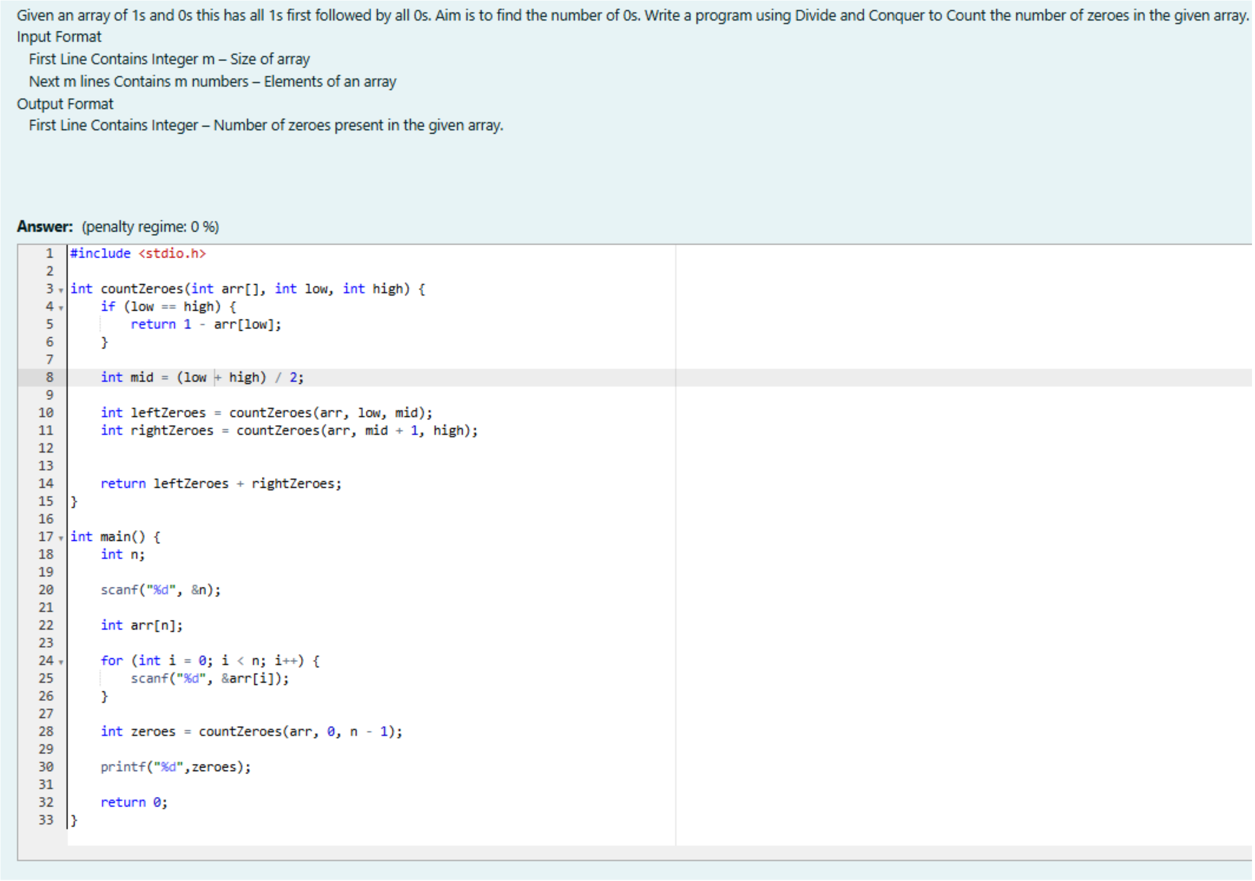
DIVIDE AND CONQUER



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

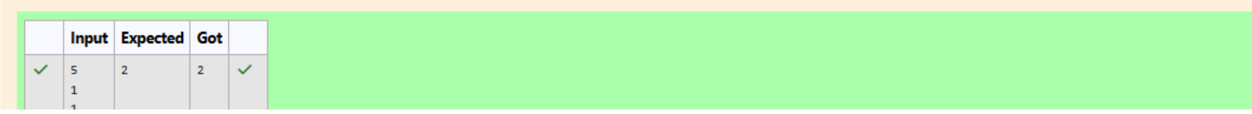
Next m lines Contains m numbers – Elements of an array

Output Format

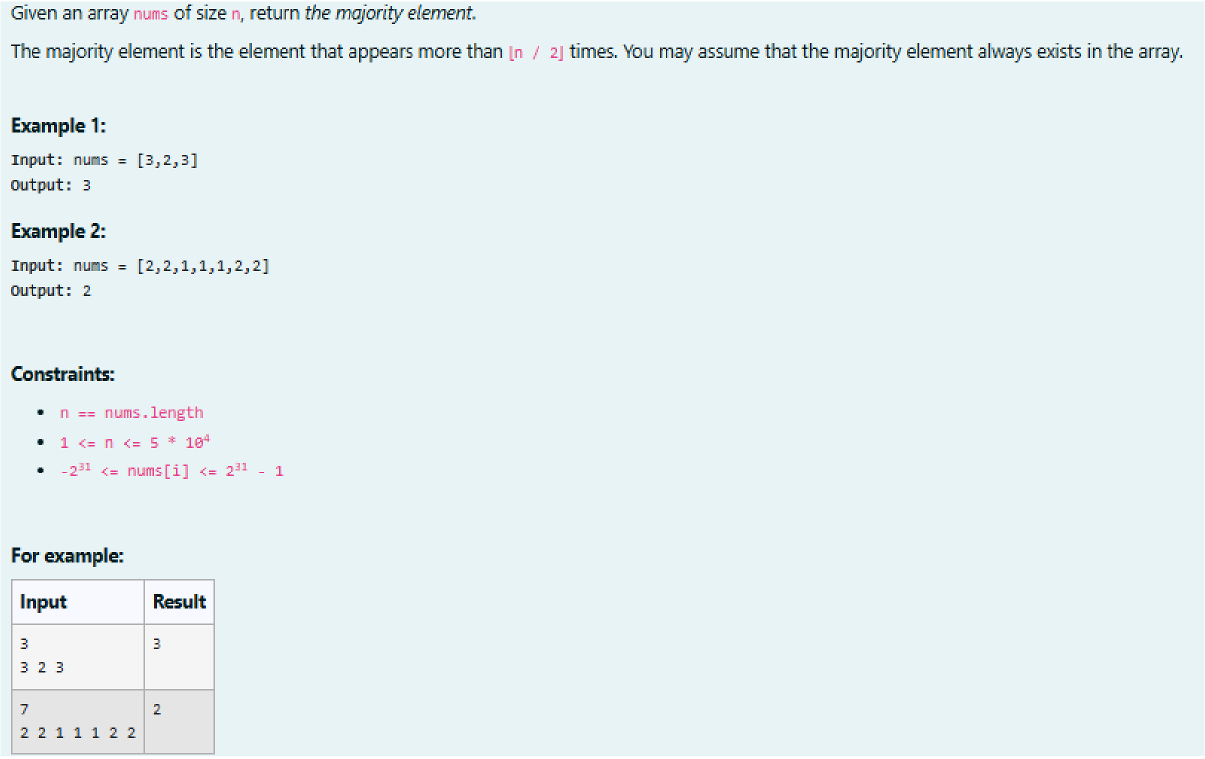
First Line Contains Integer – Number of zeroes present in the given array.

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>

int countZeroes(int arr[], int low, int high) {
    if (low == high) {
        return 1 - arr[low];
    }

    int mid = (low + high) / 2;

    int leftZeroes = countZeroes(arr, low, mid);
    int rightZeroes = countZeroes(arr, mid + 1, high);


    return leftZeroes + rightZeroes;
}

int main() {
    int n;

    scanf("%d", &n);

    int arr[n];

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

    int zeroes = countZeroes(arr, 0, n - 1);

    printf("%d",zeroes);

    return 0;
}
```



	Input	Expected	Got	
✓	5 1 1	2	2	✓



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:

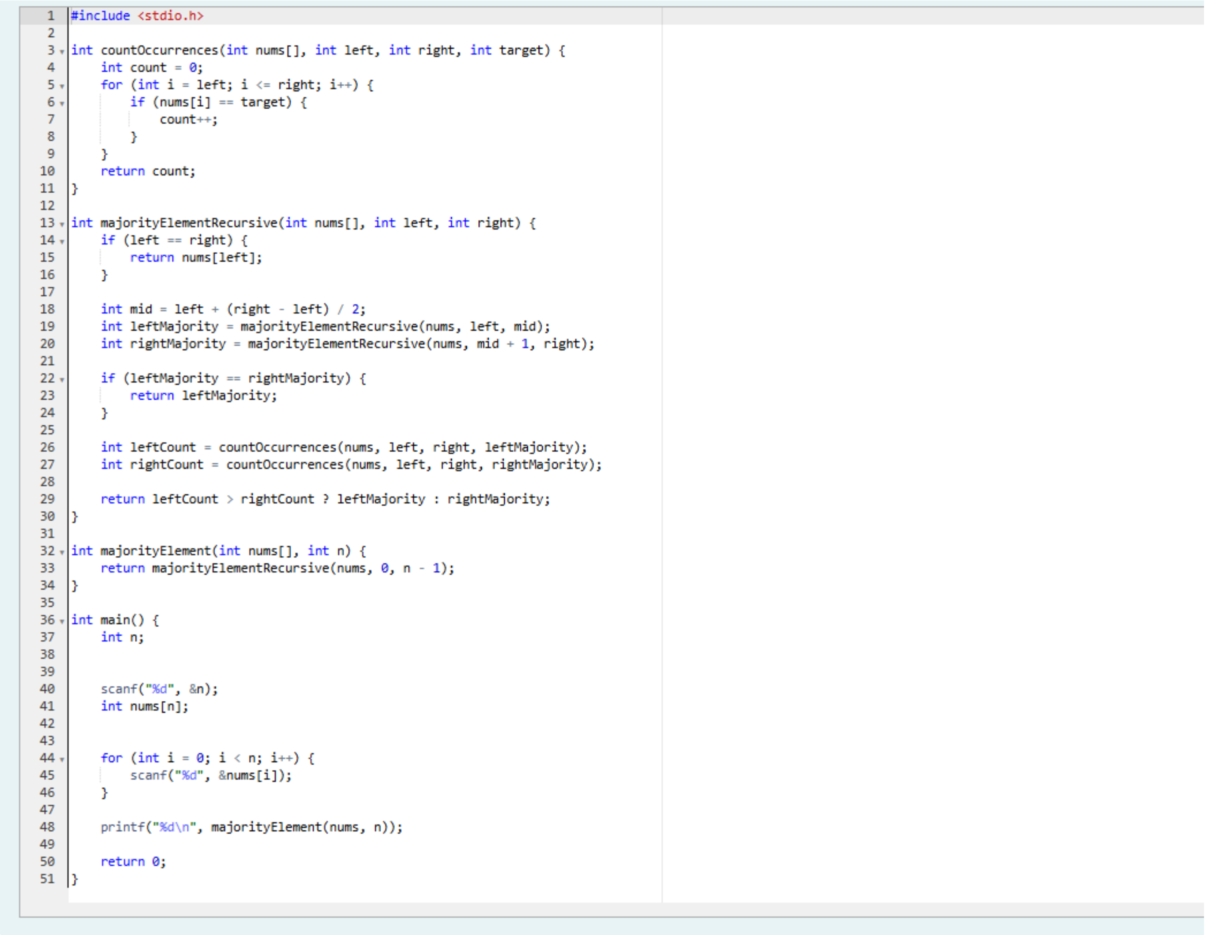
```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

Constraints:

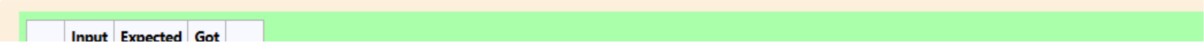
- `n == nums.length`
- $1 <= n <= 5 * 10^4$
- -2^{31} <= nums[i] <= $2^{31} - 1$

For example:

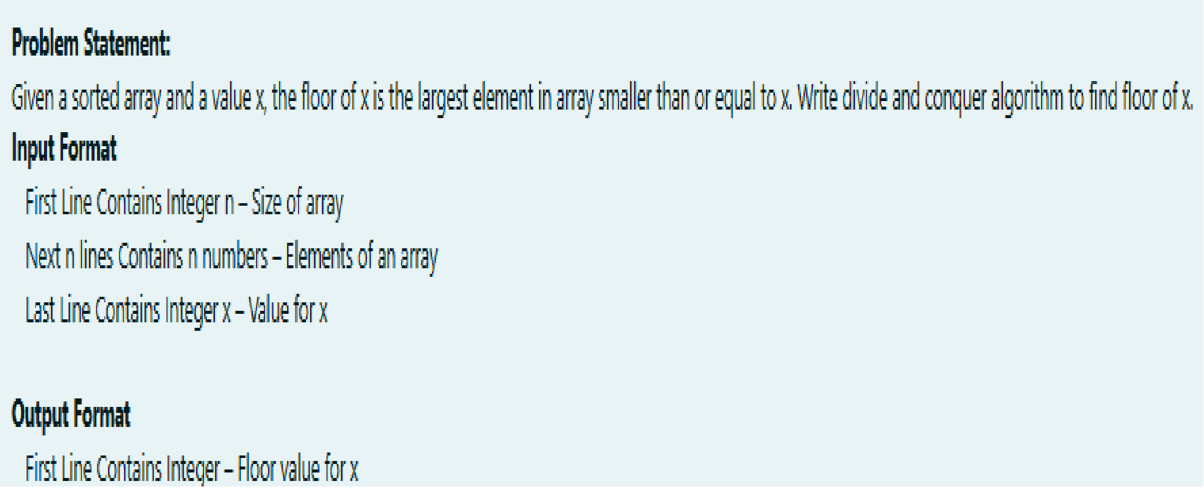
Input	Result
3 3 2 3	3
7 2 2 1 1 1 2 2	2



```c
#include <stdio.h>

int countOccurrences(int nums[], int left, int right, int target) {
    int count = 0;
    for (int i = left; i <= right; i++) {
        if (nums[i] == target) {
            count++;
        }
    }
    return count;
}

int majorityElementRecursive(int nums[], int left, int right) {
    if (left == right) {
        return nums[left];
    }

    int mid = left + (right - left) / 2;
    int leftMajority = majorityElementRecursive(nums, left, mid);
    int rightMajority = majorityElementRecursive(nums, mid + 1, right);

    if (leftMajority == rightMajority) {
        return leftMajority;
    }

    int leftCount = countOccurrences(nums, left, right, leftMajority);
    int rightCount = countOccurrences(nums, left, right, rightMajority);

    return leftCount > rightCount ? leftMajority : rightMajority;
}

int majorityElement(int nums[], int n) {
    return majorityElementRecursive(nums, 0, n - 1);
}

int main() {
    int n;


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


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

    printf("%d\n", majorityElement(nums, n));

    return 0;
}
```



	Input	Expected	Got	



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

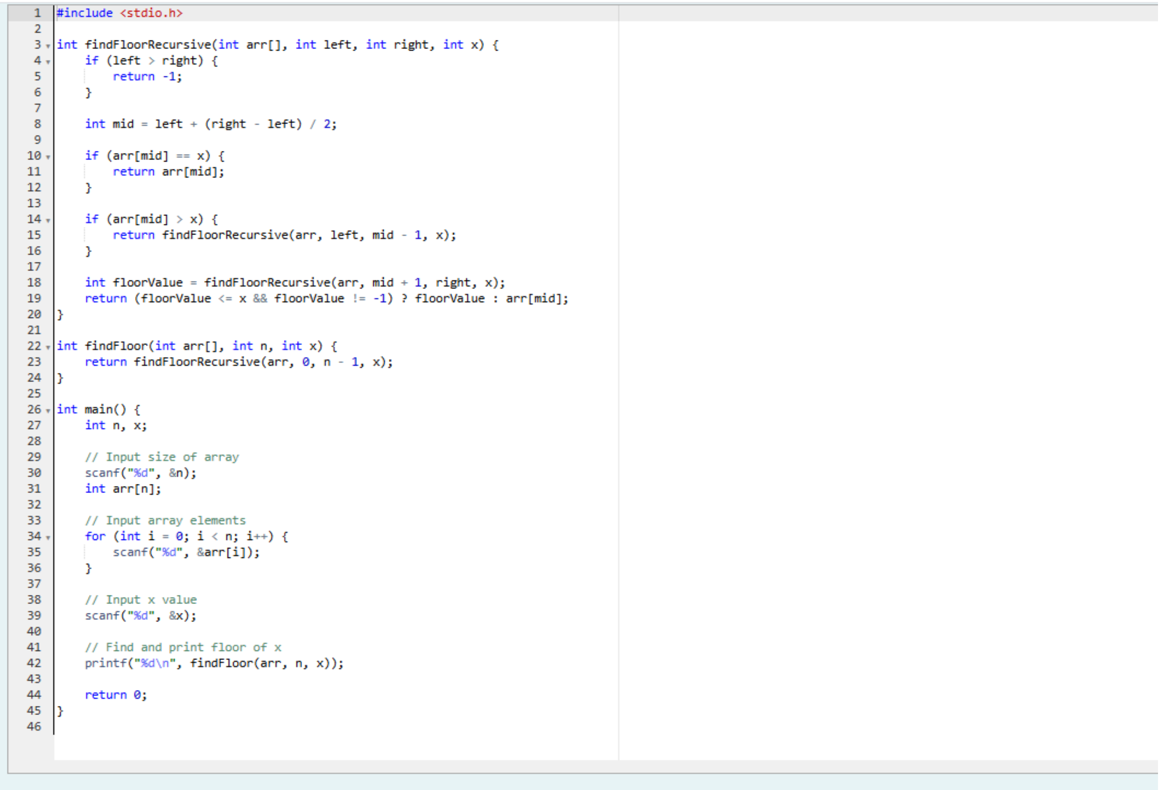
First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

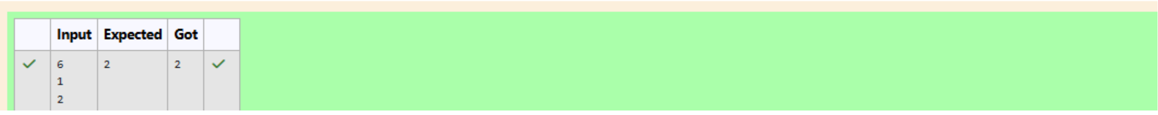
Last Line Contains Integer x – Value for x

Output Format

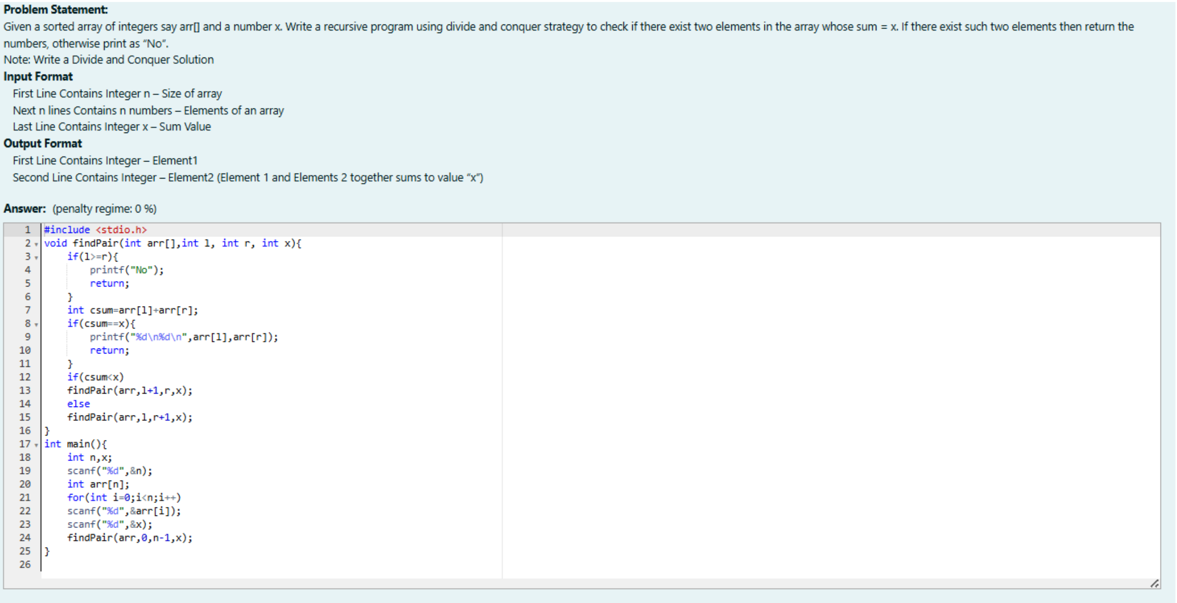
First Line Contains Integer – Floor value for x



```c
#include <stdio.h>

int findFloorRecursive(int arr[], int left, int right, int x) {
    if (left > right) {
        return -1;
    }

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

    if (arr[mid] == x) {
        return arr[mid];
    }

    if (arr[mid] > x) {
        return findFloorRecursive(arr, left, mid - 1, x);
    }

    int floorValue = findFloorRecursive(arr, mid + 1, right, x);
    return (floorValue <= x && floorValue != -1) ? floorValue : arr[mid];
}

int findFloor(int arr[], int n, int x) {
    return findFloorRecursive(arr, 0, n - 1, x);
}

int main() {
    int n, x;

    // Input size of array
    scanf("%d", &n);
    int arr[n];

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

    // Input x value
    scanf("%d", &x);

    // Find and print floor of x
    printf("%d\n", findFloor(arr, n, x));

    return 0;
}

```



	Input	Expected	Got	
✓	6 1 2	2	2	✓



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

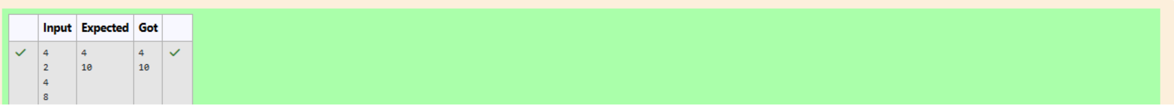
First Line Contains Integer n – Size of array
Next n lines Contains n numbers – Elements of an array
Last Line Contains Integer x – Sum Value

Output Format

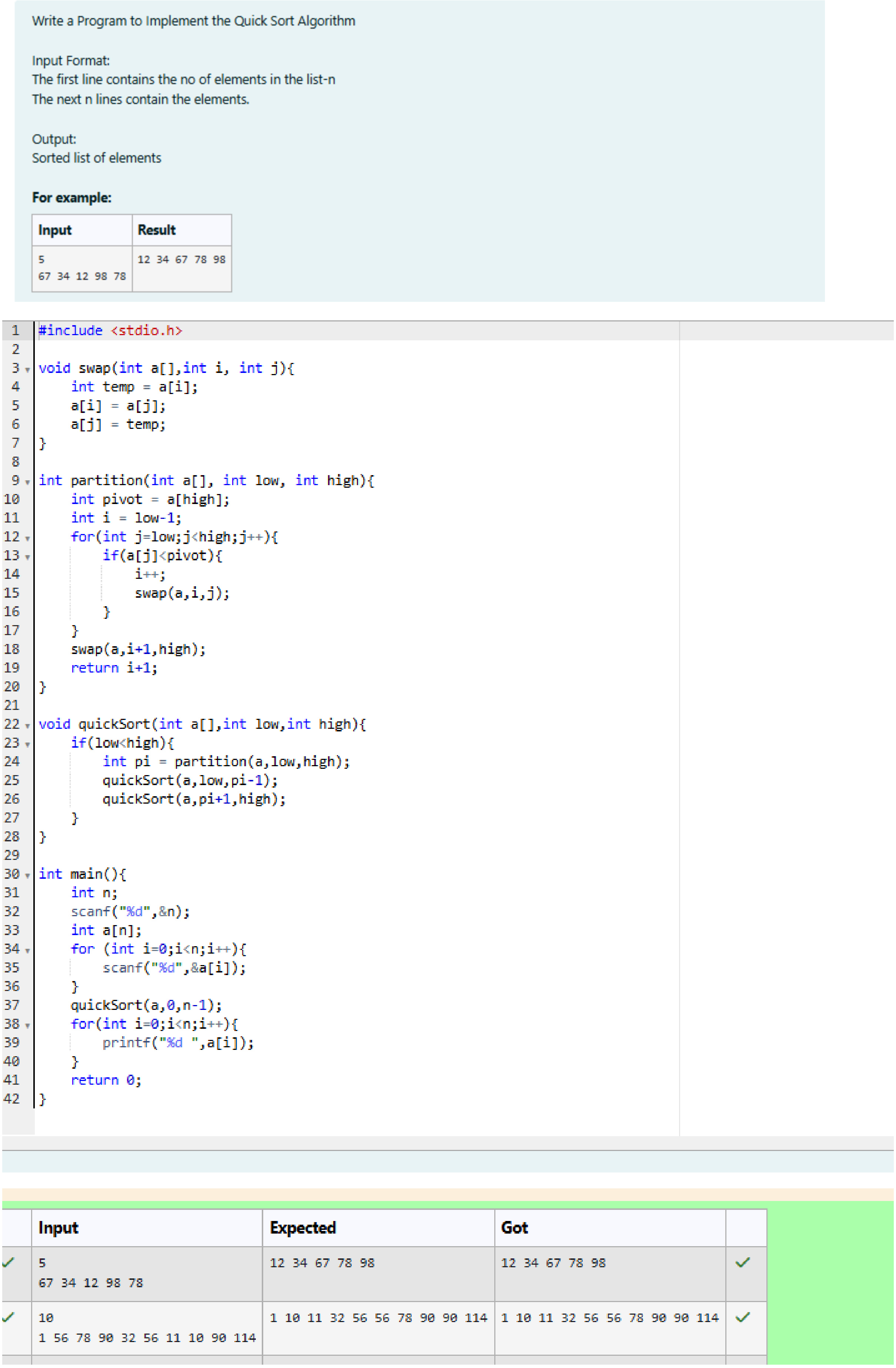
First Line Contains Integer – Element1
Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value "x")

Answer: (penalty regime: 0 %)

```c
#include <stdio.h>
void findPair(int arr[],int l, int r, int x){
    if(l>=r){
        printf("No");
        return;
    }
    int csum=arr[l]+arr[r];
    if(csum==x){
        printf("%d\n%d\n",arr[l],arr[r]);
        return;
    }
    if(csum<x)
    findPair(arr,l+1,r,x);
    else
    findPair(arr,l,r+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);
    findPair(arr,0,n-1,x);
}

```



	Input	Expected	Got	
✓	4 2 4 8	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>

void swap(int a[],int i, int j){
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

int partition(int a[], int low, int high){
    int pivot = a[high];
    int i = low-1;
    for(int j=low;j<high;j++){
        if(a[j]<pivot){
            i++;
            swap(a,i,j);
        }
    }
    swap(a,i+1,high);
    return i+1;
}

void quickSort(int a[],int low,int high){
    if(low<high){
        int pi = partition(a,low,high);
        quickSort(a,low,pi-1);
        quickSort(a,pi+1,high);
    }
}

int main(){
    int n;
    scanf("%d",&n);
    int a[n];
    for (int i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    quickSort(a,0,n-1);
    for(int i=0;i<n;i++){
        printf("%d ",a[i]);
    }
    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 78 90 90 114	✓