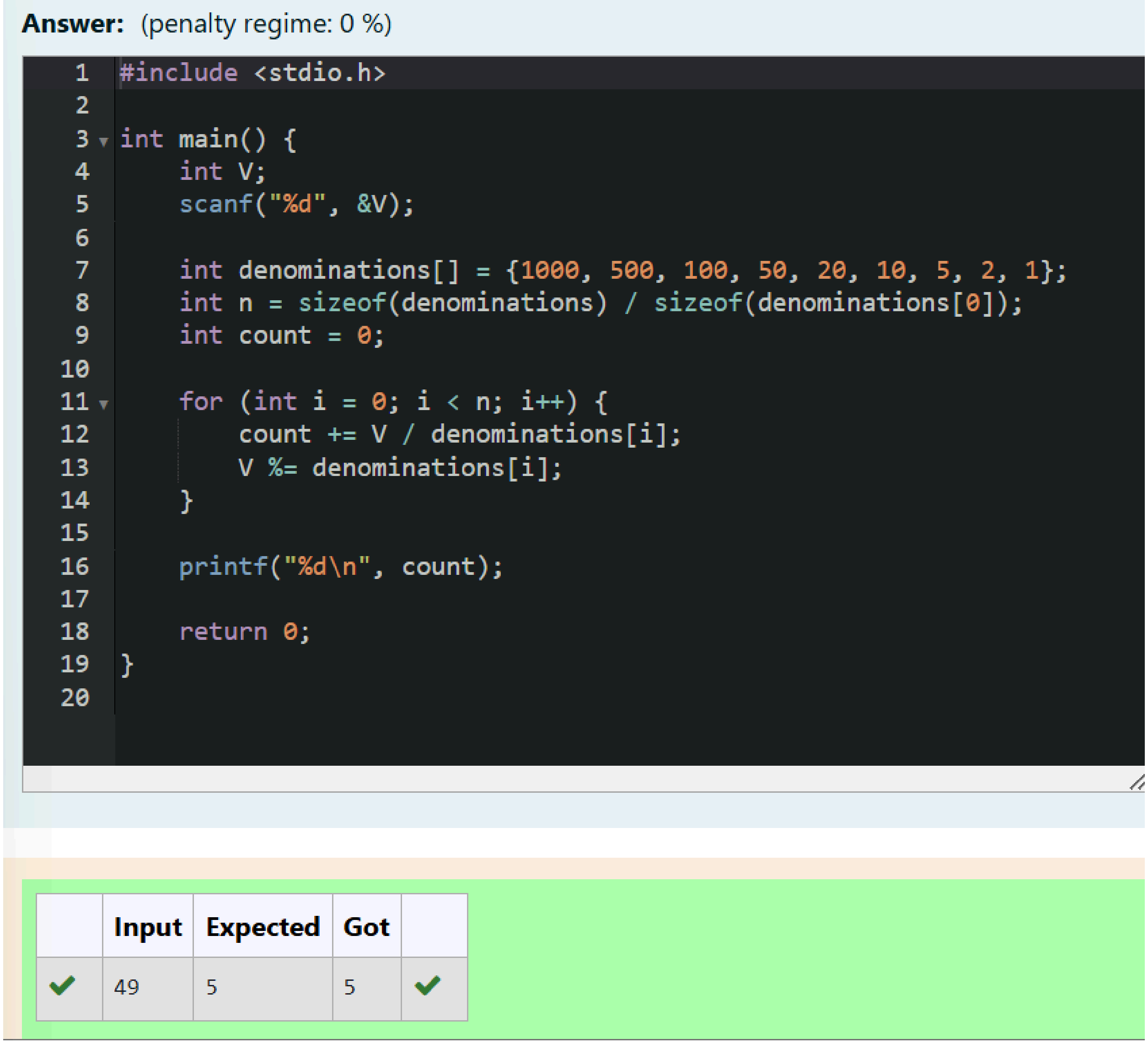
64

Output:

4

Explanaton:

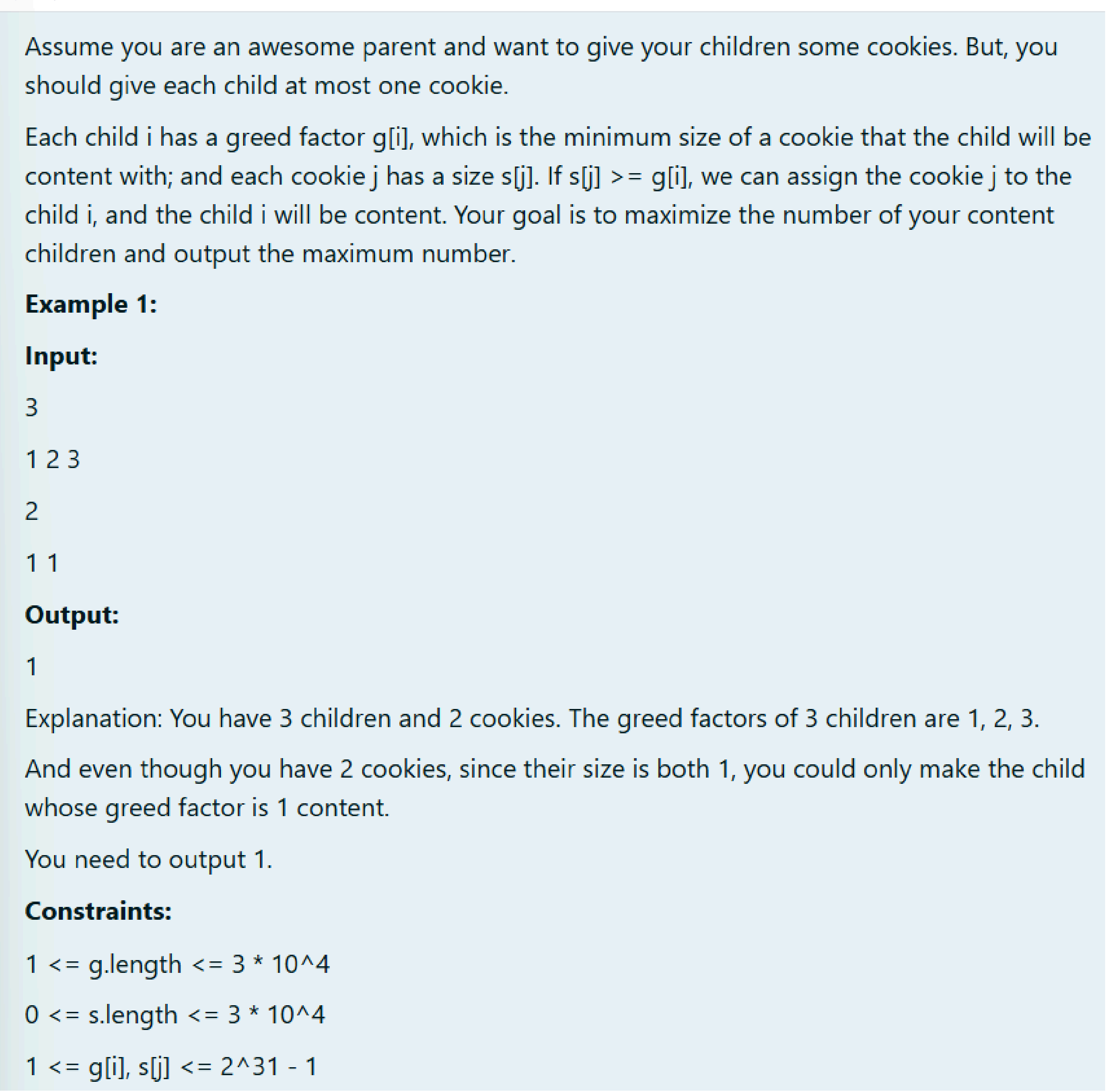
We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.



Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

int main() {
    int V;
    scanf("%d", &V);

    int denominations[] = {1000, 500, 100, 50, 20, 10, 5, 2, 1};
    int n = sizeof(denominations) / sizeof(denominations[0]);
    int count = 0;

    for (int i = 0; i < n; i++) {
        count += V / denominations[i];
        V %= denominations[i];
    }

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

    return 0;
}

```

	Input	Expected	Got	
✓	49	5	5	✓



Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

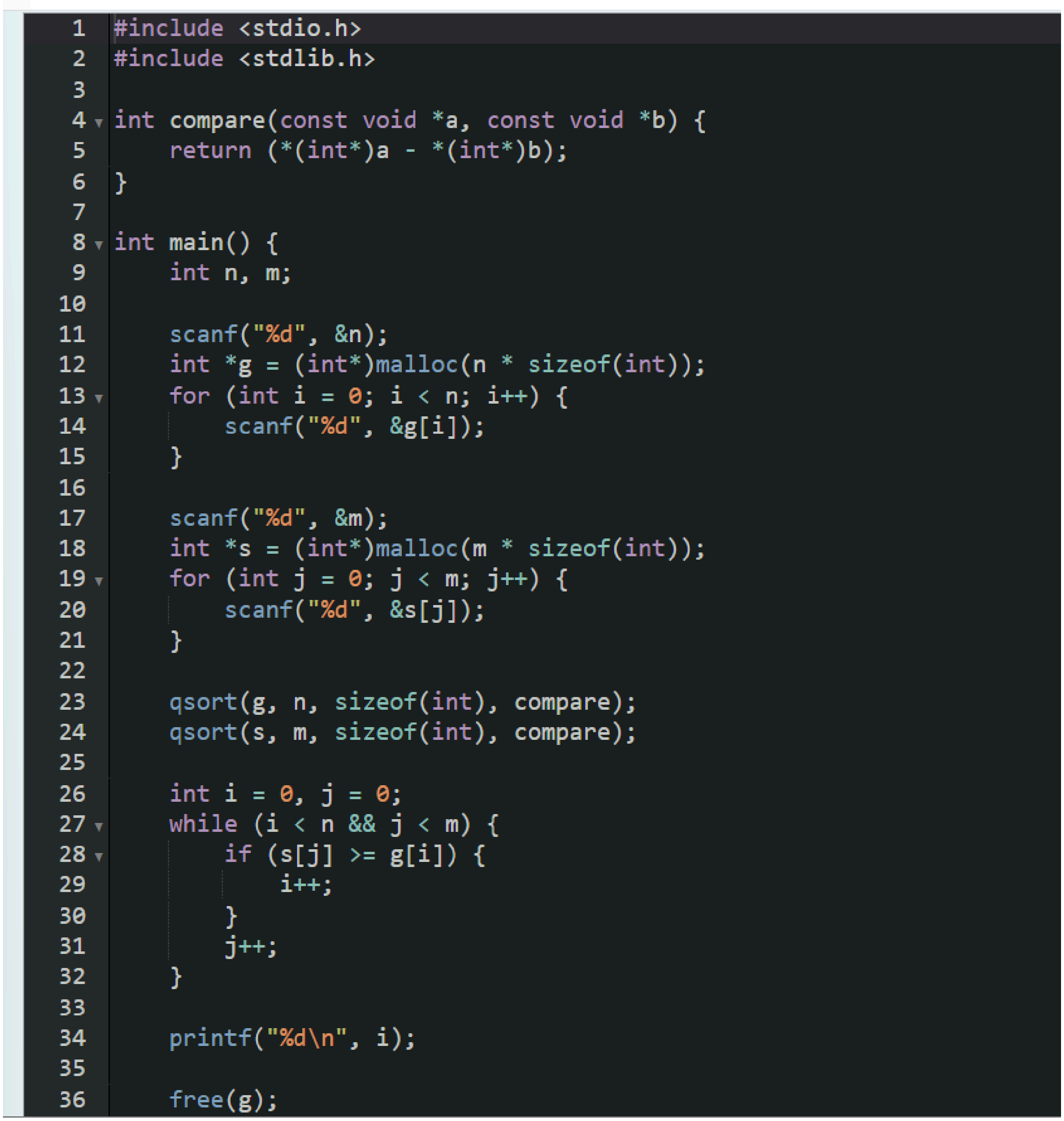
You need to output 1.

Constraints:

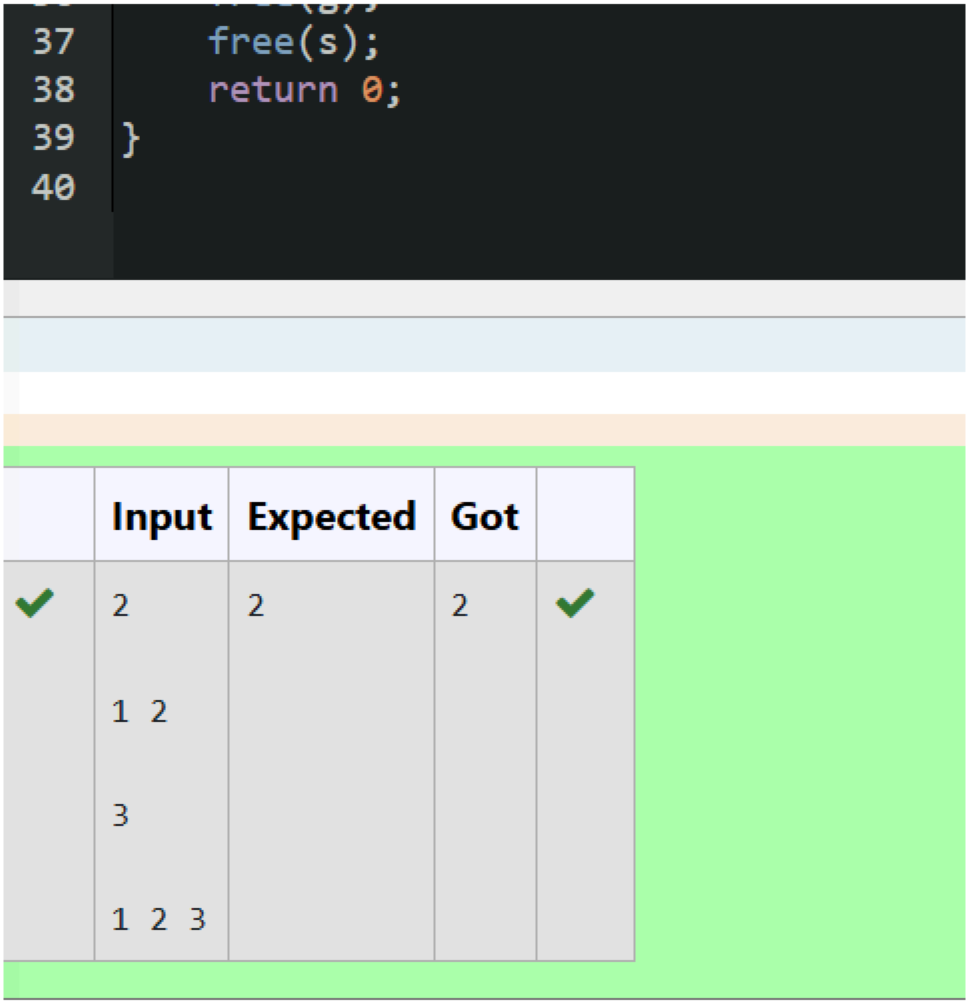
1 <= g.length <= 3 * 10^4

0 <= s.length <= 3 * 10^4

1 <= g[i], s[j] <= 2^31 - 1

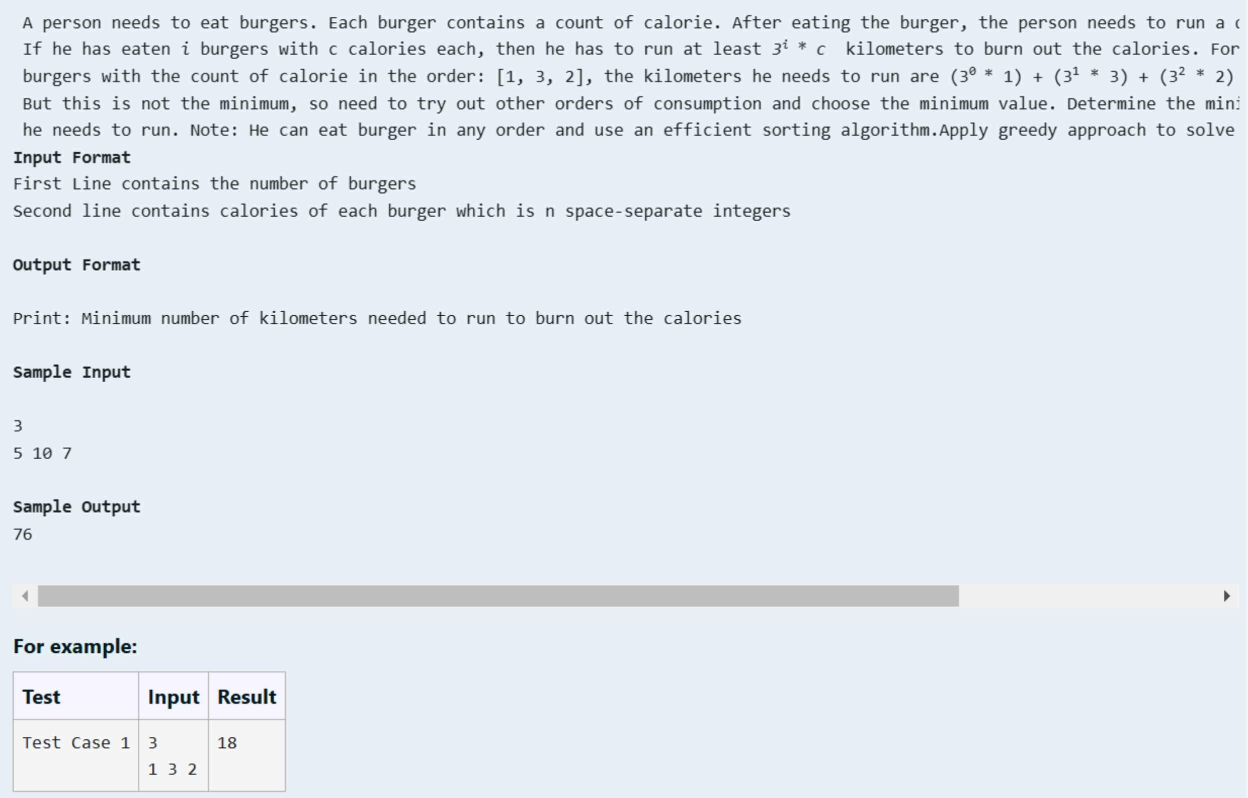


```c
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int n, m;

    scanf("%d", &n);
    int *g = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &g[i]);
    }

    scanf("%d", &m);
    int *s = (int*)malloc(m * sizeof(int));
    for (int j = 0; j < m; j++) {
        scanf("%d", &s[j]);
    }

    qsort(g, n, sizeof(int), compare);
    qsort(s, m, sizeof(int), compare);

    int i = 0, j = 0;
    while (i < n && j < m) {
        if (s[j] >= g[i]) {
            i++;
        }
        j++;
    }

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

    free(g);
```



```
    free(s);
    return 0;
}
```

	Input	Expected	Got	
✓	2 1 2 3 1 2 3	2	2	✓



A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a
If he has eaten i burgers with c calories each, then he has to run at least $3^i * c$ kilometers to burn out the calories. For
burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are $(3^0 * 1) + (3^1 * 3) + (3^2 * 2)$
But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the mini
he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve

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

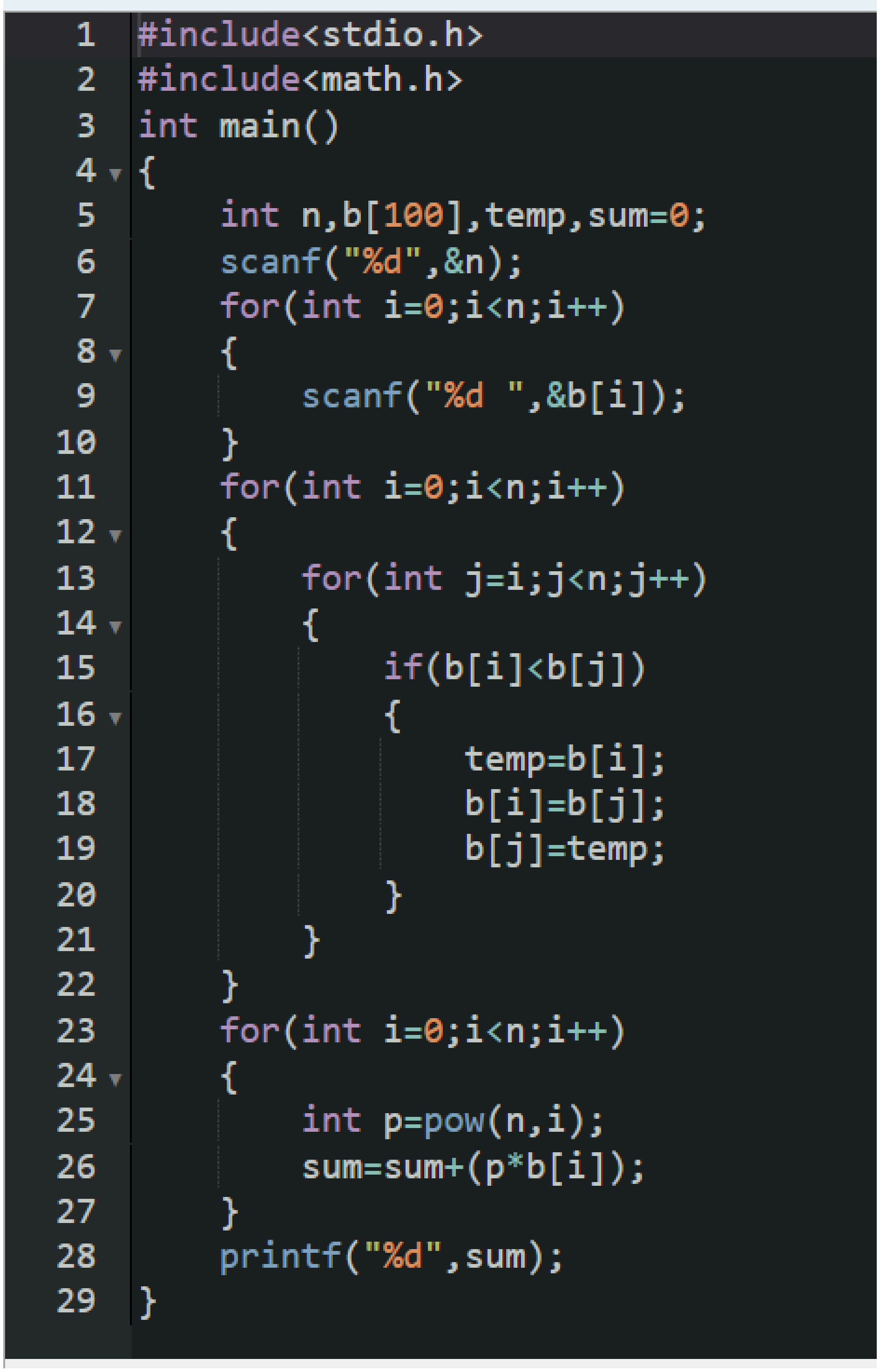
```
3
5 10 7
```

Sample Output

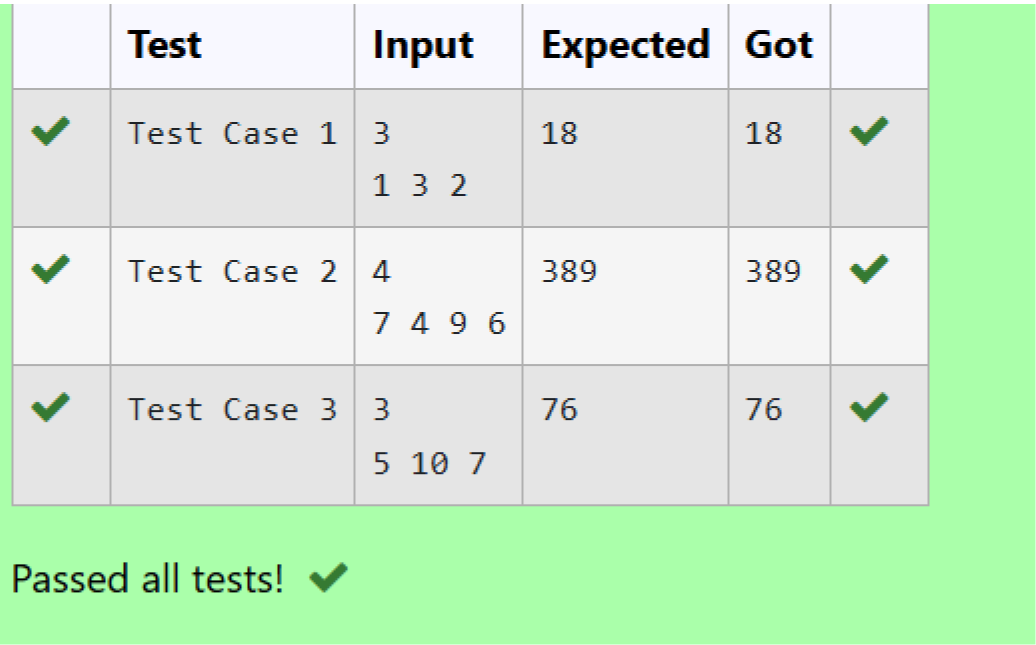
```
76
```

For example:

Test	Input	Result
Test Case 1	3 1 3 2	18



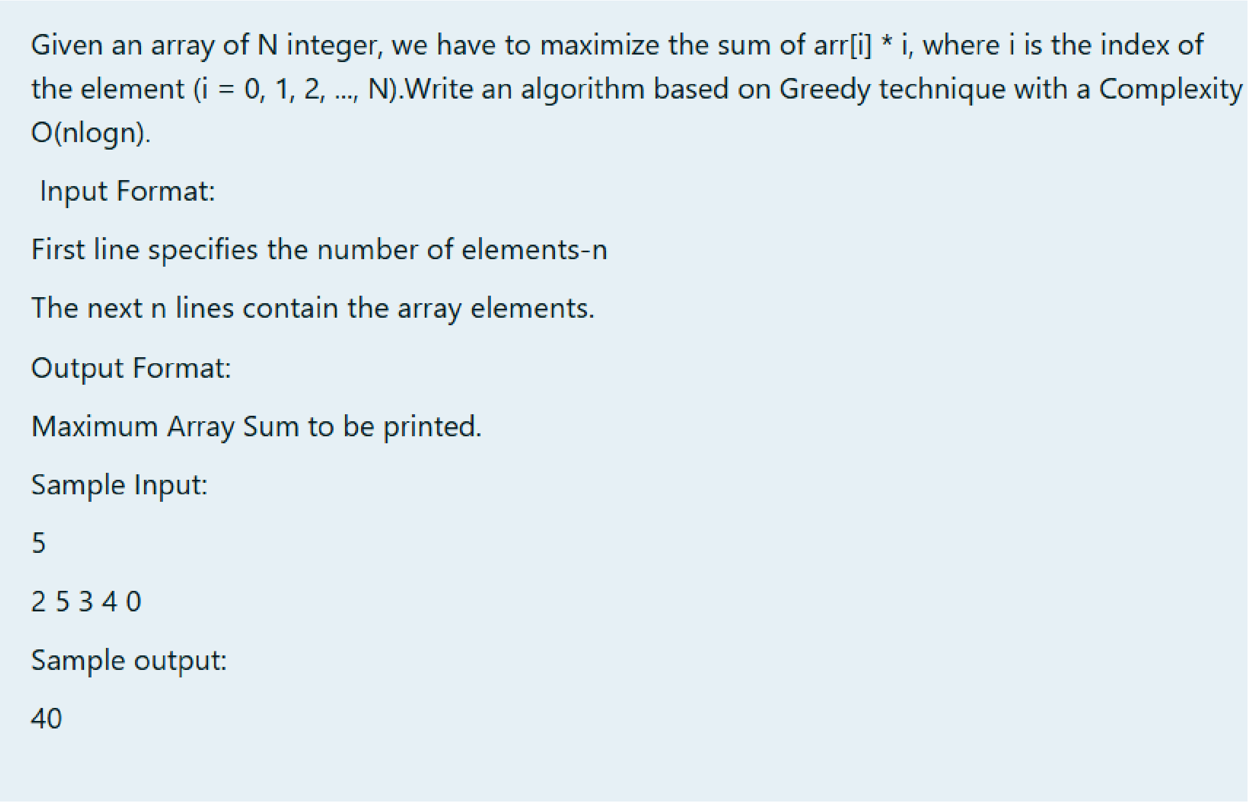
```c
#include<stdio.h>
#include<math.h>
int main()
{
    int n,b[100],temp,sum=0;
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d ",&b[i]);
    }
    for(int i=0;i<n;i++)
    {
        for(int j=i;j<n;j++)
        {
            if(b[i]<b[j])
            {
                temp=b[i];
                b[i]=b[j];
                b[j]=temp;
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        int p=pow(n,i);
        sum=sum+(p*b[i]);
    }
    printf("%d",sum);
}
```



	Test	Input	Expected	Got	
✓	Test Case 1	3 1 3 2	18	18	✓
✓	Test Case 2	4 7 4 9 6	389	389	✓
✓	Test Case 3	3 5 10 7	76	76	✓

Passed all tests! ✓

Array sum max problem



Given an array of N integer, we have to maximize the sum of arr[i] * i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

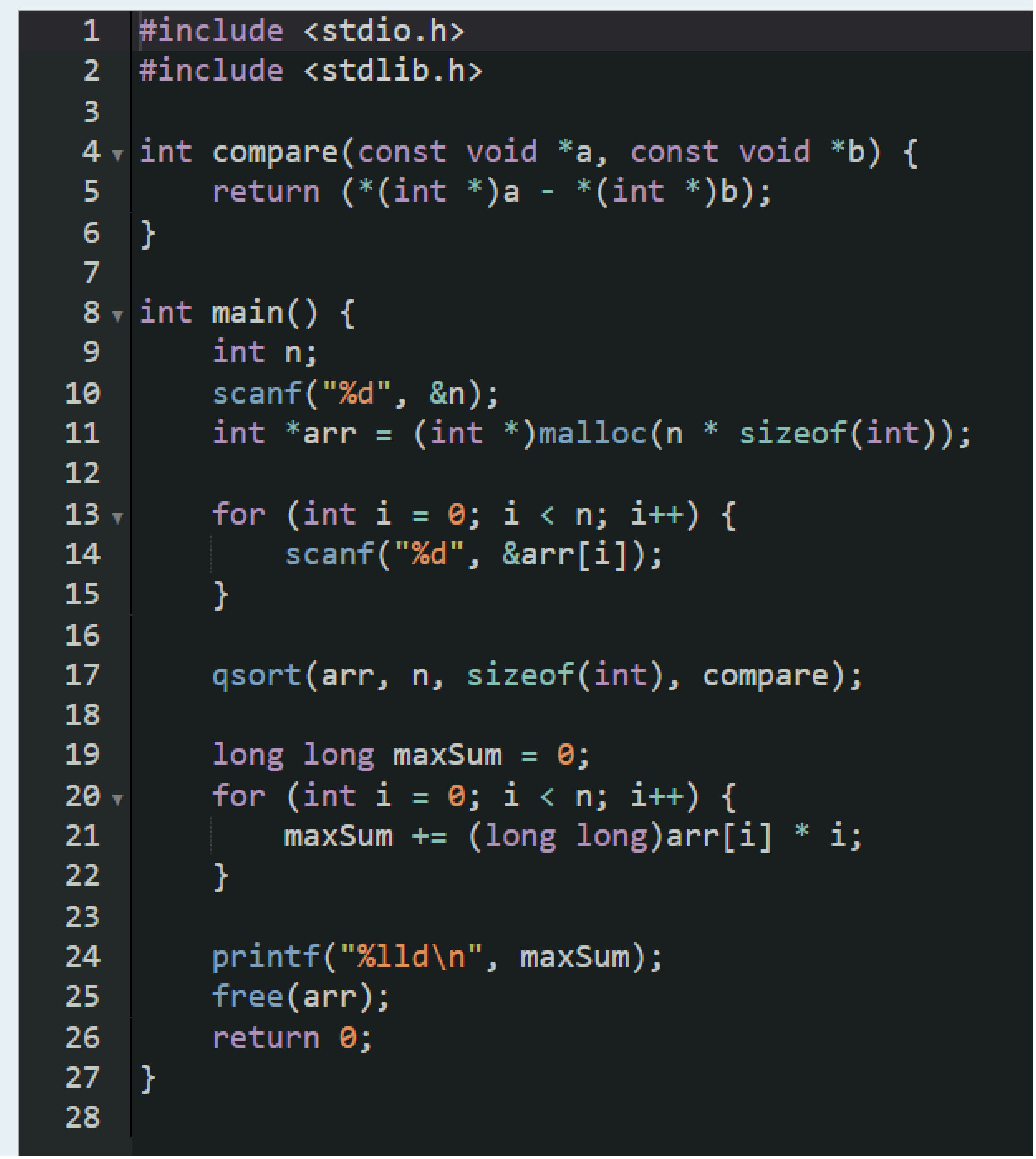
Sample Input:

5

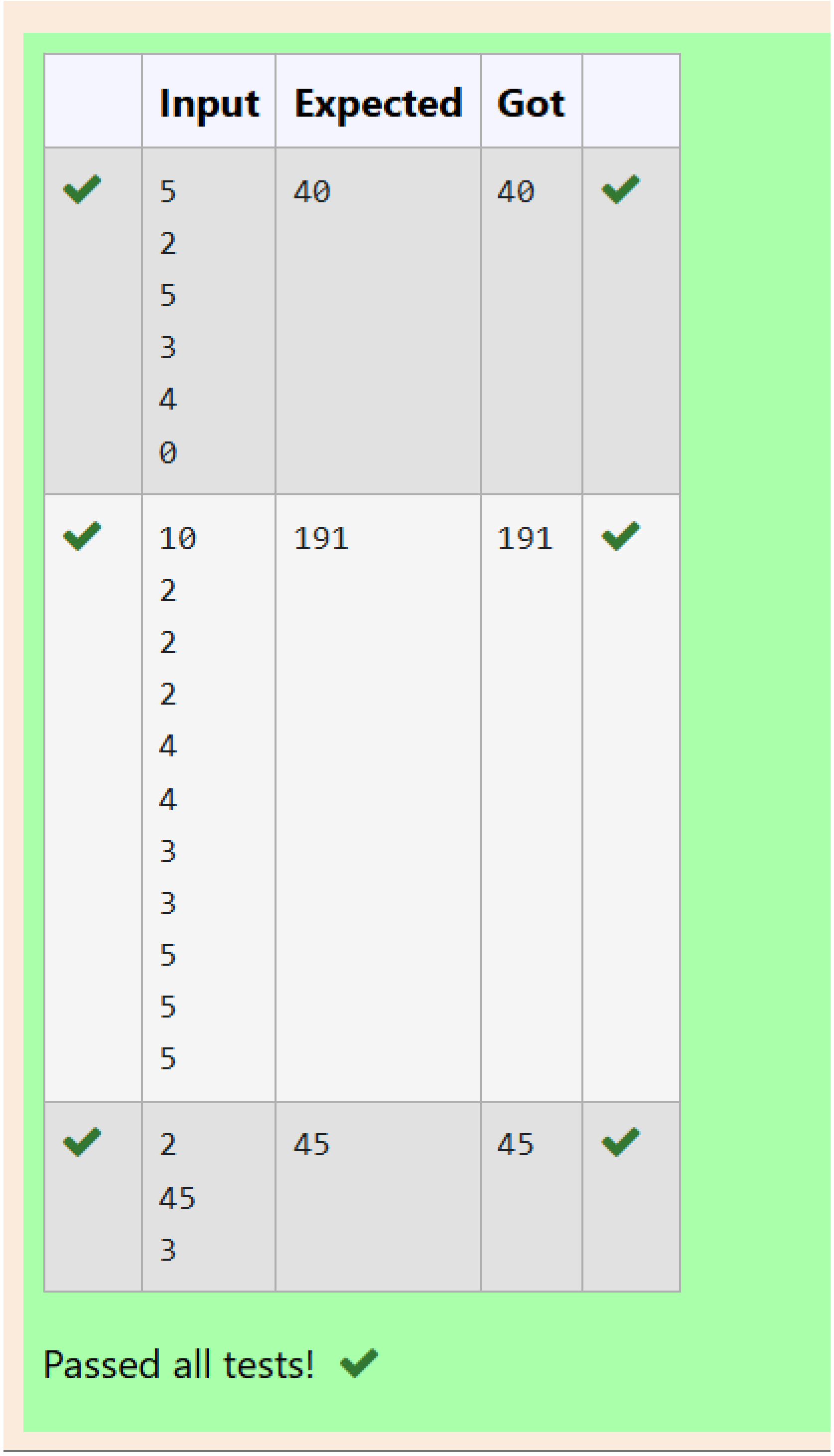
2 5 3 4 0

Sample output:

40



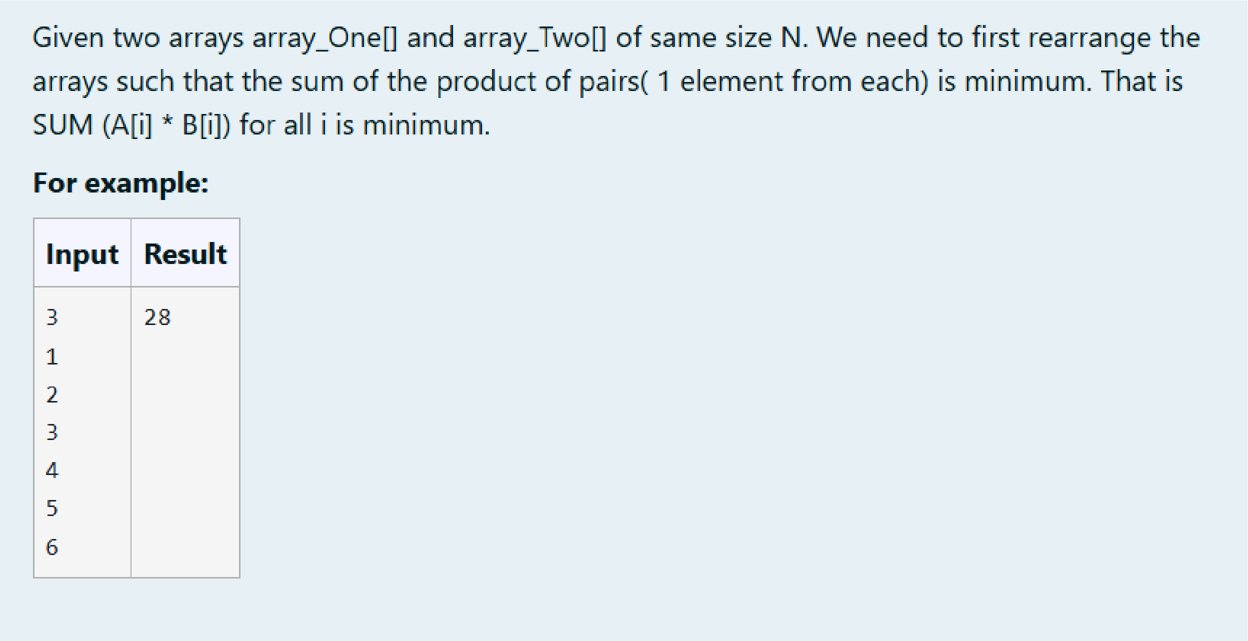
```c
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    int n;
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    qsort(arr, n, sizeof(int), compare);

    long long maxSum = 0;
    for (int i = 0; i < n; i++) {
        maxSum += (long long)arr[i] * i;
    }

    printf("%lld\n", maxSum);
    free(arr);
    return 0;
}
```



	Input	Expected	Got	
✓	5 2 5 3 4 0	40	40	✓
✓	10 2 2 2 4 4 3 3 5 5 5	191	191	✓
✓	2 45 3	45	45	✓

Passed all tests! ✓

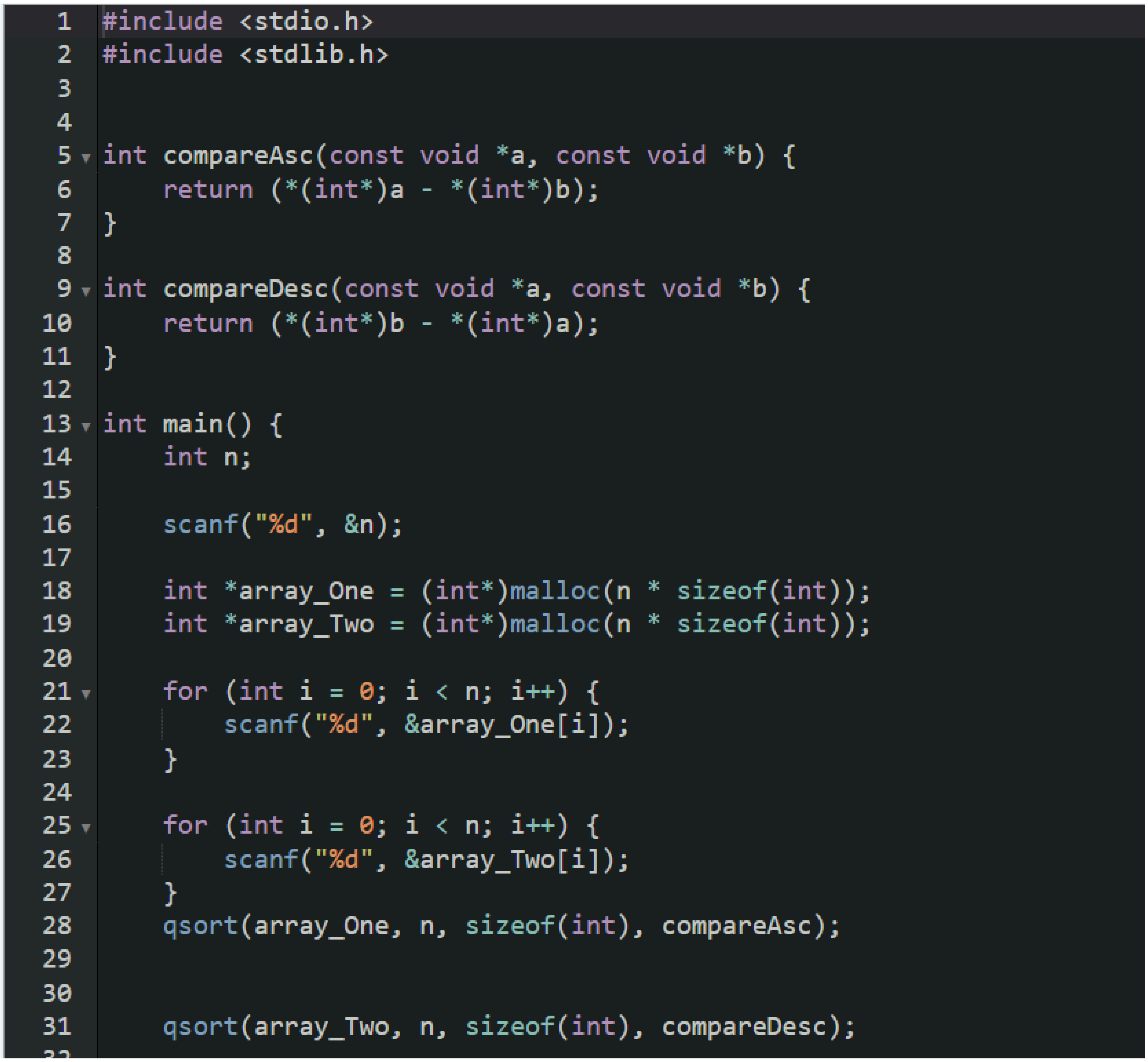
Product of array elements minimum



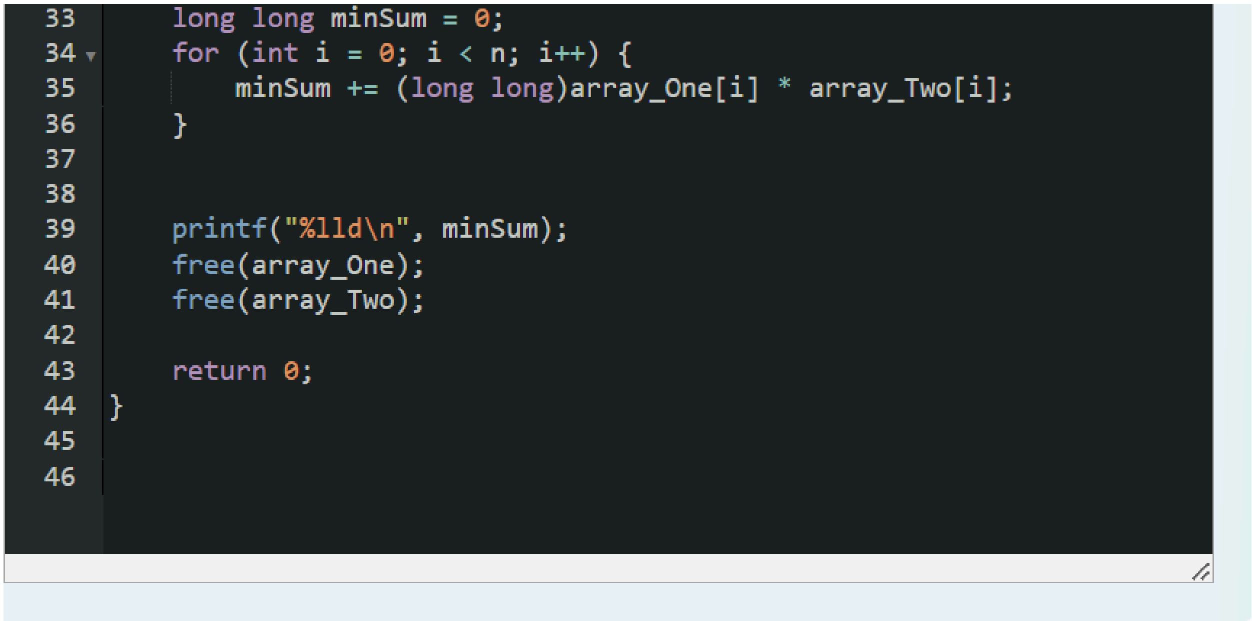
Given two arrays array_One[] and array_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs(1 element from each) is minimum. That is SUM (A[i] * B[i]) for all i is minimum.

For example:

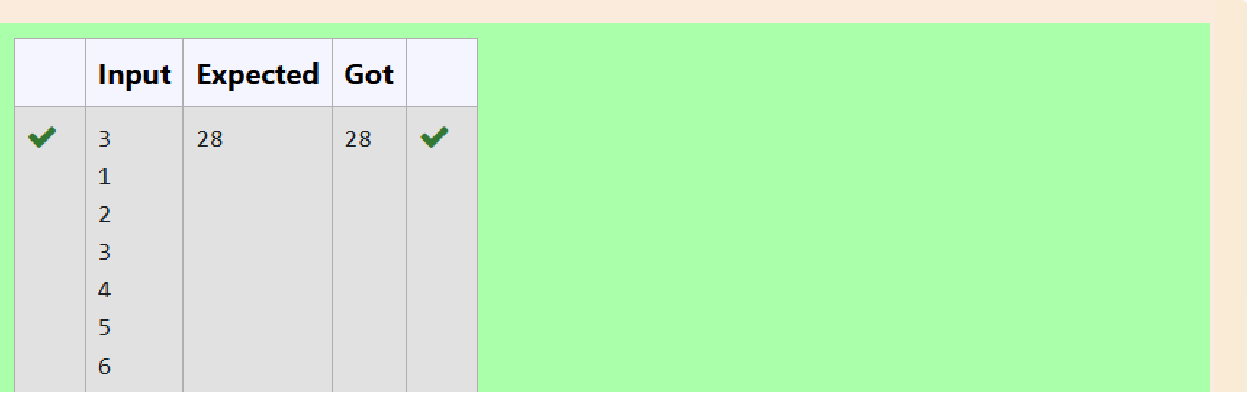
Input	Result
3 1 2 3 4 5 6	28



```c
#include <stdio.h>
#include <stdlib.h>


int compareAsc(const void *a, const void *b) {
    return (*(int*)a - *(int*)b);
}

int compareDesc(const void *a, const void *b) {
    return (*(int*)b - *(int*)a);
}

int main() {
    int n;

    scanf("%d", &n);

    int *array_One = (int*)malloc(n * sizeof(int));
    int *array_Two = (int*)malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        scanf("%d", &array_One[i]);
    }

    for (int i = 0; i < n; i++) {
        scanf("%d", &array_Two[i]);
    }
    qsort(array_One, n, sizeof(int), compareAsc);


    qsort(array_Two, n, sizeof(int), compareDesc);
```

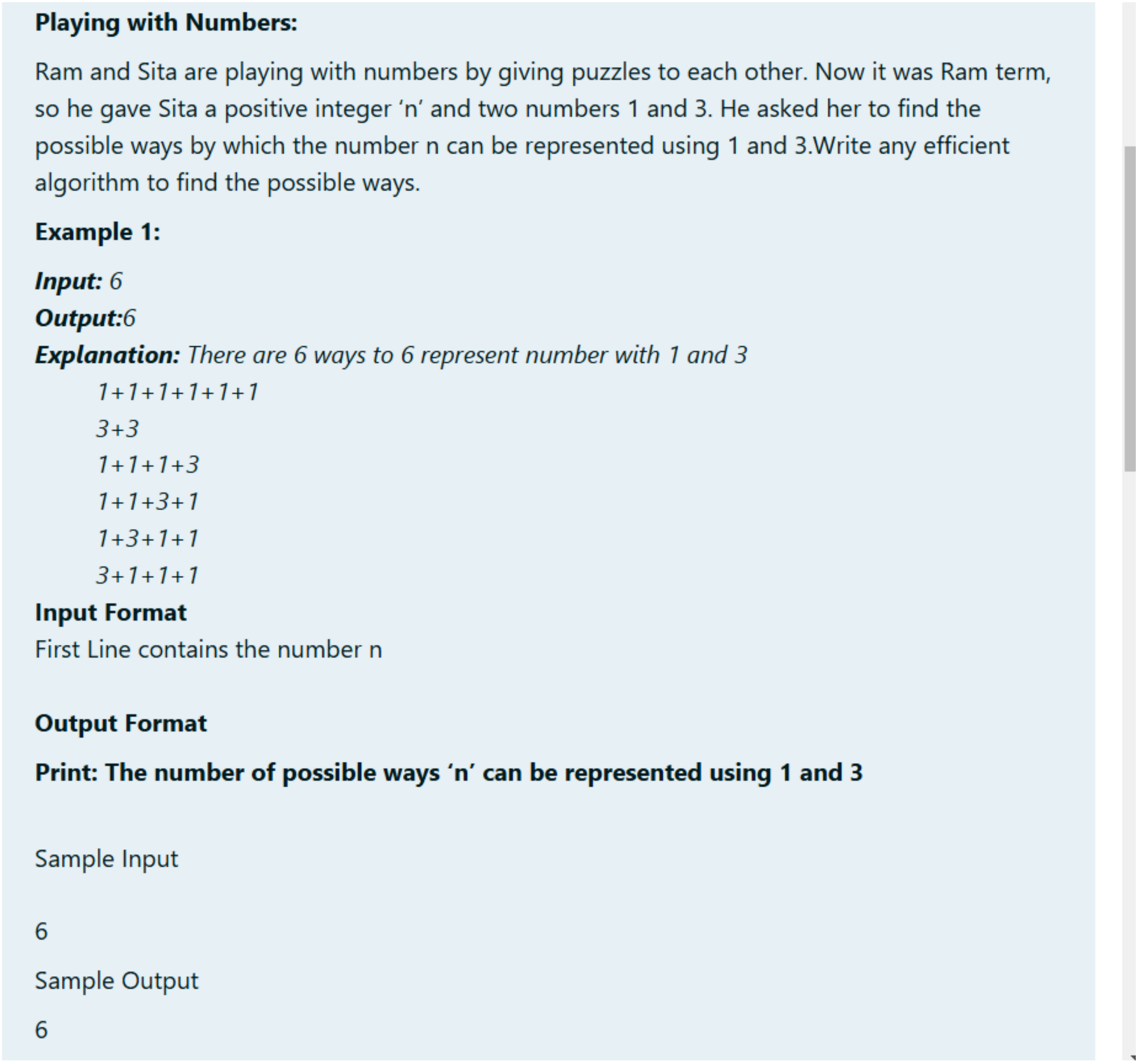


```c
    long long minSum = 0;
    for (int i = 0; i < n; i++) {
        minSum += (long long)array_One[i] * array_Two[i];
    }


    printf("%lld\n", minSum);
    free(array_One);
    free(array_Two);

    return 0;
}

```



	Input	Expected	Got	
✓	3 1 2 3 4 5 6	28	28	✓

DYNAMIC PROGRAMMING



Playing with Numbers:

Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer ‘n’ and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

Input: *6*

Output:*6*

Explanation: *There are 6 ways to 6 represent number with 1 and 3*

1+1+1+1+1+1
3+3
1+1+1+3
1+1+3+1
1+3+1+1
3+1+1+1

Input Format

First Line contains the number n

Output Format

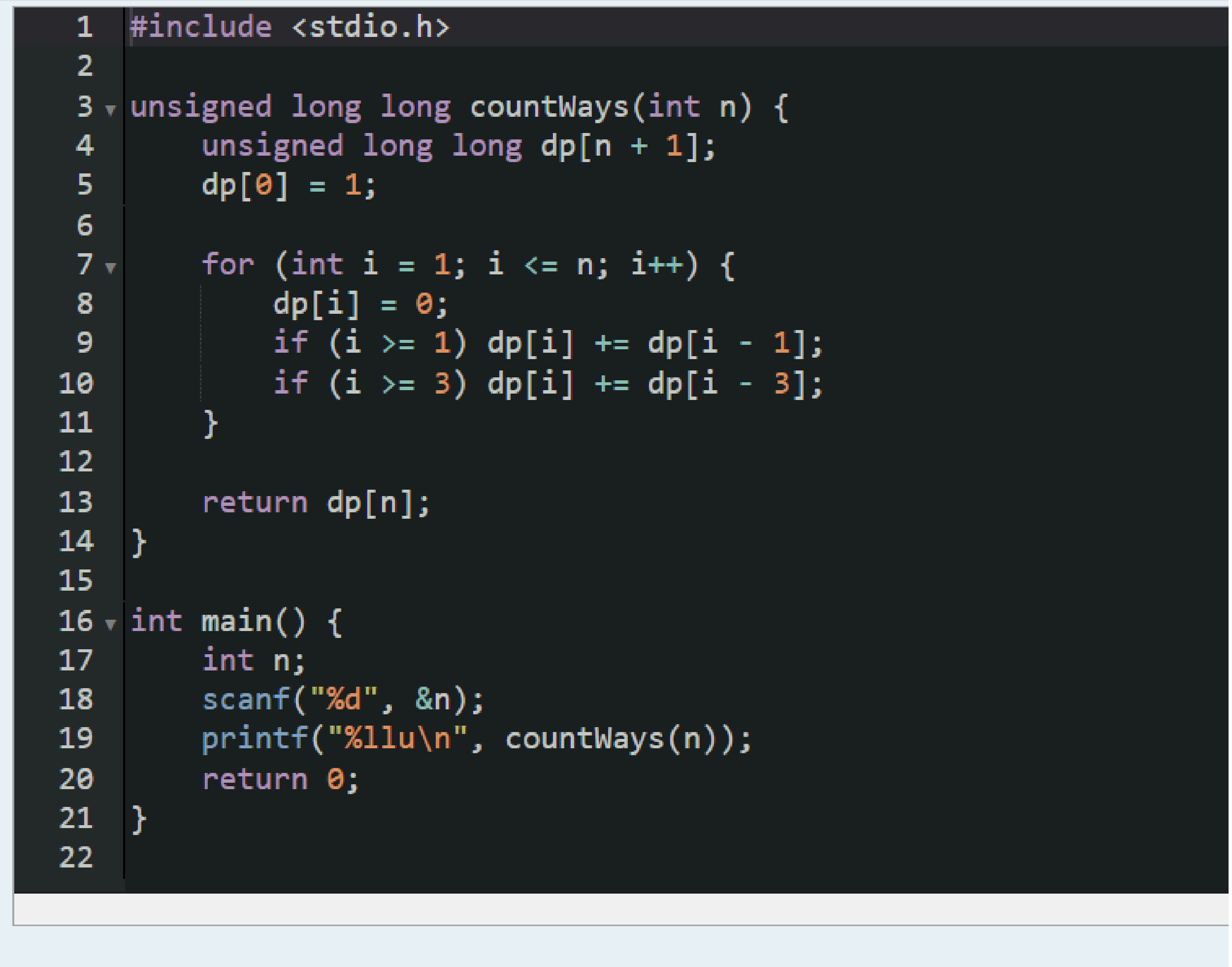
Print: The number of possible ways ‘n’ can be represented using 1 and 3

Sample Input

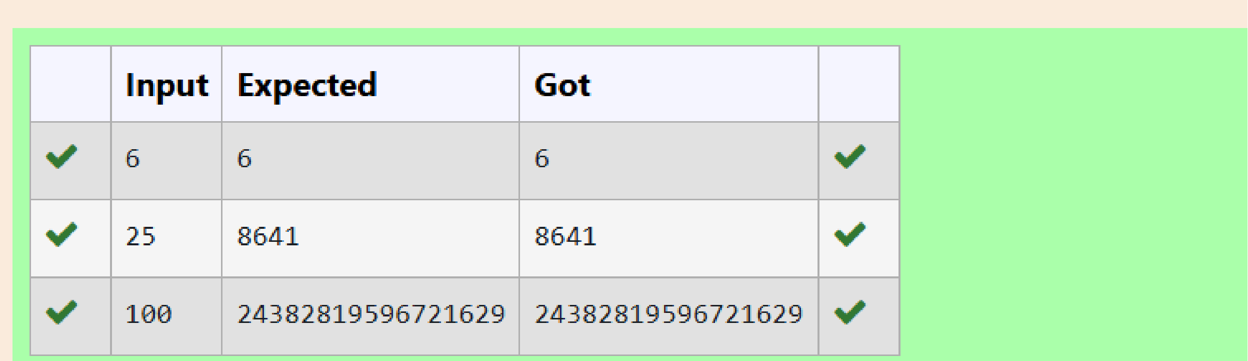
6

Sample Output

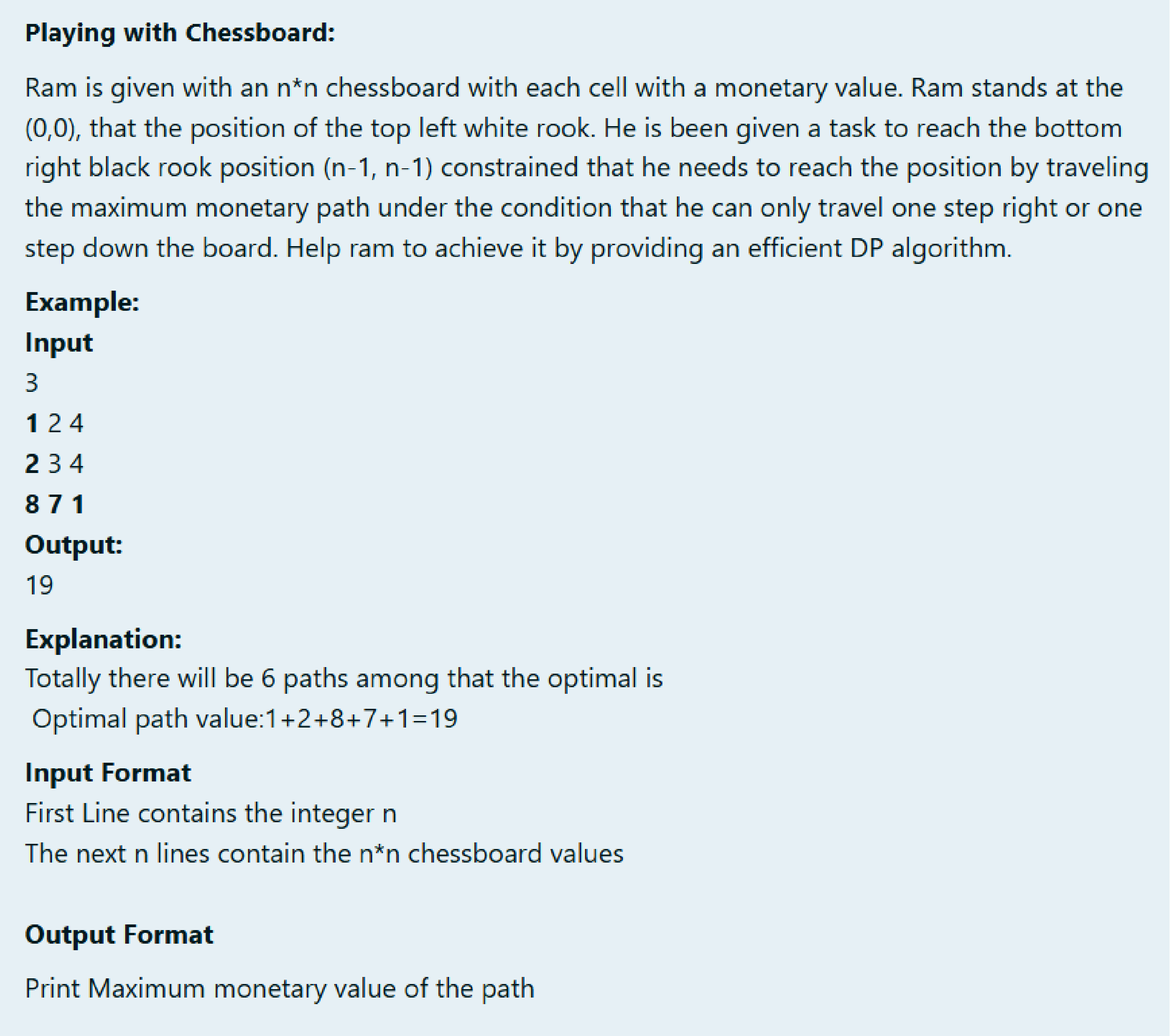
6



```c
#include <stdio.h>

unsigned long long countWays(int n) {
    unsigned long long dp[n + 1];
    dp[0] = 1;

    for (int i = 1; i <= n; i++) {
        dp[i] = 0;
        if (i >= 1) dp[i] += dp[i - 1];
        if (i >= 3) dp[i] += dp[i - 3];
    }

    return dp[n];
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%llu\n", countWays(n));
    return 0;
}

```



	Input	Expected	Got	
✓	6	6	6	✓
✓	25	8641	8641	✓
✓	100	24382819596721629	24382819596721629	✓



Playing with Chessboard:

Ram is given with an n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by traveling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:

Input

3

1 2 4

2 3 4

8 7 1

Output:

19

Explanation:

Totally there will be 6 paths among that the optimal is

Optimal path value:1+2+8+7+1=19

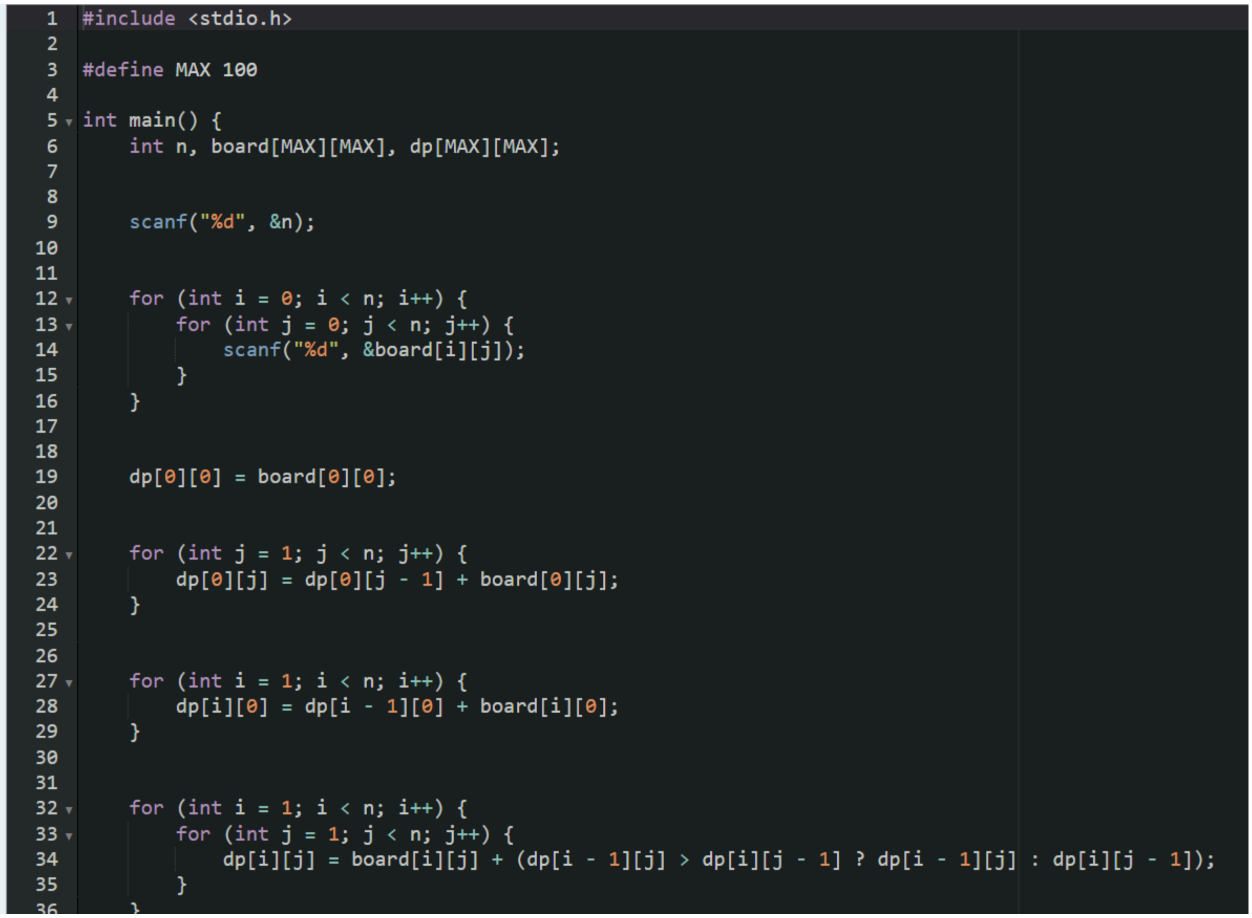
Input Format

First Line contains the integer n

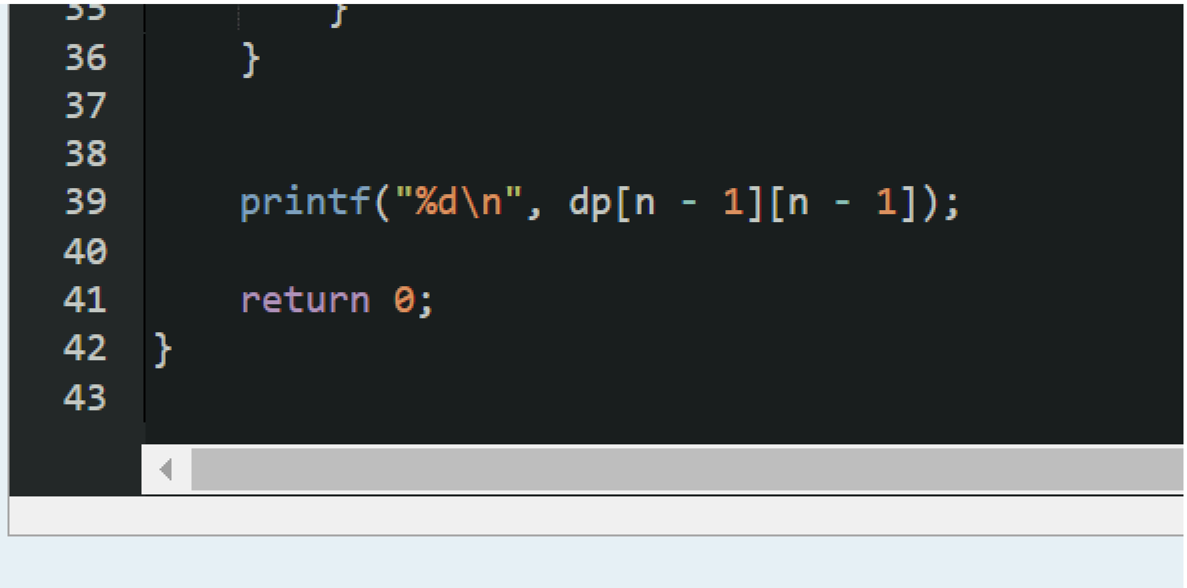
The next n lines contain the n*n chessboard values

Output Format

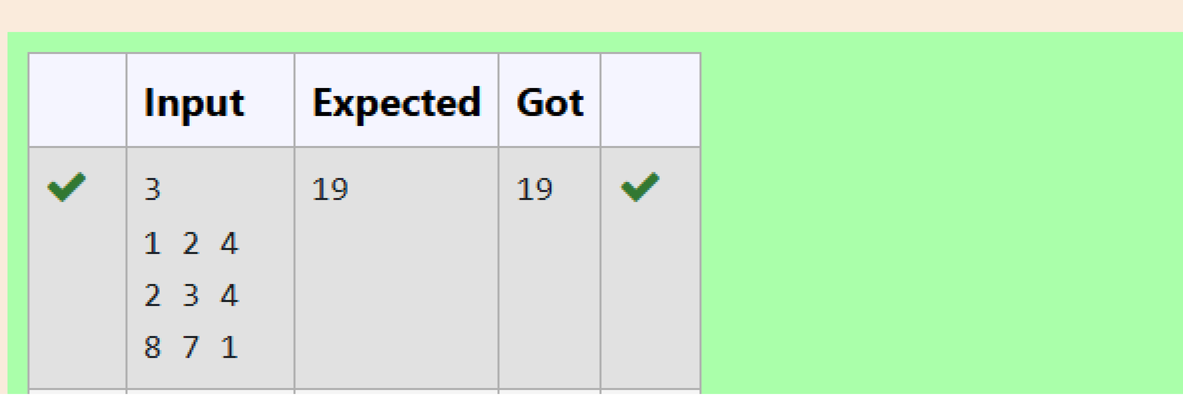
Print Maximum monetary value of the path



```c
#include <stdio.h>

#define MAX 100

int main() {
    int n, board[MAX][MAX], dp[MAX][MAX];


    scanf("%d", &n);


    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &board[i][j]);
        }
    }


    dp[0][0] = board[0][0];


    for (int j = 1; j < n; j++) {
        dp[0][j] = dp[0][j - 1] + board[0][j];
    }


    for (int i = 1; i < n; i++) {
        dp[i][0] = dp[i - 1][0] + board[i][0];
    }


    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            dp[i][j] = board[i][j] + (dp[i - 1][j] > dp[i][j - 1] ? dp[i - 1][j] : dp[i][j - 1]);
        }
    }
```

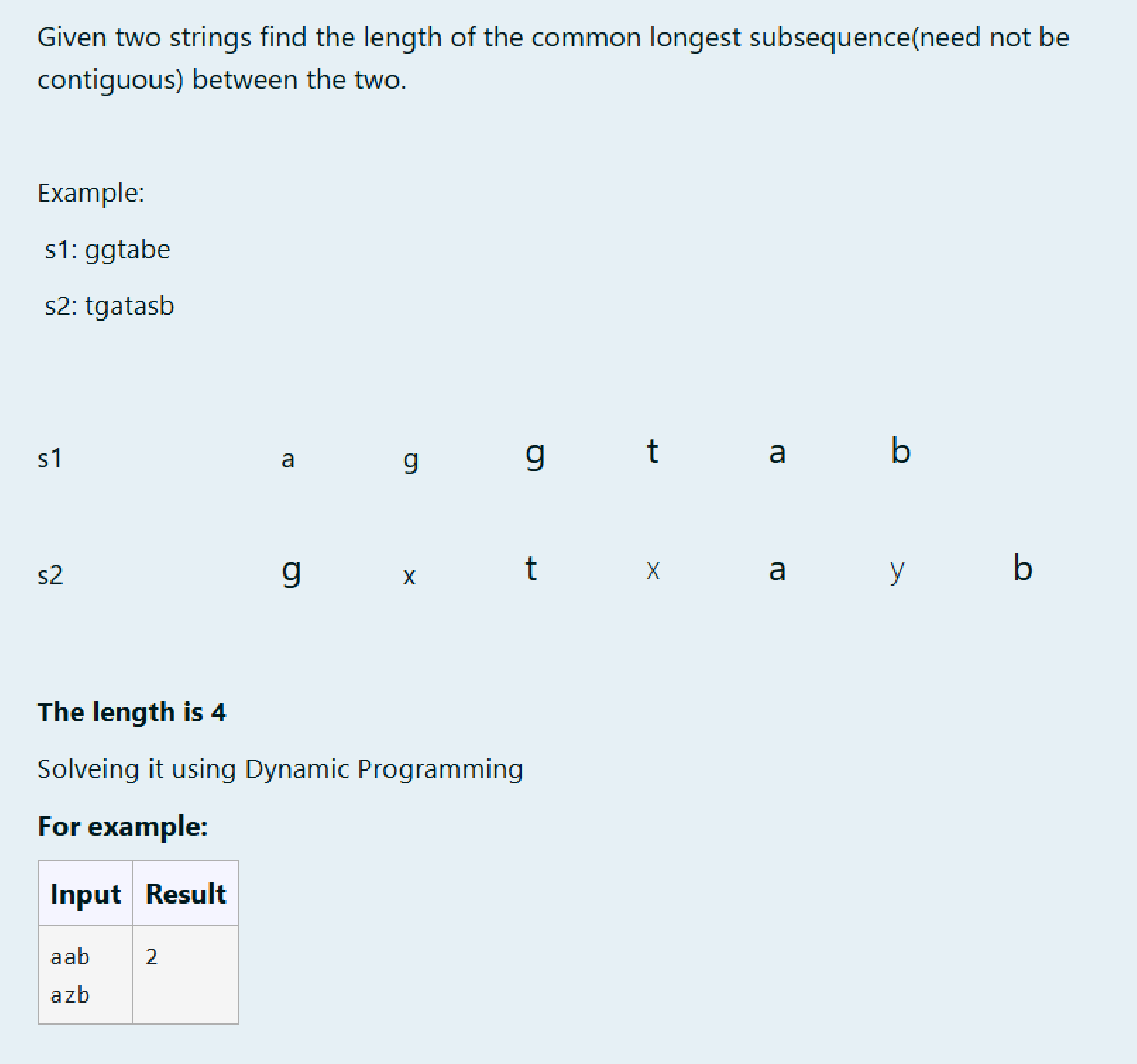


```
        }
    }


    printf("%d\n", dp[n - 1][n - 1]);

    return 0;
}
```



	Input	Expected	Got	
✓	3 1 2 4 2 3 4 8 7 1	19	19	✓

Longest common subsequence



Given two strings find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb

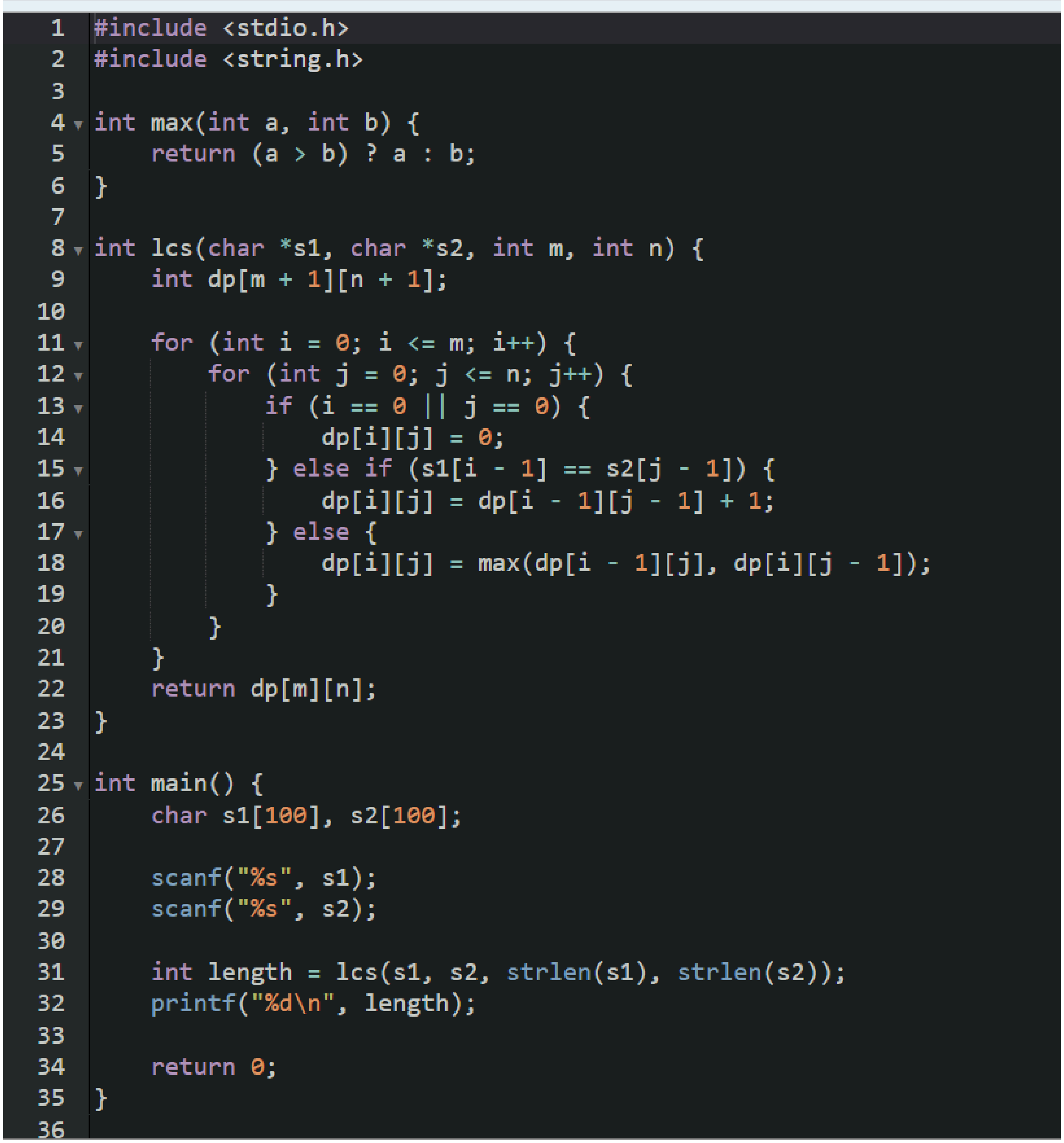
s1	a	g	g	t	a	b	
s2	g	x	t	x	a	y	b

The length is 4

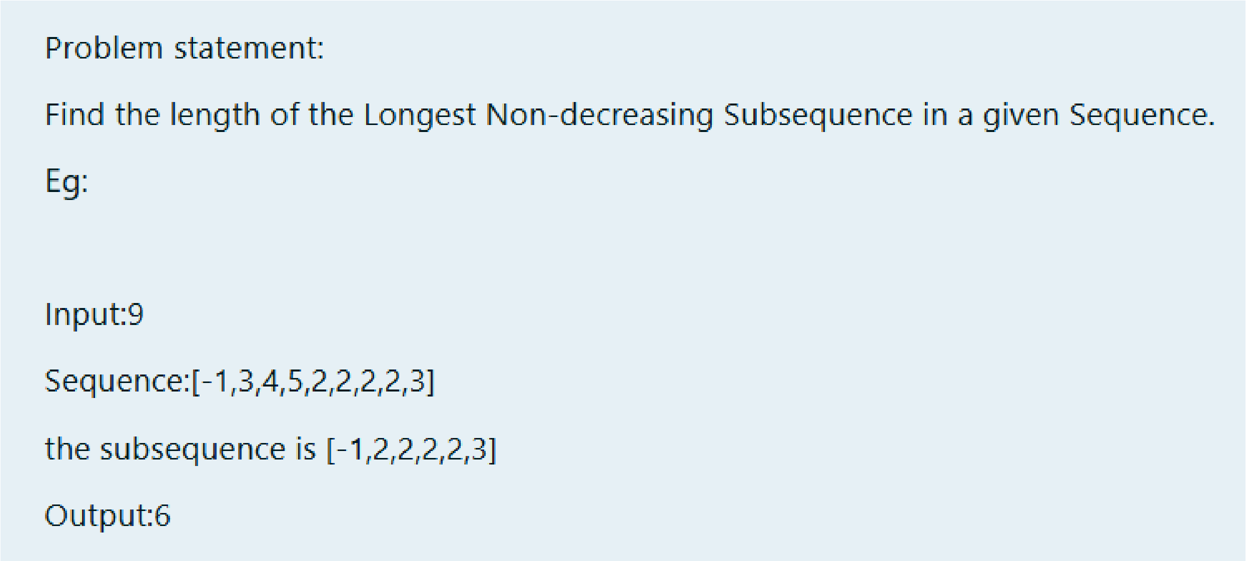
Solveing it using Dynamic Programming

For example:

Input	Result
aab azb	2



```
#include <stdio.h>
#include <string.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}

int lcs(char *s1, char *s2, int m, int n) {
    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; 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] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
    return dp[m][n];
}

int main() {
    char s1[100], s2[100];

    scanf("%s", s1);
    scanf("%s", s2);

    int length = lcs(s1, s2, strlen(s1), strlen(s2));
    printf("%d\n", length);

    return 0;
}
```



Problem statement:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

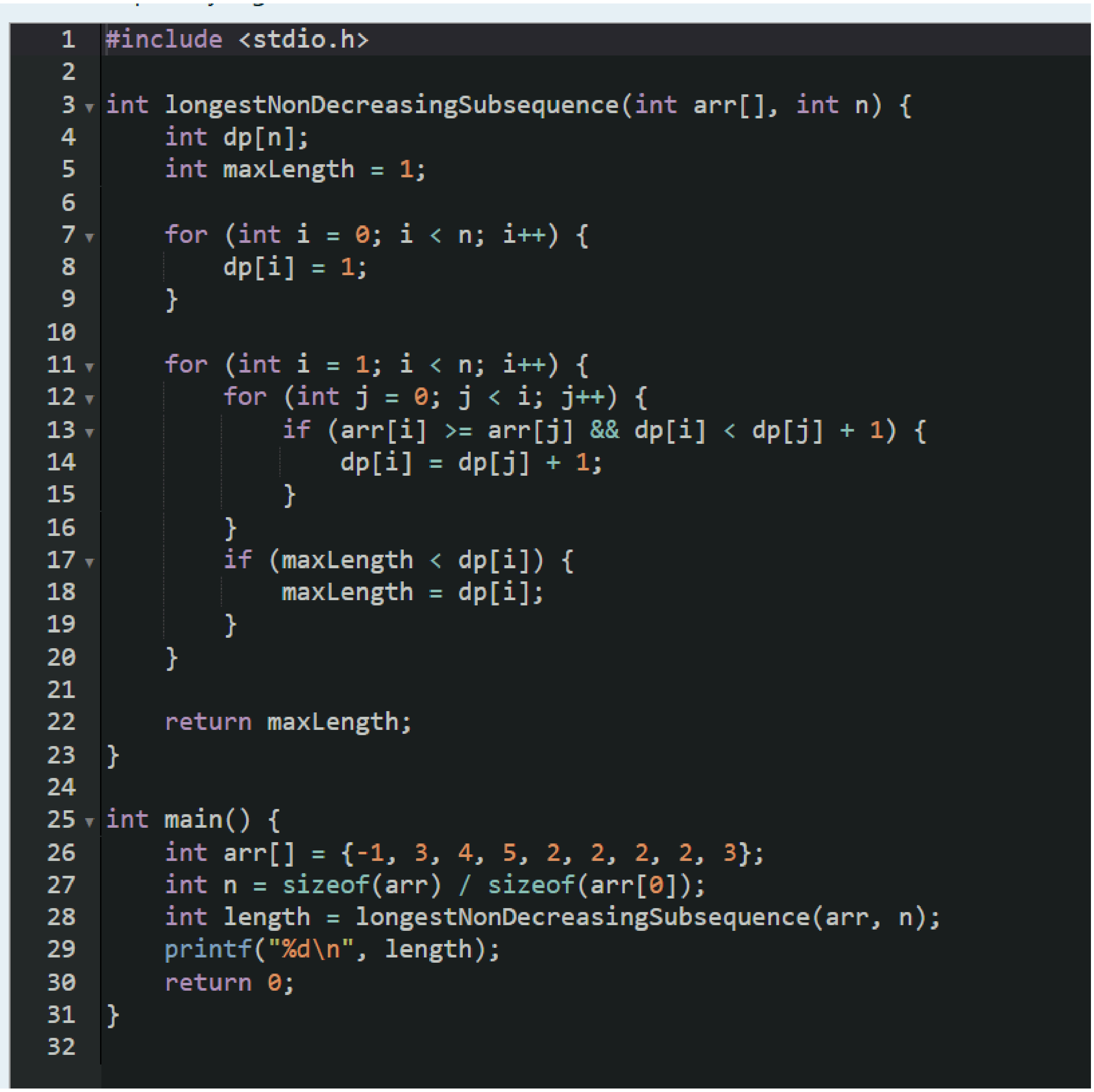
Eg:

Input:9

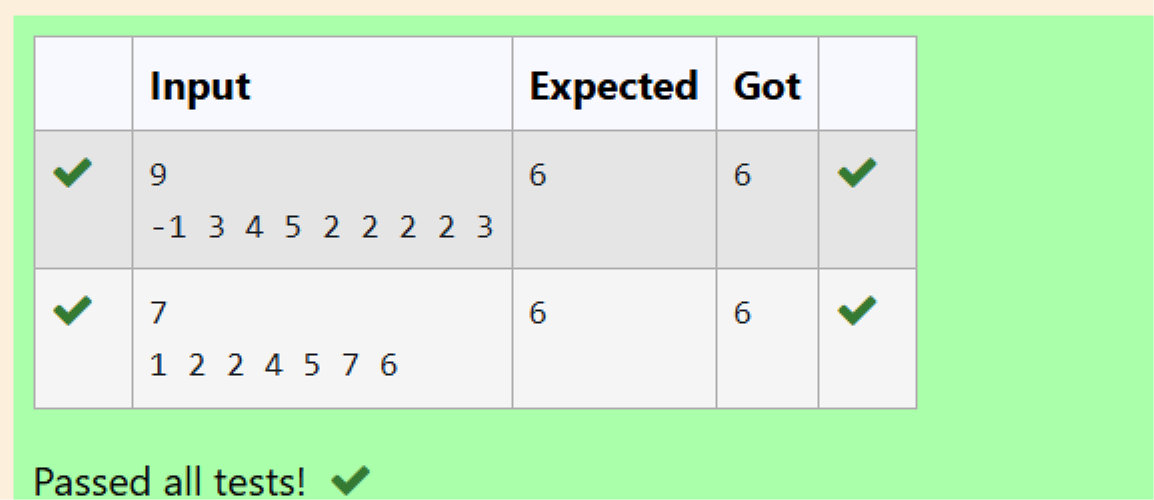
Sequence:[-1,3,4,5,2,2,2,2,3]

the subsequence is [-1,2,2,2,2,3]

Output:6



```c
#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[i] >= arr[j] && dp[i] < dp[j] + 1) {
                dp[i] = dp[j] + 1;
            }
        }
        if (maxLength < dp[i]) {
            maxLength = dp[i];
        }
    }

    return maxLength;
}

int main() {
    int arr[] = {-1, 3, 4, 5, 2, 2, 2, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    int length = longestNonDecreasingSubsequence(arr, n);
    printf("%d\n", length);
    return 0;
}

```

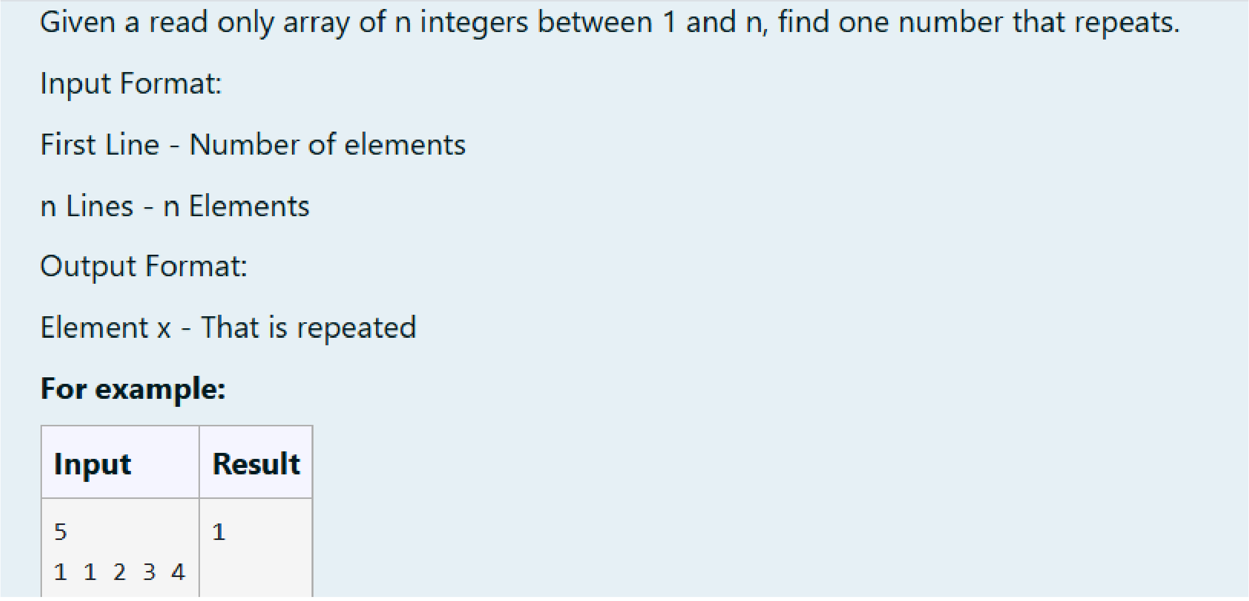


	Input	Expected	Got	
✓	9 -1 3 4 5 2 2 2 2 3	6	6	✓
✓	7 1 2 2 4 5 7 6	6	6	✓

Passed all tests! ✓

COMPETITIVE PROGRAMMING

Complexity - O(n^n)



Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

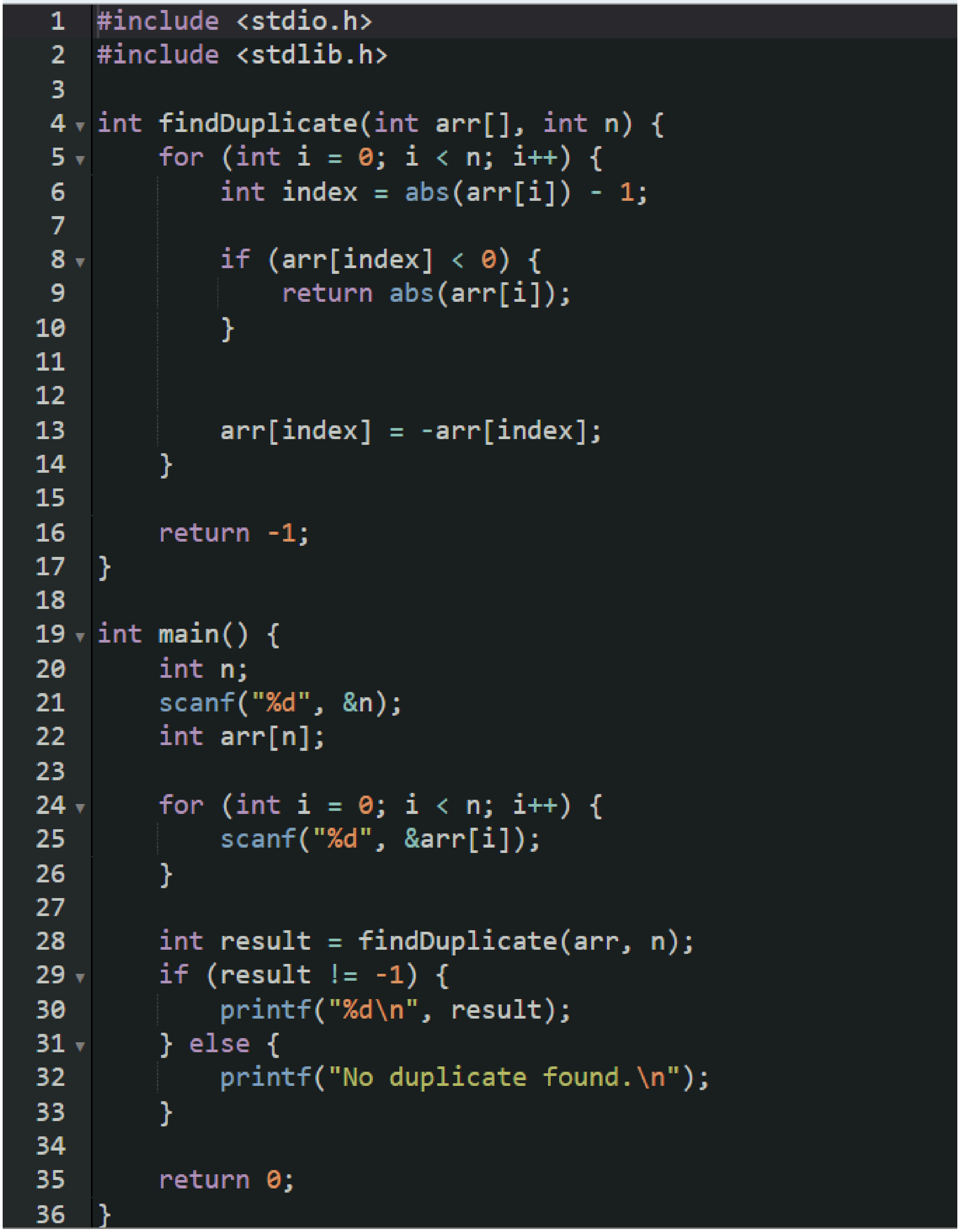
n Lines - n Elements

Output Format:

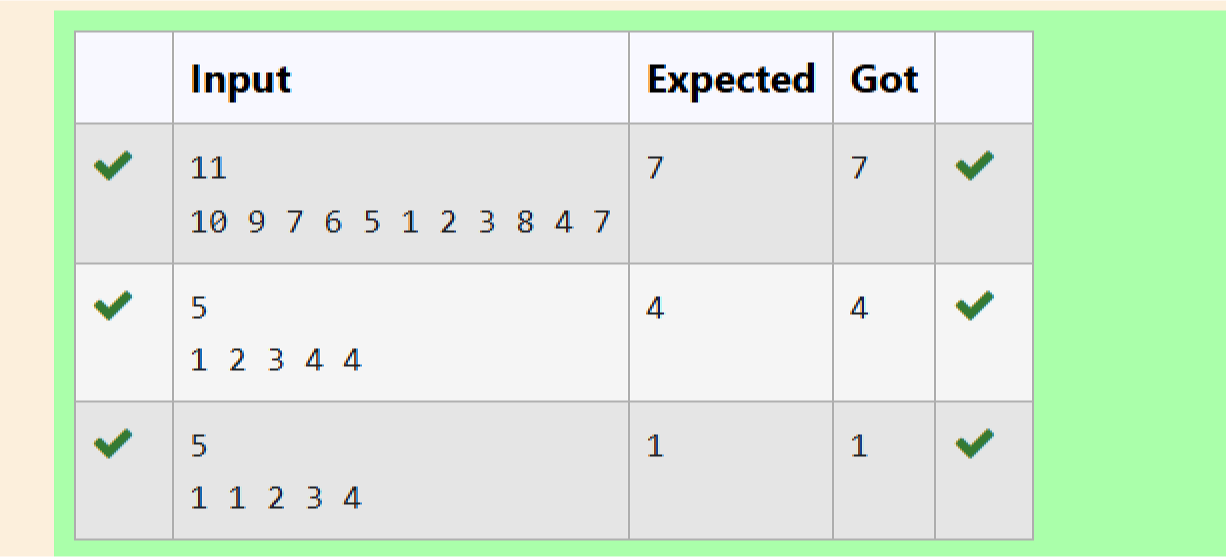
Element x - That is repeated

For example:

Input	Result
5 1 1 2 3 4	1

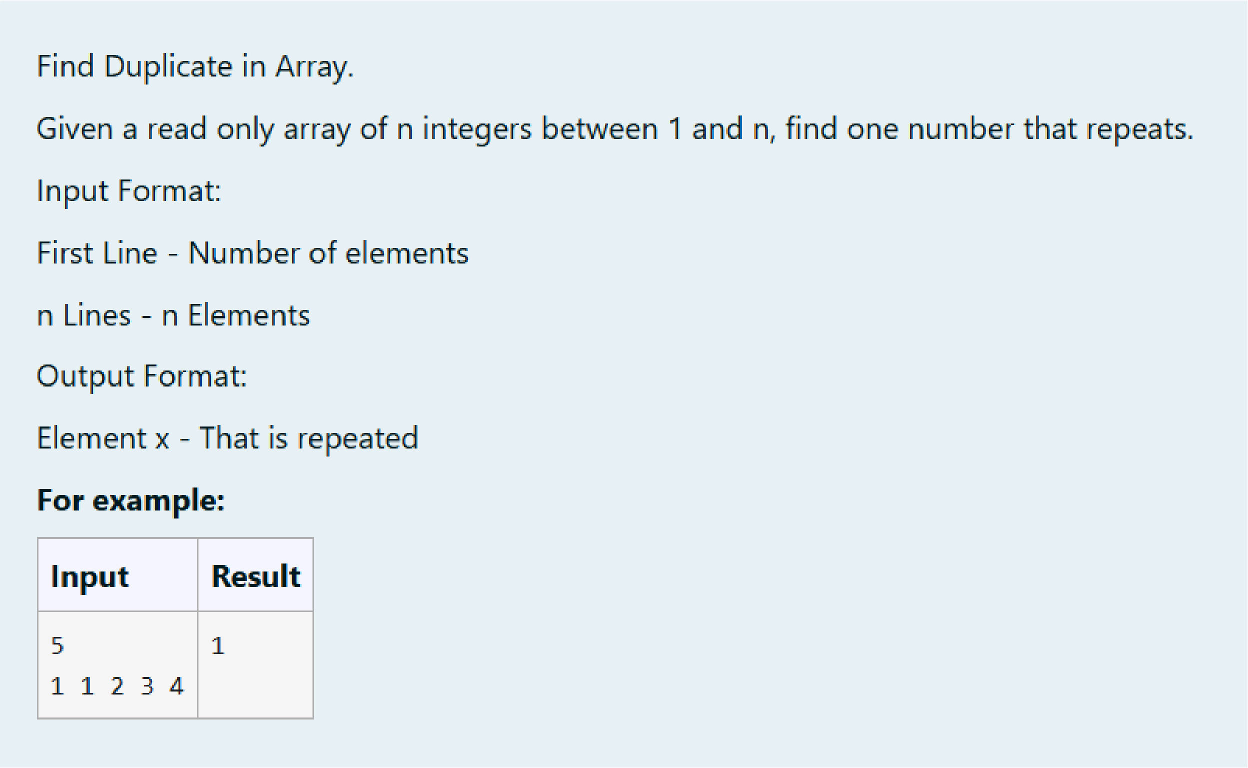


```c
#include <stdio.h>
#include <stdlib.h>

int findDuplicate(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int index = abs(arr[i]) - 1;

        if (arr[index] < 0) {
            return abs(arr[i]);
        }


        arr[index] = -arr[index];
    }

    return -1;
}

int main() {
    int n;
    scanf("%d", &n);
    int arr[n];

    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int result = findDuplicate(arr, n);
    if (result != -1) {
        printf("%d\n", result);
    } else {
        printf("No duplicate found.\n");
    }

    return 0;
}
```



	Input	Expected	Got	
✓	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✓
✓	5 1 2 3 4 4	4	4	✓
✓	5 1 1 2 3 4	1	1	✓

O(n) complexity



Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

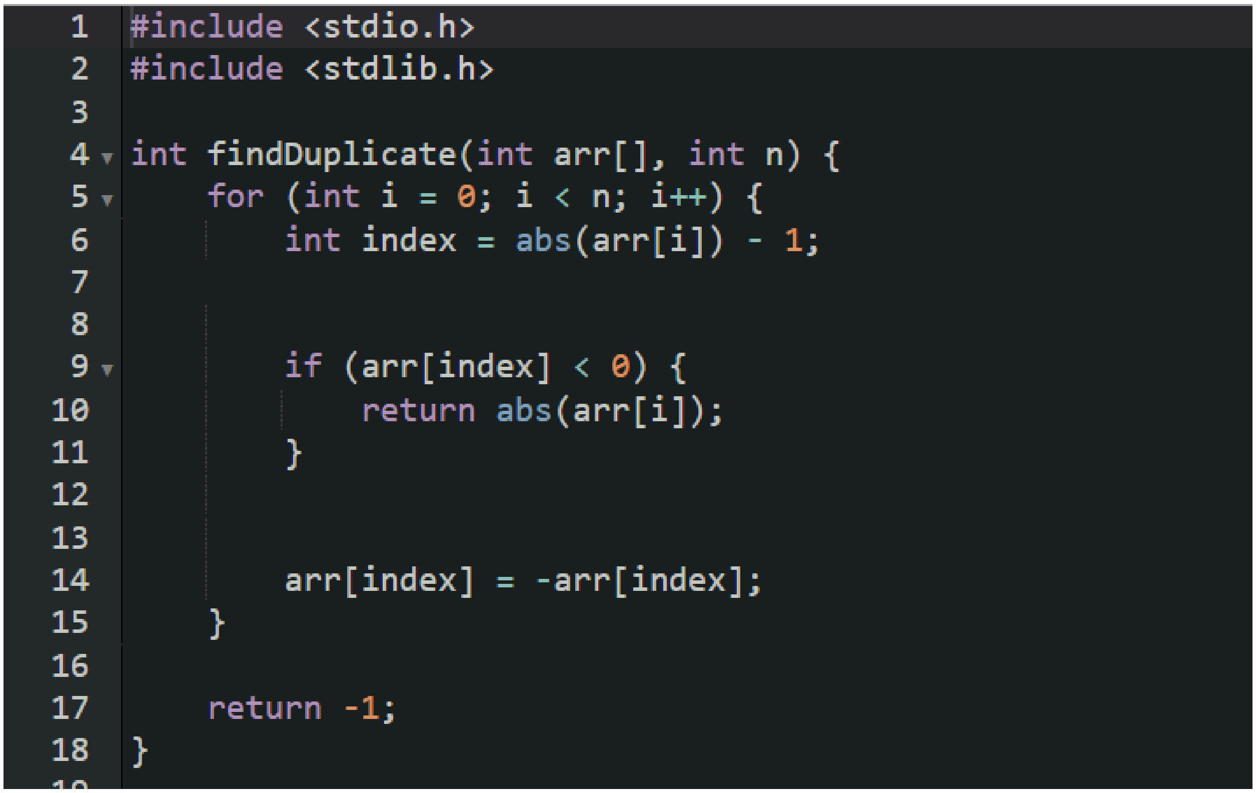
n Lines - n Elements

Output Format:

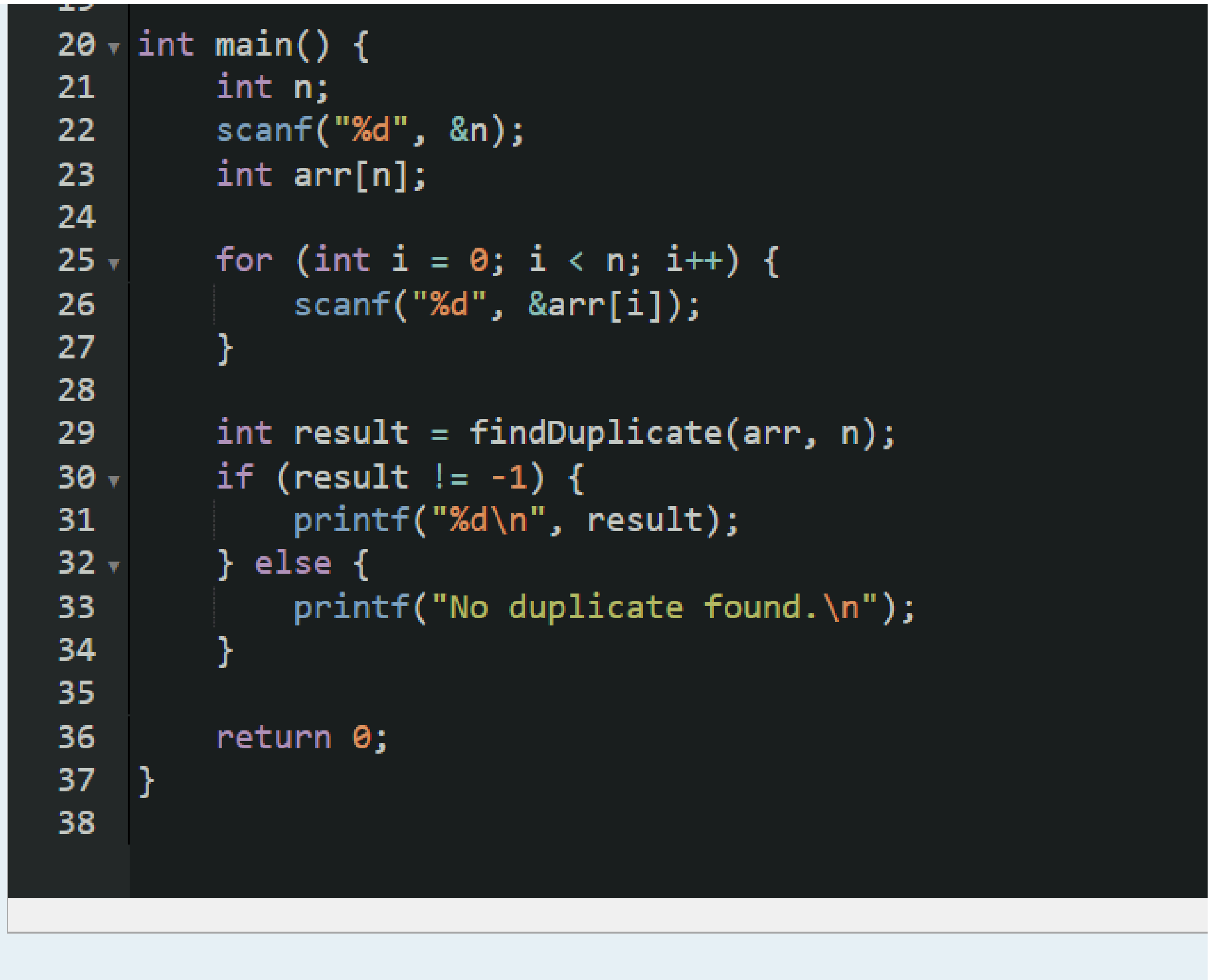
Element x - That is repeated

For example:

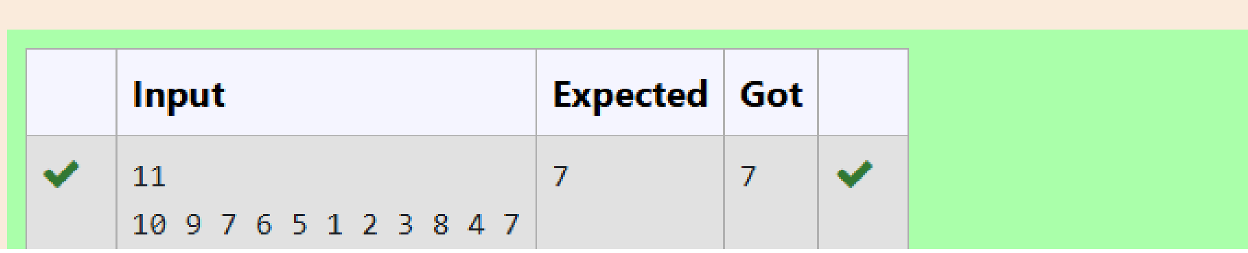
Input	Result
5 1 1 2 3 4	1



```c
#include <stdio.h>
#include <stdlib.h>

int findDuplicate(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int index = abs(arr[i]) - 1;


        if (arr[index] < 0) {
            return abs(arr[i]);
        }


        arr[index] = -arr[index];
    }

    return -1;
}
```

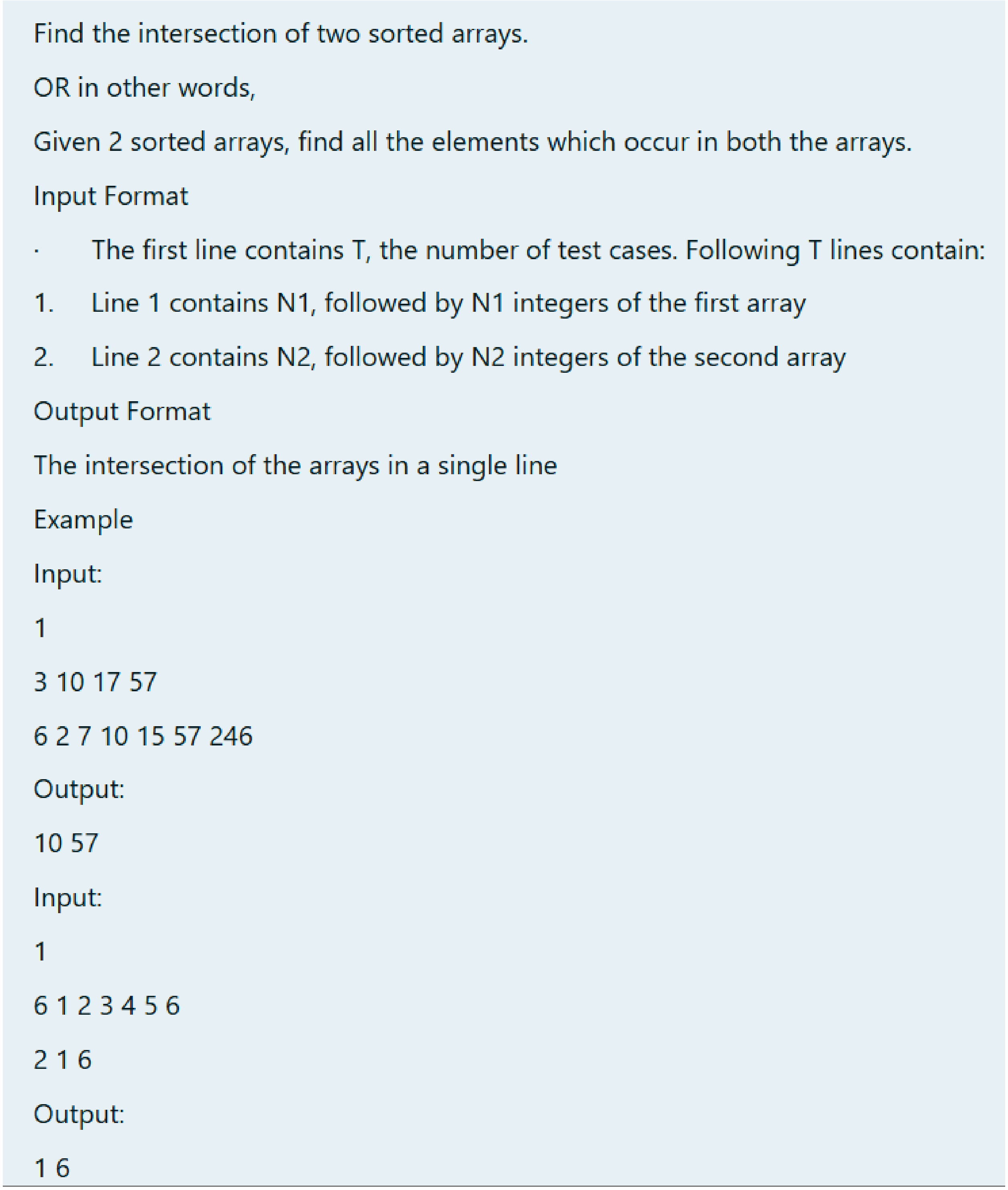


```c
int main() {
    int n;
    scanf("%d", &n);
    int arr[n];

    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int result = findDuplicate(arr, n);
    if (result != -1) {
        printf("%d\n", result);
    } else {
        printf("No duplicate found.\n");
    }

    return 0;
}

```



	Input	Expected	Got	
✓	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✓

O(m*n) complexity



Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

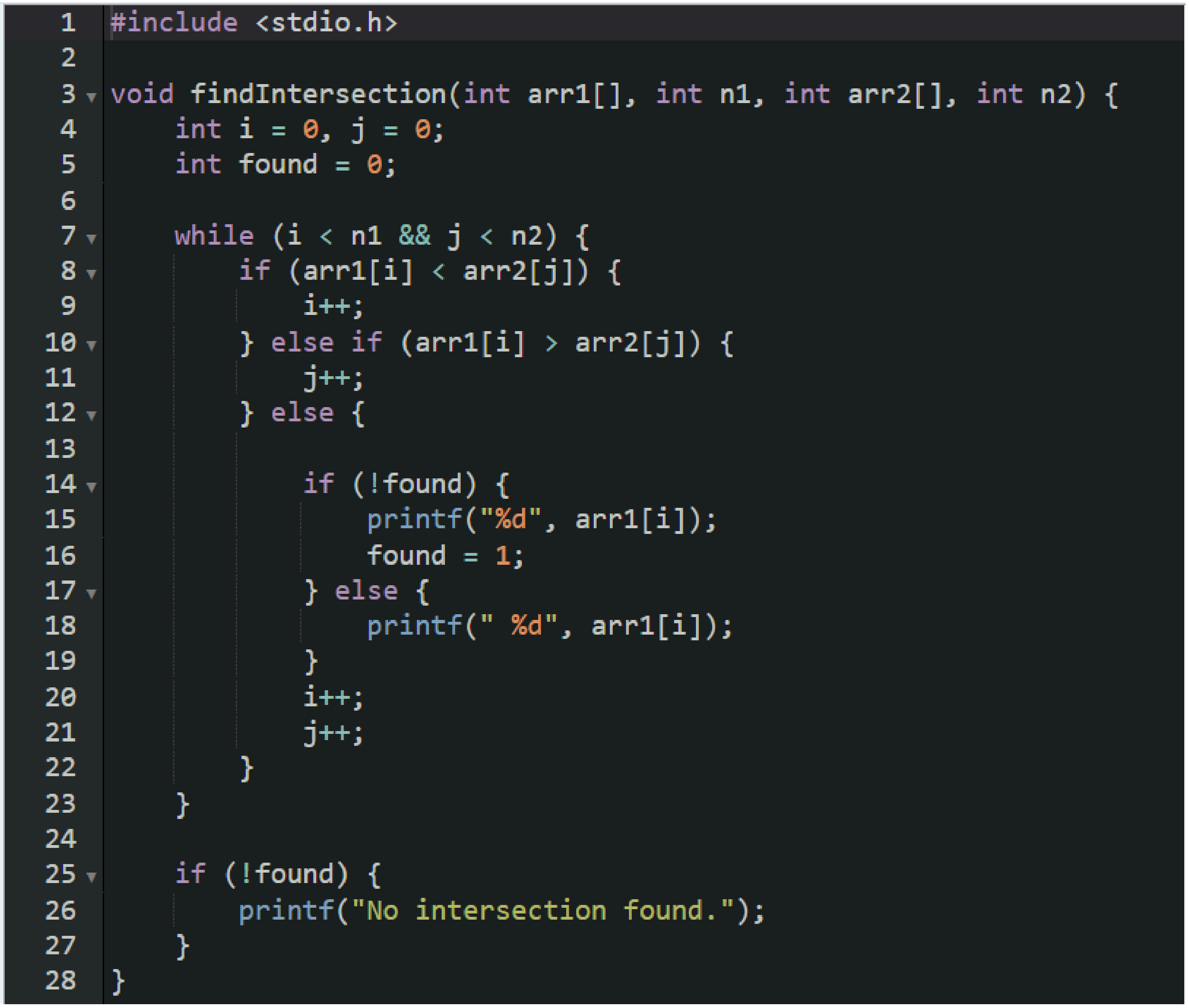
1

6 1 2 3 4 5 6

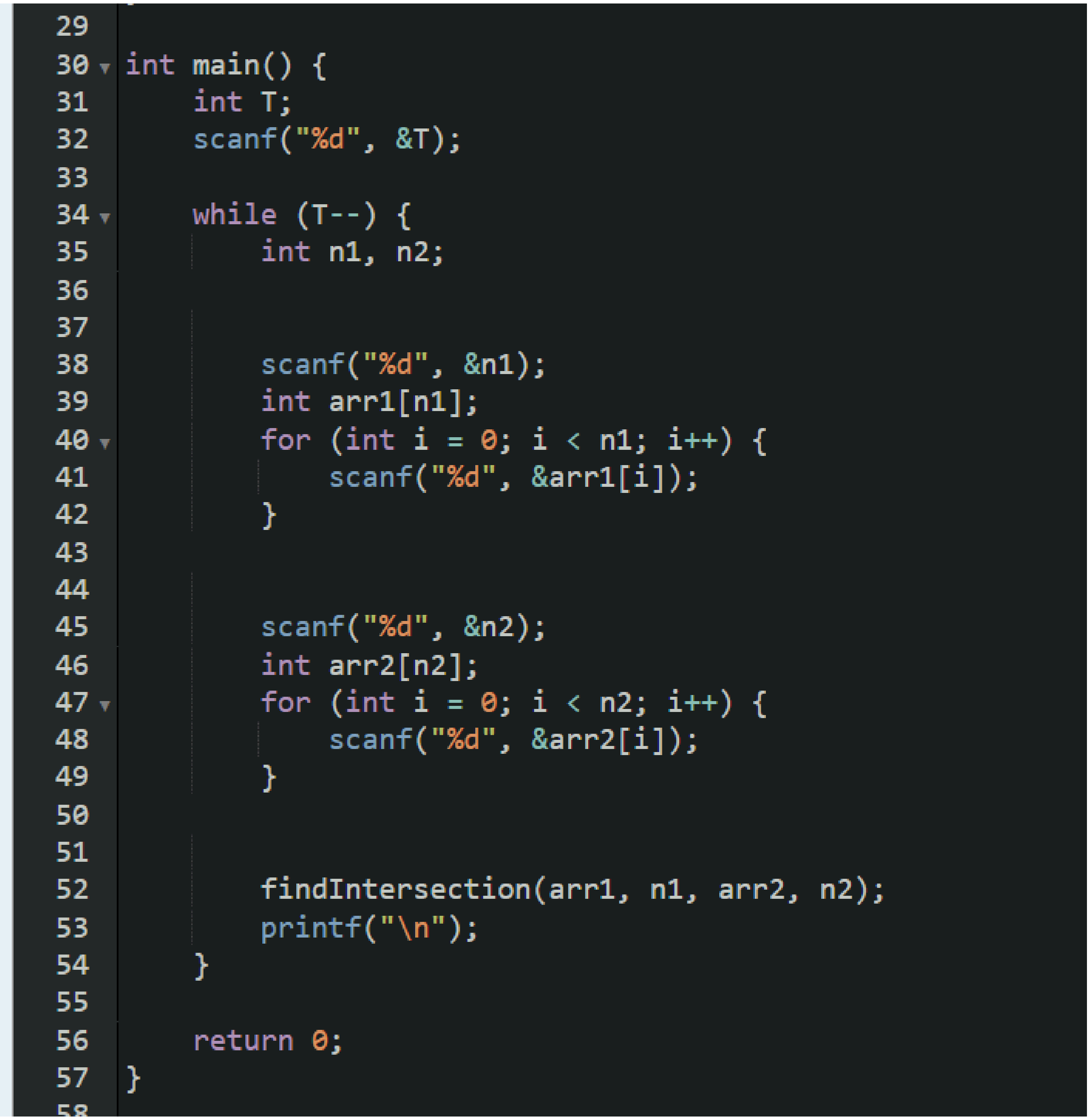
2 1 6

Output:

1 6

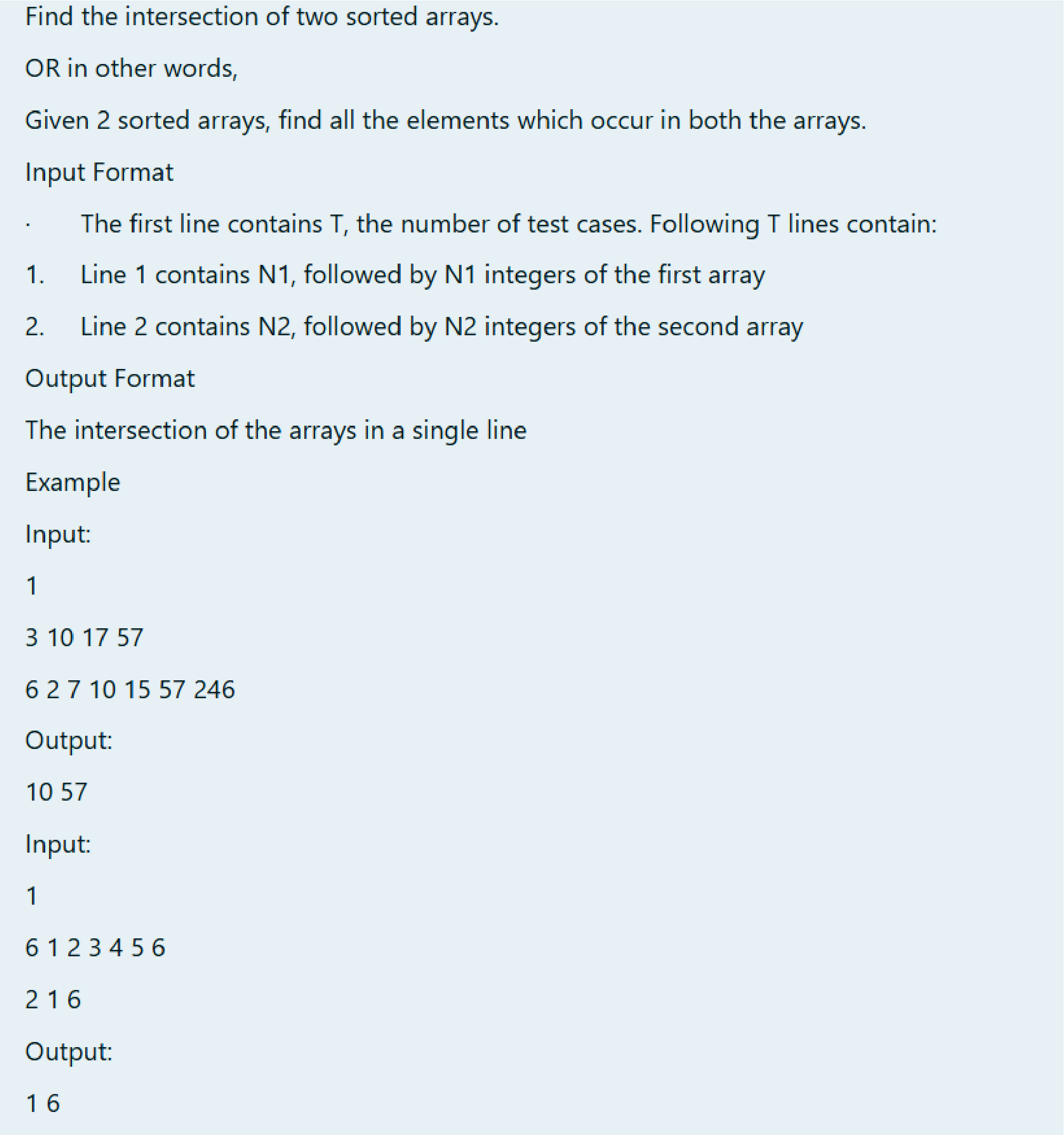


```c
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int found = 0;

    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr1[i] > arr2[j]) {
            j++;
        } else {

            if (!found) {
                printf("%d", arr1[i]);
                found = 1;
            } else {
                printf(" %d", arr1[i]);
            }
            i++;
            j++;
        }
    }

    if (!found) {
        printf("No intersection found.");
    }
}
```



```c

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);
        printf("\n");
    }

    return 0;
}
```

O(m+n) complexity



Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

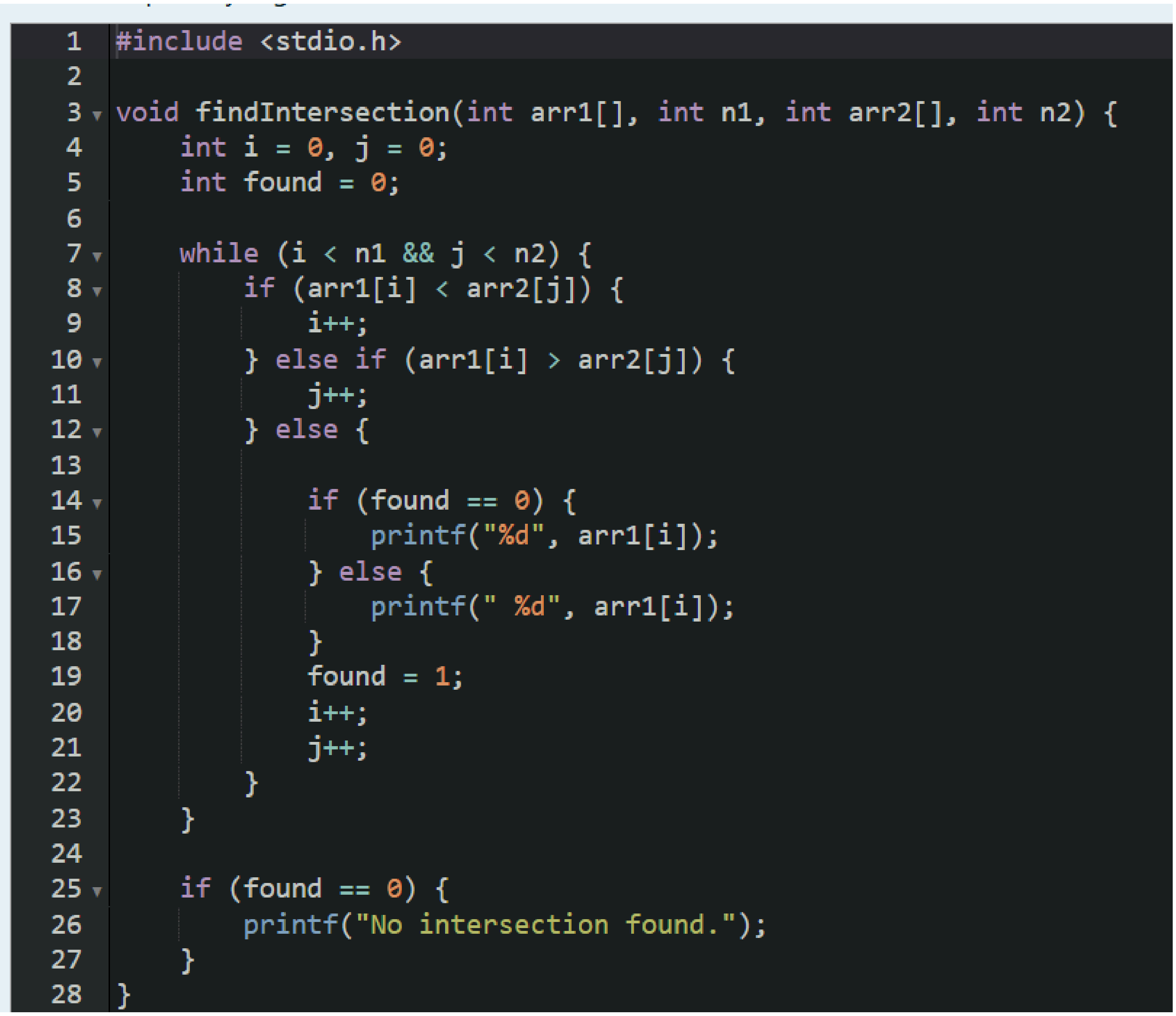
1

6 1 2 3 4 5 6

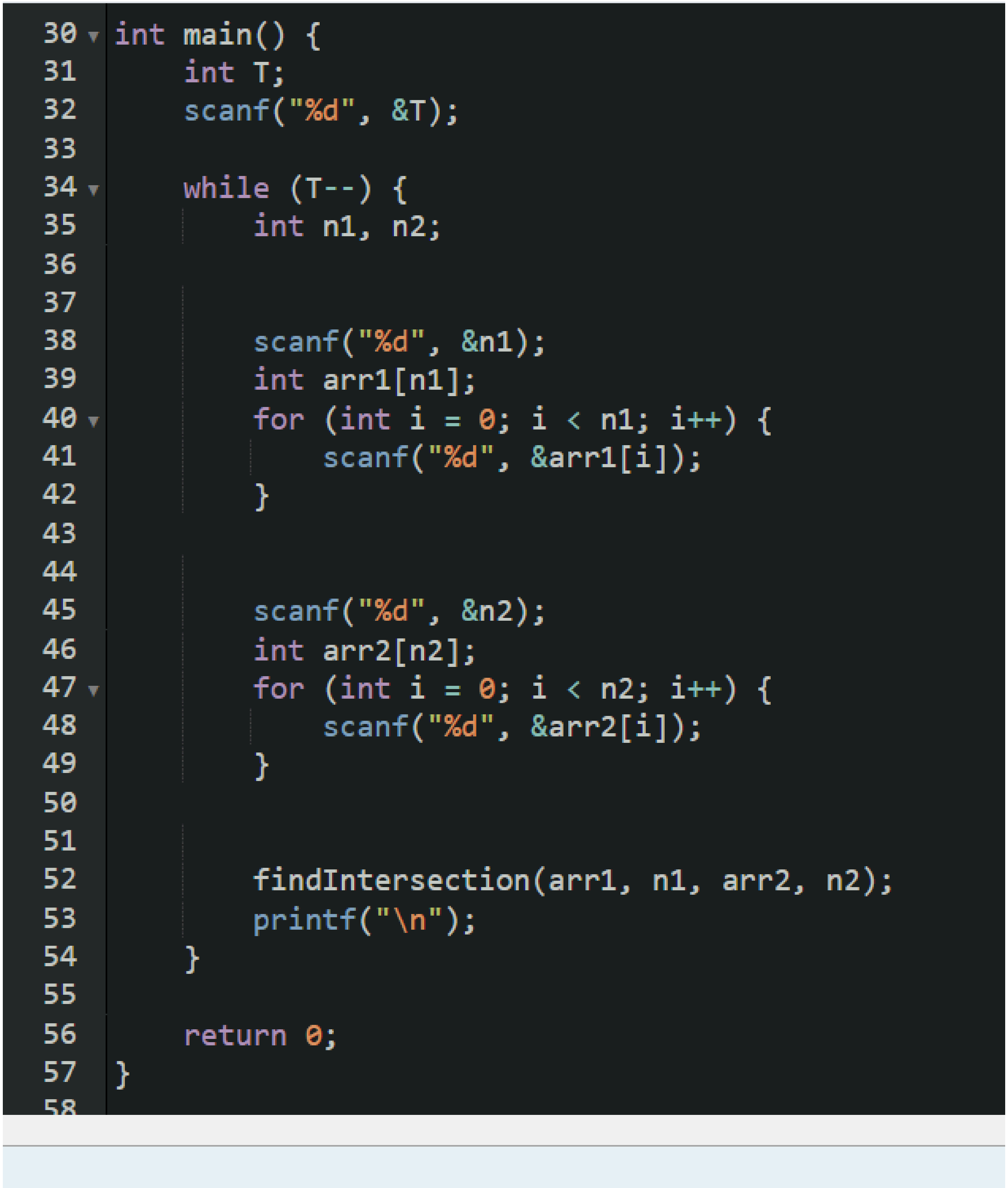
2 1 6

Output:

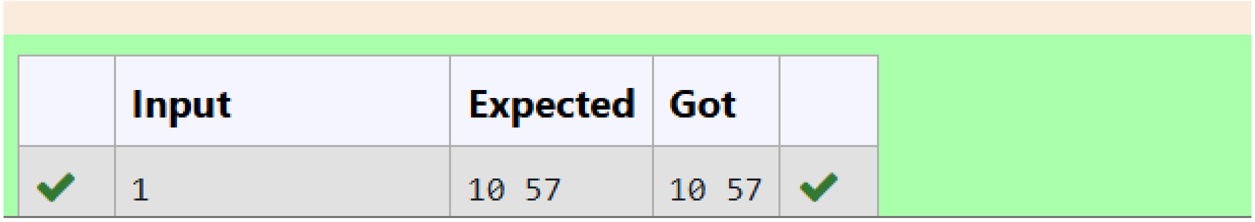
1 6



```c
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int found = 0;

    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            i++;
        } else if (arr1[i] > arr2[j]) {
            j++;
        } else {

            if (found == 0) {
                printf("%d", arr1[i]);
            } else {
                printf(" %d", arr1[i]);
            }
            found = 1;
            i++;
            j++;
        }
    }

    if (found == 0) {
        printf("No intersection found.");
    }
}
```

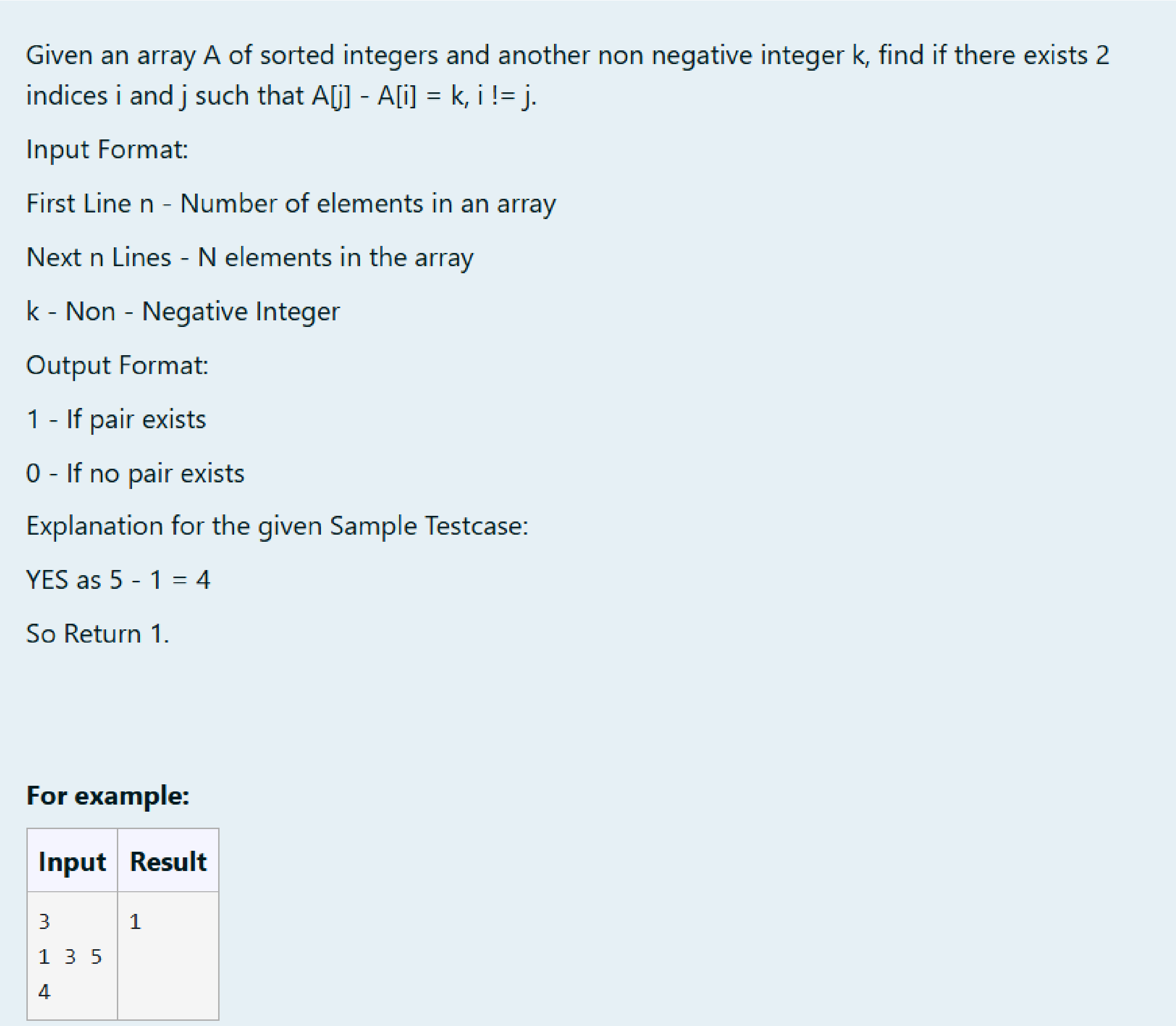


```c
int main() {
    int T;
    scanf("%d", &T);

    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);
        printf("\n");
    }

    return 0;
}
```



	Input	Expected	Got	
✓	1	10 57	10 57	✓

O(n^2) complexity



Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

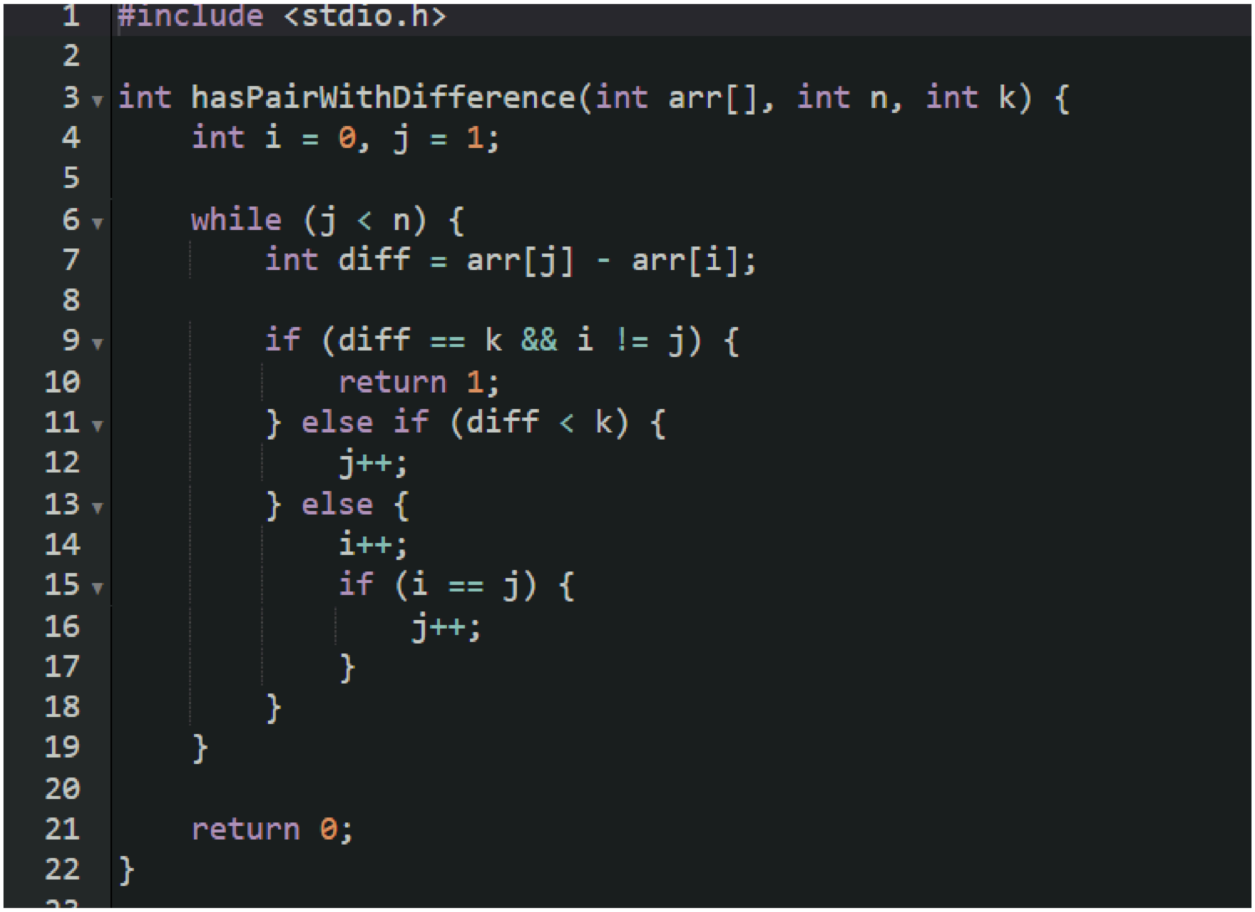
YES as 5 - 1 = 4

So Return 1.

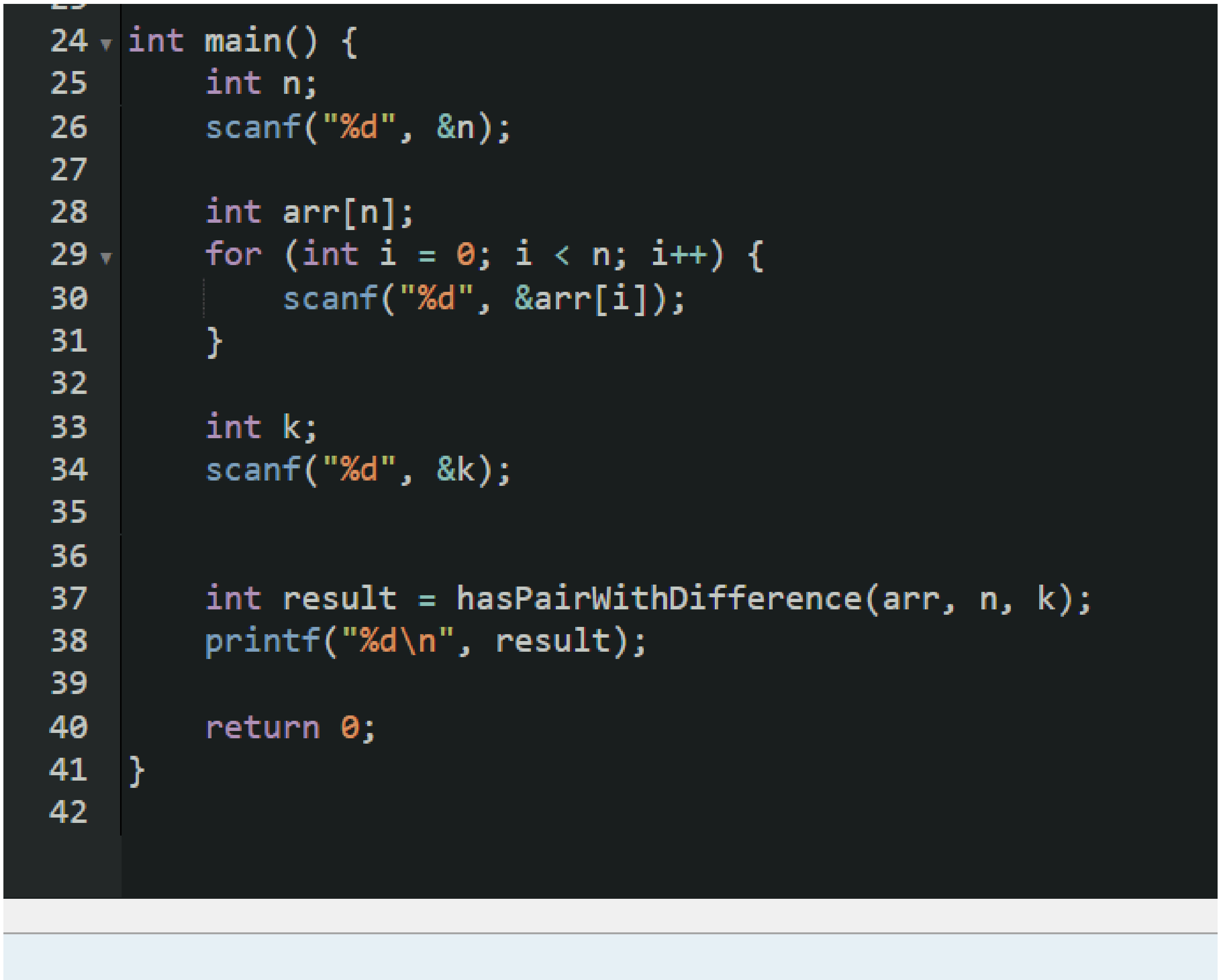
For example:

Input	Result

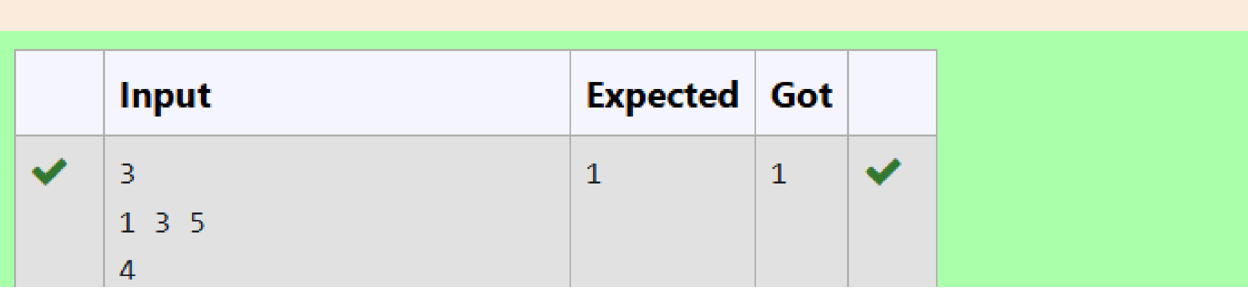
3
1 3 5
4
	1



```c
#include <stdio.h>

int hasPairWithDifference(int arr[], int n, int k) {
    int i = 0, j = 1;

    while (j < n) {
        int diff = arr[j] - arr[i];

        if (diff == k && i != j) {
            return 1;
        } else if (diff < k) {
            j++;
        } else {
            i++;
            if (i == j) {
                j++;
            }
        }
    }

    return 0;
}
```

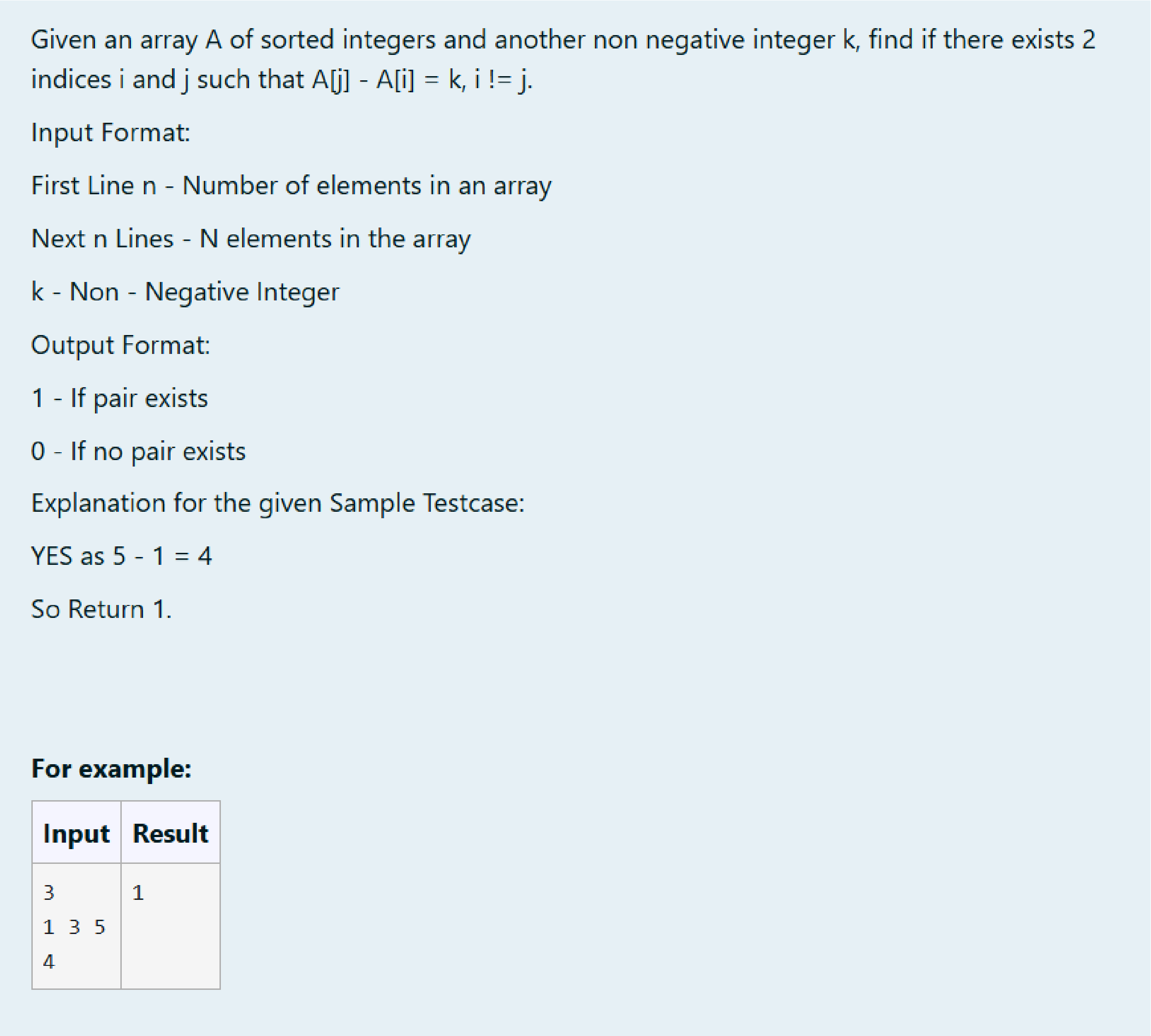


```c
int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int k;
    scanf("%d", &k);


    int result = hasPairWithDifference(arr, n, k);
    printf("%d\n", result);

    return 0;
}

```



	Input	Expected	Got	
✓	3 1 3 5 4	1	1	✓

O(n) complexity



Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

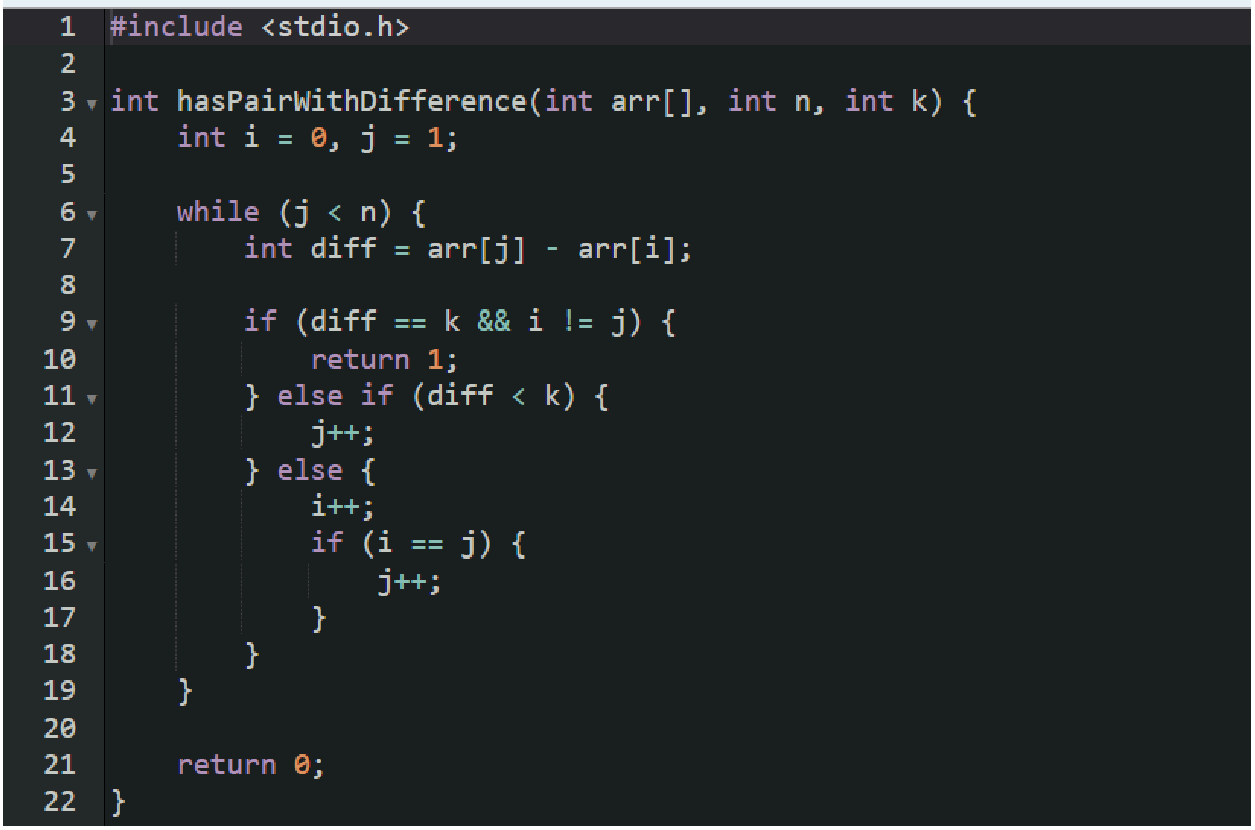
YES as 5 - 1 = 4

So Return 1.

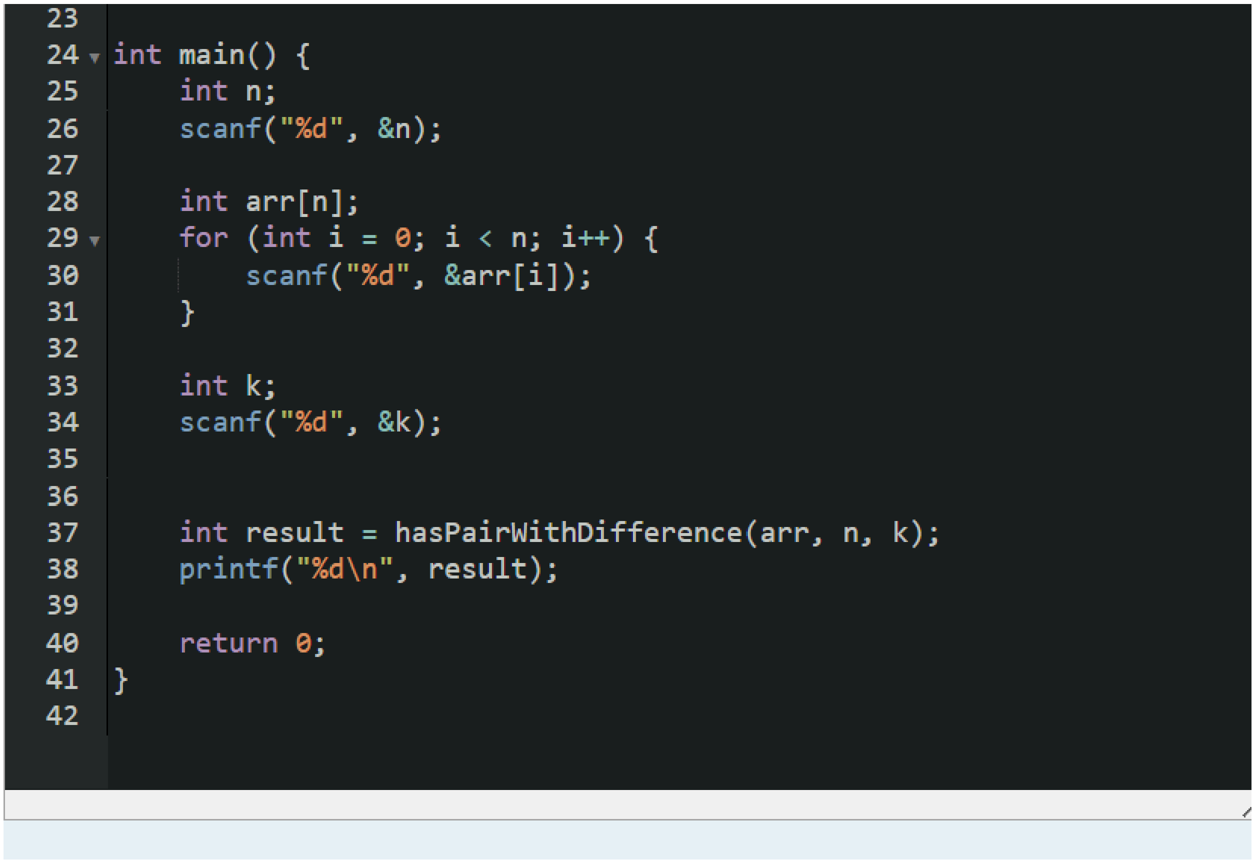
For example:

Input	Result

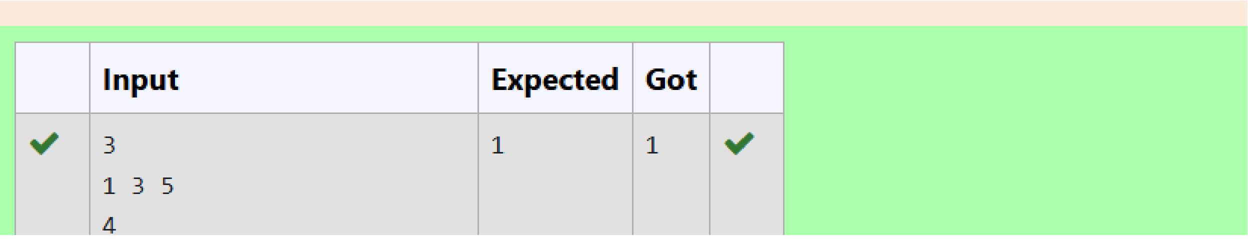
3
1 3 5
4
	1



```c
#include <stdio.h>

int hasPairWithDifference(int arr[], int n, int k) {
    int i = 0, j = 1;

    while (j < n) {
        int diff = arr[j] - arr[i];

        if (diff == k && i != j) {
            return 1;
        } else if (diff < k) {
            j++;
        } else {
            i++;
            if (i == j) {
                j++;
            }
        }
    }

    return 0;
}
```



```c

int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int k;
    scanf("%d", &k);


    int result = hasPairWithDifference(arr, n, k);
    printf("%d\n", result);

    return 0;
}

```



	Input	Expected	Got	
✓	3 1 3 5 4	1	1	✓