GE23131-Programming Using C-2024-2025

Week 6 Assesment

Abenanthan P

240701005

Decision Making and Looping - for loop

Linear Search

Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

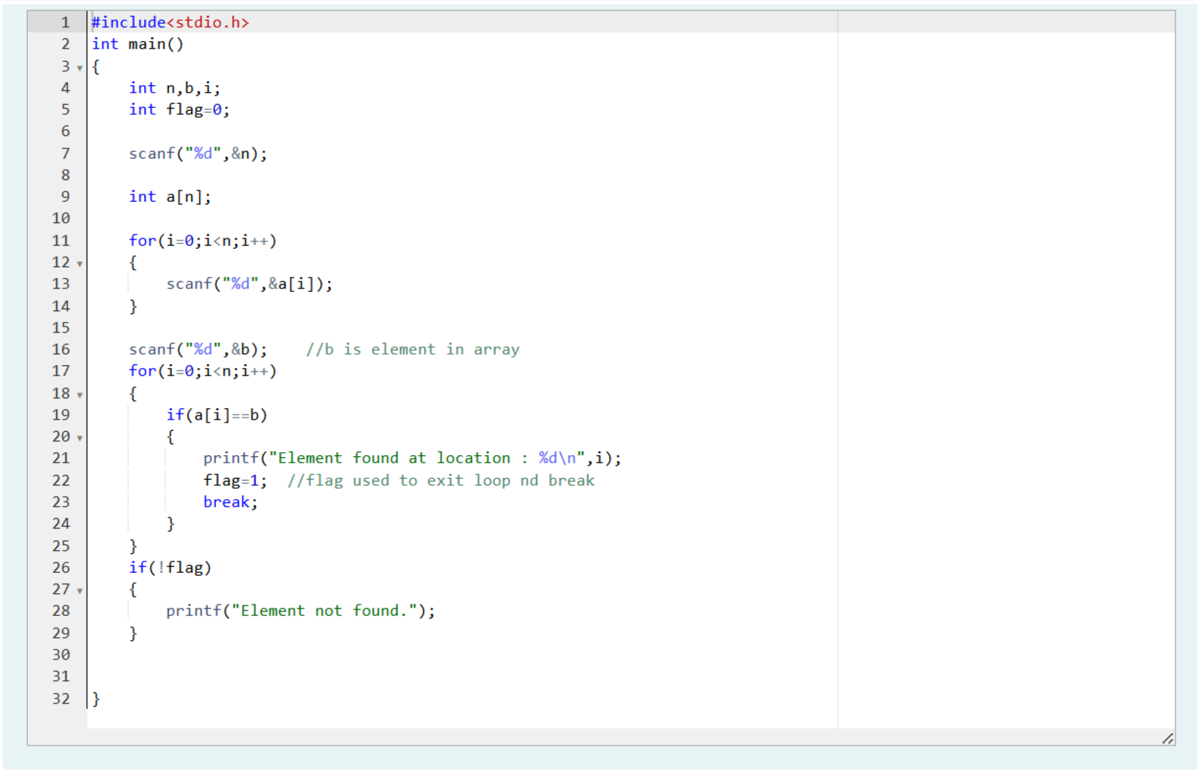
5

30 40 50 20 10

55

Sample Output 2:

Element not found.



```c
#include<stdio.h>
int main()
{
    int n,b,i;
    int flag=0;

    scanf("%d",&n);

    int a[n];

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

    scanf("%d",&b);    //b is element in array
    for(i=0;i<n;i++)
    {
        if(a[i]==b)
        {
            printf("Element found at location : %d\n",i);
            flag=1;  //flag used to exit loop nd break
            break;
        }
    }
    if(!flag)
    {
        printf("Element not found.");
    }


}
```

	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Binary Search

Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

5

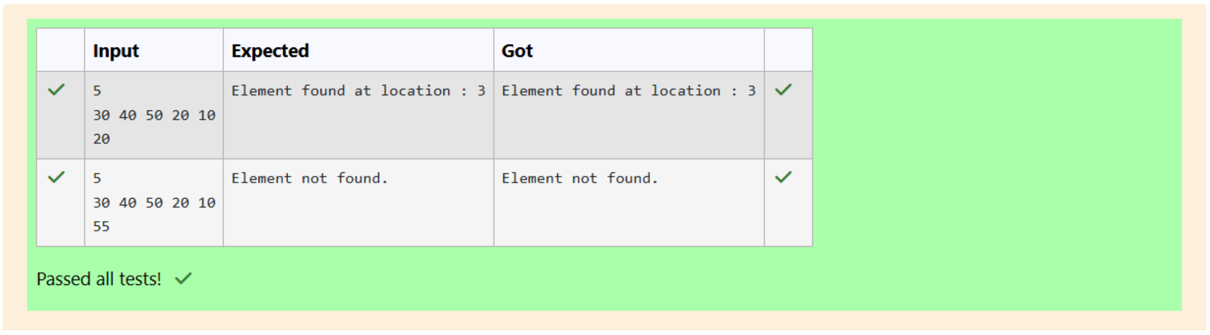
30 40 50 20 10

55

Sample Output 2:

Element not found.

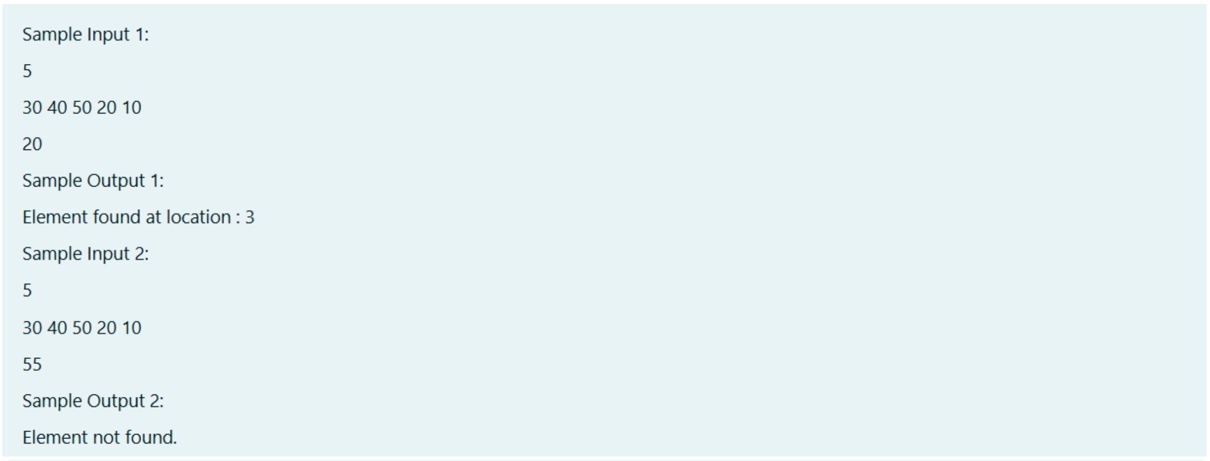
```c
#include<stdio.h>

int binsearch(int arr[],int n,int t)
{
    int low=0,high=n-1;
    while(low<=high){
        int mid=(low+high)/2;
        if(arr[mid]==t)
        return mid;
        if(arr[mid]<t)
        low=mid+1;
        else
        high=mid-1;
    }
    return -1;
}
int main()
{
    int n,t;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    scanf("%d",&t);
    int result=binsearch(arr,n,t);
    if(result!=-1)
    printf("Element found at location : %d\n",result);
    else
    printf("Element not found.\n");
    return 0;
}
```



	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Insertion Sort



Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

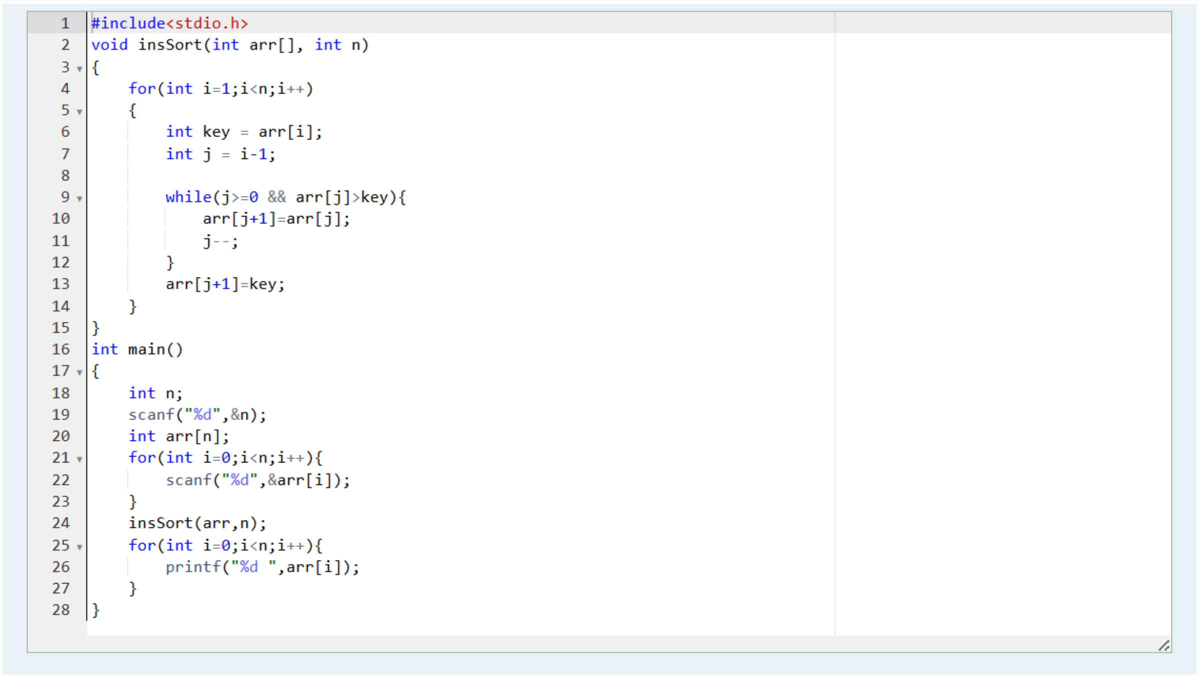
5

30 40 50 20 10

55

Sample Output 2:

Element not found.



```c
#include<stdio.h>
void insSort(int arr[], int n)
{
    for(int i=1;i<n;i++)
    {
        int key = arr[i];
        int j = i-1;

        while(j>=0 && arr[j]>key){
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=key;
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    insSort(arr,n);
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Selection Sort

Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

5

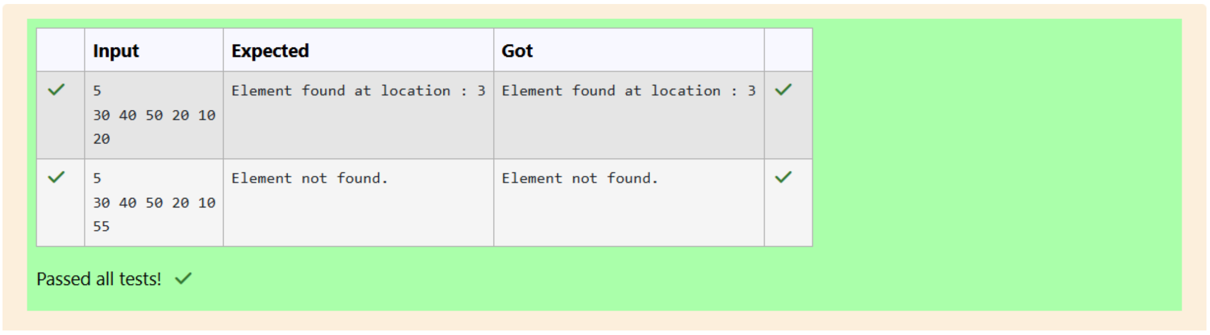
30 40 50 20 10

55

Sample Output 2:

Element not found.

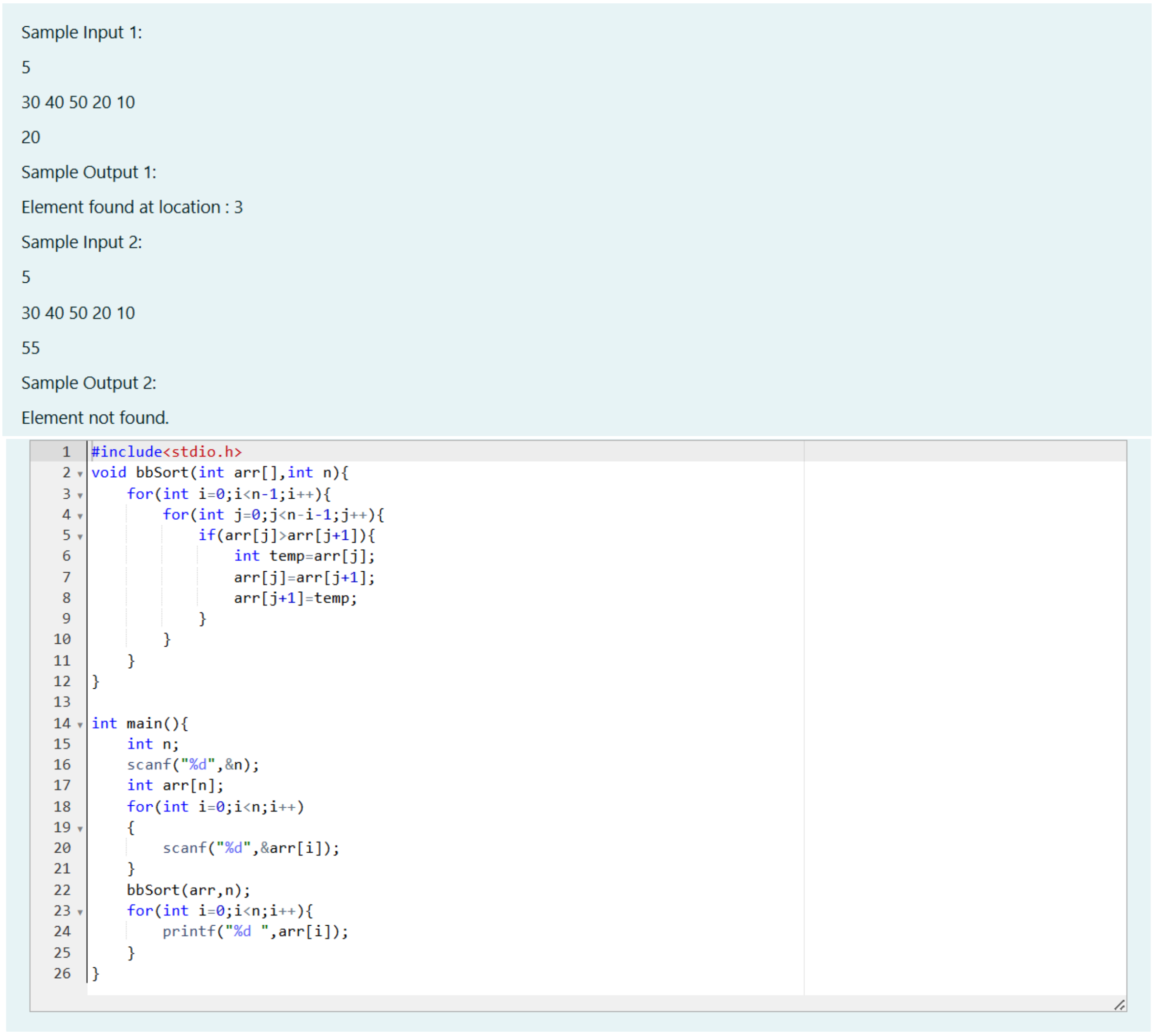
```c
#include<stdio.h>
void selSort(int arr[],int n){
    for(int i=0;i<n-1;i++){
        int minIndex = i;

        for(int j=i+1;j<n;j++){
            if(arr[j]<arr[minIndex]){
                minIndex=j;
            }
        }
        if(minIndex!=i){
            int temp=arr[i];
            arr[i]=arr[minIndex];
            arr[minIndex]=temp;
        }
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    selSort(arr,n);
    for(int i=0;i<n;i++)
    {
        printf("%d ",arr[i]);
    }
}
```



	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Bubble Sort



Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

5

30 40 50 20 10

55

Sample Output 2:

Element not found.

```c
#include<stdio.h>
void bbSort(int arr[],int n){
    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 main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    bbSort(arr,n);
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Quick Sort

Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

5

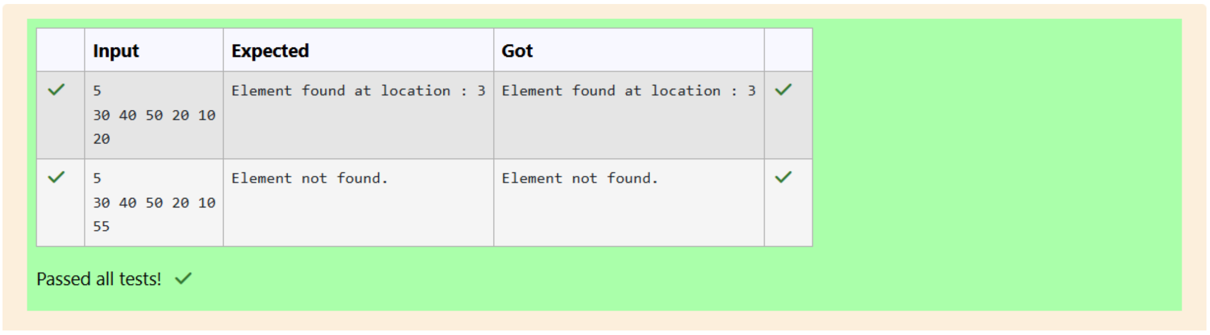
30 40 50 20 10

55

Sample Output 2:

Element not found.

```c
#include <stdio.h>

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++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    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;
}
```



	Input	Expected	Got	
✓	5 30 40 50 20 10 20	Element found at location : 3	Element found at location : 3	✓
✓	5 30 40 50 20 10 55	Element not found.	Element not found.	✓

Passed all tests! ✓

Merge Sort

Sample Input 1:

5

30 40 50 20 10

20

Sample Output 1:

Element found at location : 3

Sample Input 2:

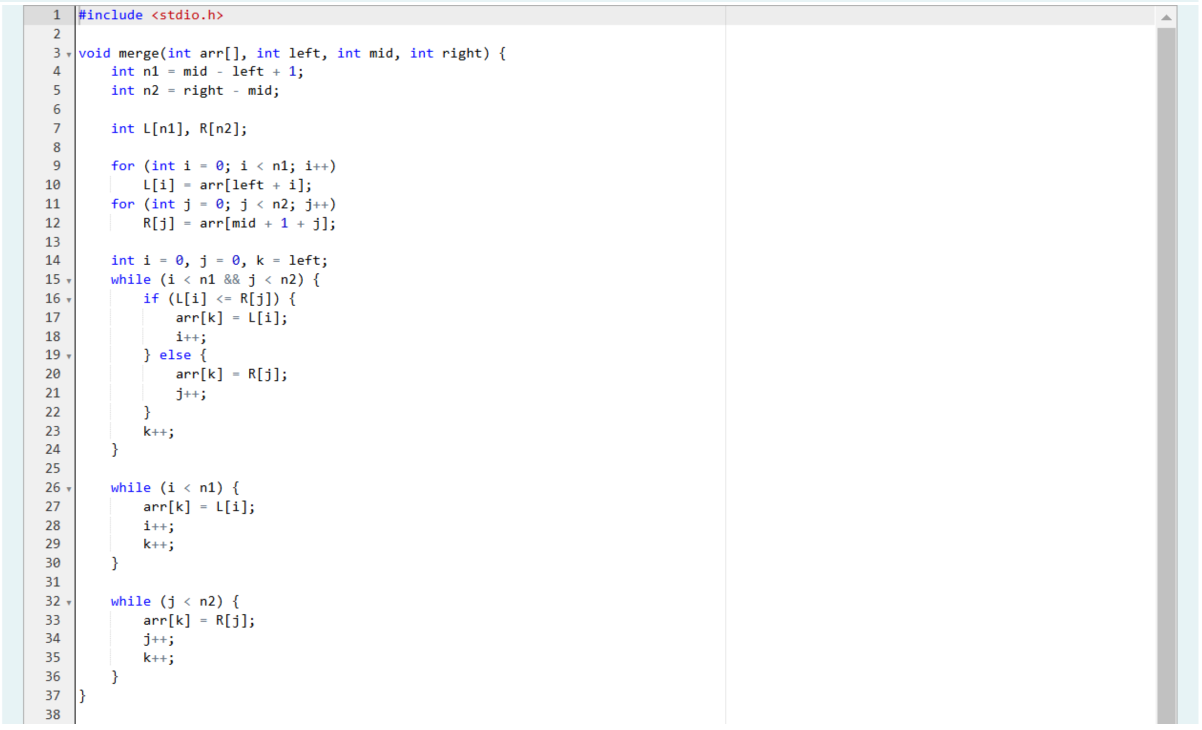
5

30 40 50 20 10

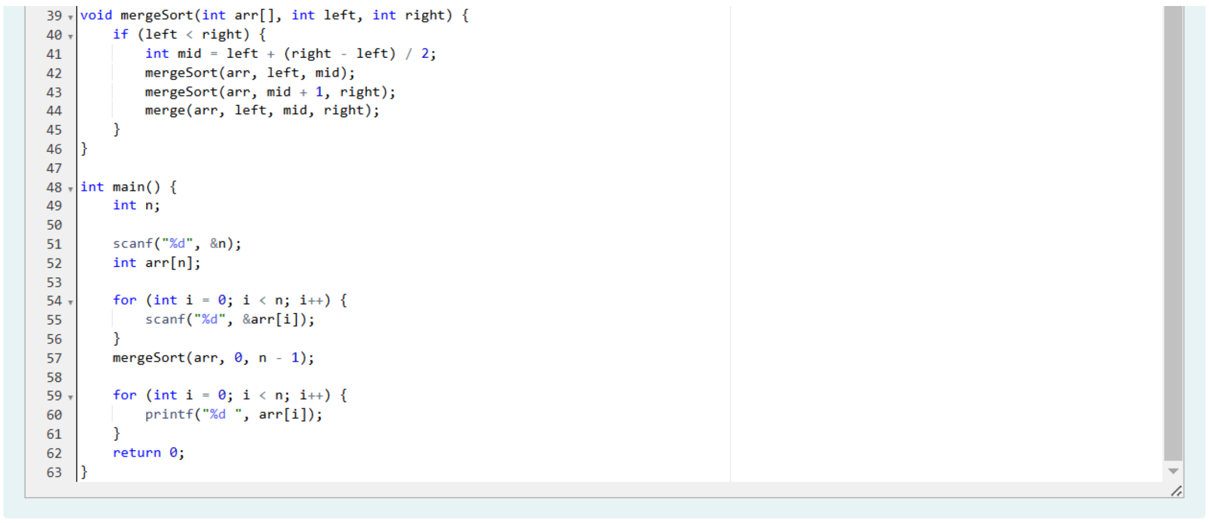
55

Sample Output 2:

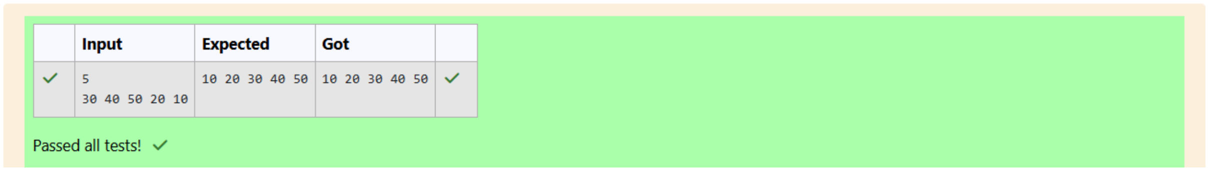
Element not found.



```c
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

```



```c
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int n;

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

    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    mergeSort(arr, 0, n - 1);

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



	Input	Expected	Got	
✓	5 30 40 50 20 10	10 20 30 40 50	10 20 30 40 50	✓

Passed all tests! ✓

Decision Making And For Loop

Objective

In this challenge, we're going to use loops to help us do some simple math. Check out the Tutorial tab to learn more.

Task

Given an integer, ***n***, print its first ***10*** multiples. Each multiple ***n X i*** (where ***1 ≤ i ≤ 10***) should be printed on a new line in the form: n x i = result.

Input Format

A single integer, ***n***.

Constraints

2 ≤ n ≤ 20

Output Format

Print ***10*** lines of output; each line ***i*** (where ***1 ≤ i ≤ 10***) contains the ***result*** of ***n X i*** in the form:

n x i = result.

Sample Input

2

Sample Output

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

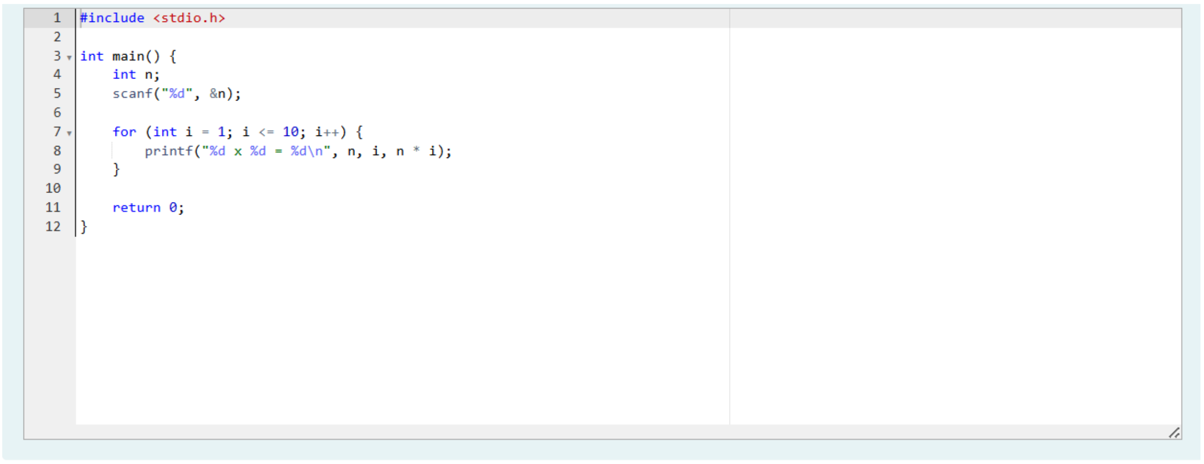
2 x 6 = 12

2 x 7 = 14

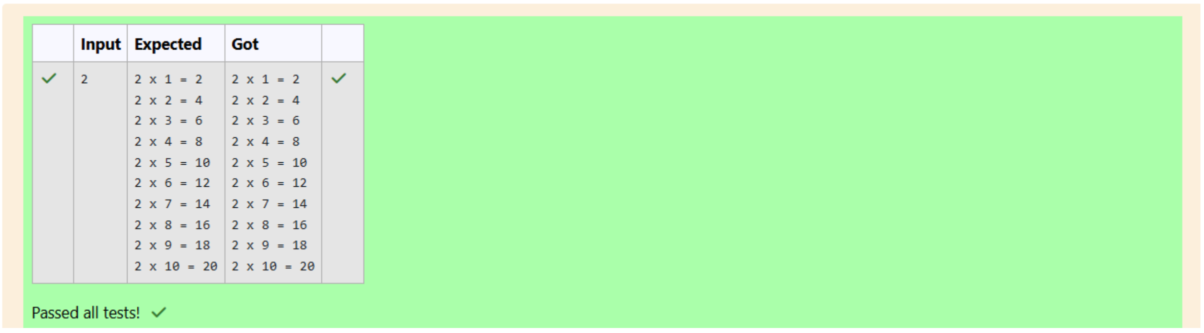
2 x 8 = 16

2 x 9 = 18

2 x 10 = 20



```c
#include <stdio.h>

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

    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", n, i, n * i);
    }

    return 0;
}
```



	Input	Expected	Got	
✓	2	2 x 1 = 2	2 x 1 = 2	✓
		2 x 2 = 4	2 x 2 = 4	
		2 x 3 = 6	2 x 3 = 6	
		2 x 4 = 8	2 x 4 = 8	
		2 x 5 = 10	2 x 5 = 10	
		2 x 6 = 12	2 x 6 = 12	
		2 x 7 = 14	2 x 7 = 14	
		2 x 8 = 16	2 x 8 = 16	
		2 x 9 = 18	2 x 9 = 18	
		2 x 10 = 20	2 x 10 = 20	

Passed all tests! ✓

Decision Making And For Loop

A nutritionist is labeling all the best power foods in the market. Every food item arranged in a single line, will have a value beginning from 1 and increasing by 1 for each, until all items have a value associated with them. An item's value is the same as the number of macronutrients it has. For example, food item with value 1 has 1 macronutrient, food item with value 2 has 2 macronutrients, and incrementing in this fashion.

The nutritionist has to recommend the best combination to patients, i.e. maximum total of macronutrients. However, the nutritionist must avoid prescribing a particular sum of macronutrients (an 'unhealthy' number), and this sum is known. The nutritionist chooses food items in the increasing order of their value. Compute the highest total of macronutrients that can be prescribed to a patient, without the sum matching the given 'unhealthy' number.

Here's an illustration:

Given *4* food items (hence value: *1,2,3* and *4*), and the unhealthy sum being *6* macronutrients, on choosing items *1, 2, 3* -> the sum is *6*, which matches the 'unhealthy' sum. Hence, one of the three needs to be skipped. Thus, the best combination is from among:

- *2 + 3 + 4 = 9*
- *1 + 3 + 4 = 8*
- *1 + 2 + 4 = 7*

Since *2 + 3 + 4 = 9*, allows for maximum number of macronutrients, 9 is the right answer.

Complete the code in the editor below. It must return an integer that represents the maximum total of macronutrients, modulo *1000000007* (*10^9 + 7*).

It has the following:

n: an integer that denotes the number of food items

k: an integer that denotes the unhealthy number

Constraints

- $1 \leq n \leq 2 \times 10^9$
- $1 \leq k \leq 4 \times 10^{15}$

Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of food items.

The second line contains an integer, *k*, that denotes the unhealthy number.

Sample Input 0

2

2

Sample Output 0

3

Explanation 0

The following sequence of $n = 2$ food items:

Sample Input 1

2

1

Sample Output 1

2

Explanation 1

1. Cannot use item *1* because $k = 1$ and $sum \equiv k$ has to be avoided at any time.
2. Hence, max total is achieved by $sum = 0 + 2 = 2$.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    long long int n,t,i,nut=0;
    scanf("%lld %lld",&n,&t);
    for(i=1;i<=n;i++)
    {
        nut=nut+i;
        if(nut==t)
        {
            nut=nut-1;
        }
    }
    printf("%lld",nut%1000000007);
}
```

	Input	Expected	Got	
✓	2 2	3	3	✓
✓	2 1	2	2	✓
✓	3 3	5	5	✓

Passed all tests! ✓

Decision Making And For Loop

Determine all positive integer values that evenly divide into a number, its factors. Return the p^{th} element of your list, sorted ascending. If there is no p^{th} element, return *0*.

For example, given the number $n = 20$, its factors are *{1,2,4,5,10,20}*. Using **1-based indexing** if $p = 3$, return *4*. If $p > 6$, return *0*.

Complete the code in the editor below. The function should return a long integer value of the p^{th} integer factor of *n*.

It has the following:

n: an integer

p: an integer

Constraints

- $1 \leq n \leq 10^{15}$
- $1 \leq p \leq 10^{9}$

Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the number to factor.

The second line contains an integer *p*, the 1-based index of the factor to return.

Sample Input 1

```
10
5
```

Sample Output 1

```
0
```

Explanation 1

Factoring $n = 10$ we get *{1, 2, 5, 10}*. There are only *4* factors and $p = 5$. We return *0* as our answer.

Sample Input 2

```
1
1
```

Sample Output 2

```
1
```

Explanation 2

Factoring $n = 1$ we get *{1}*. We then return the $p = 1^{st}$ factor as our answer.

Answer: (penalty regime: 0 %)

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

long long find_pth_divisor(long long n, long long p) {
    long long divisors[1000000];
    int count = 0;

    for (long long i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            divisors[count++] = i;
            if (i != n / i) {
                divisors[count++] = n / i;
            }
        }
    }

    for (int i = 0; i < count - 1; i++) {
        for (int j = i + 1; j < count; j++) {
            if (divisors[i] > divisors[j]) {
                long long temp = divisors[i];
                divisors[i] = divisors[j];
                divisors[j] = temp;
            }
        }
    }

    if (p <= count) {
        return divisors[p - 1];
    } else {
        return 0;
    }
}

int main() {
    long long n, p;
    scanf("%lld", &n);
    scanf("%lld", &p);
    long long result = find_pth_divisor(n, p);
    printf("%lld\n", result);
    return 0;
}
```

	Input	Expected	Got	
✓	10 3	5	5	✓
✓	10 5	0	0	✓
✓	1 1	1	1	✓

Passed all tests! ✓

One Dimensional Arrays

Given an array of numbers and a window of size k. Print the maximum of numbers inside the window for each step as the window moves from the beginning of the array.

Input Format

Input contains the array size, no of elements and the window size

Output Format

Print the maximum of numbers

Constraints

1 <= size <= 1000

Sample Input 1

8

1 3 5 2 1 8 6 9

3

Sample Output 1

5 5 5 8 8 9

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>

void findMaxInSlidingWindow(int arr[],int n,int k)
{
    int max;
    for(int i=0;i<=n-k;i++)
    {
        max=arr[i];
        for(int j=1;j<k;j++)
        {
            if(arr[i+j]>max)
            {
                max=arr[i+j];
            }
        }
        printf("%d ",max);
    }
    printf("\n");
}
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);
    findMaxInSlidingWindow(arr,n,k);
    return 0;
}
```

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

Passed all tests! ✓

One Dimensional Arrays

Given an array and a threshold value find the output.

Input: {5,8,10,13,6,2}

Threshold = 3

Output count = 17

Explanation:

Number	Parts	Counts
5	{3,2}	2
8	{3,3,2}	3
10	{3,3,3,1}	4
13	{3,3,3,3,1}	5
6	{3,3}	2
2	{2}	1

Input Format

N - no of elements in an array

Array of elements

Threshold value

Output Format

Display the count

Sample Input 1

```
6
5 8 10 13 6 2
3
```

Sample Output 1

```
17
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int n,tres,totp=0;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    scanf("%d",&tres);
    for(int i=0;i<n;i++)
    {
        totp+=(arr[i]+tres-1)/tres;
    }
    printf("%d\n",totp);
}
```

	Input	Expected	Got	
✓	6 5 8 10 13 6 2 3	17	17	✓
✓	7 20 35 57 30 56 87 30 10	33	33	✓

Passed all tests! ✓

One Dimensional Arrays

Output is a merged array without duplicates.

Input Format

N1 - no of elements in array 1

Array elements for array 1

N2 - no of elements in array 2

Array elements for array2

Output Format

Display the merged array

Sample Input 1

5

1 2 3 6 9

4

2 4 5 10

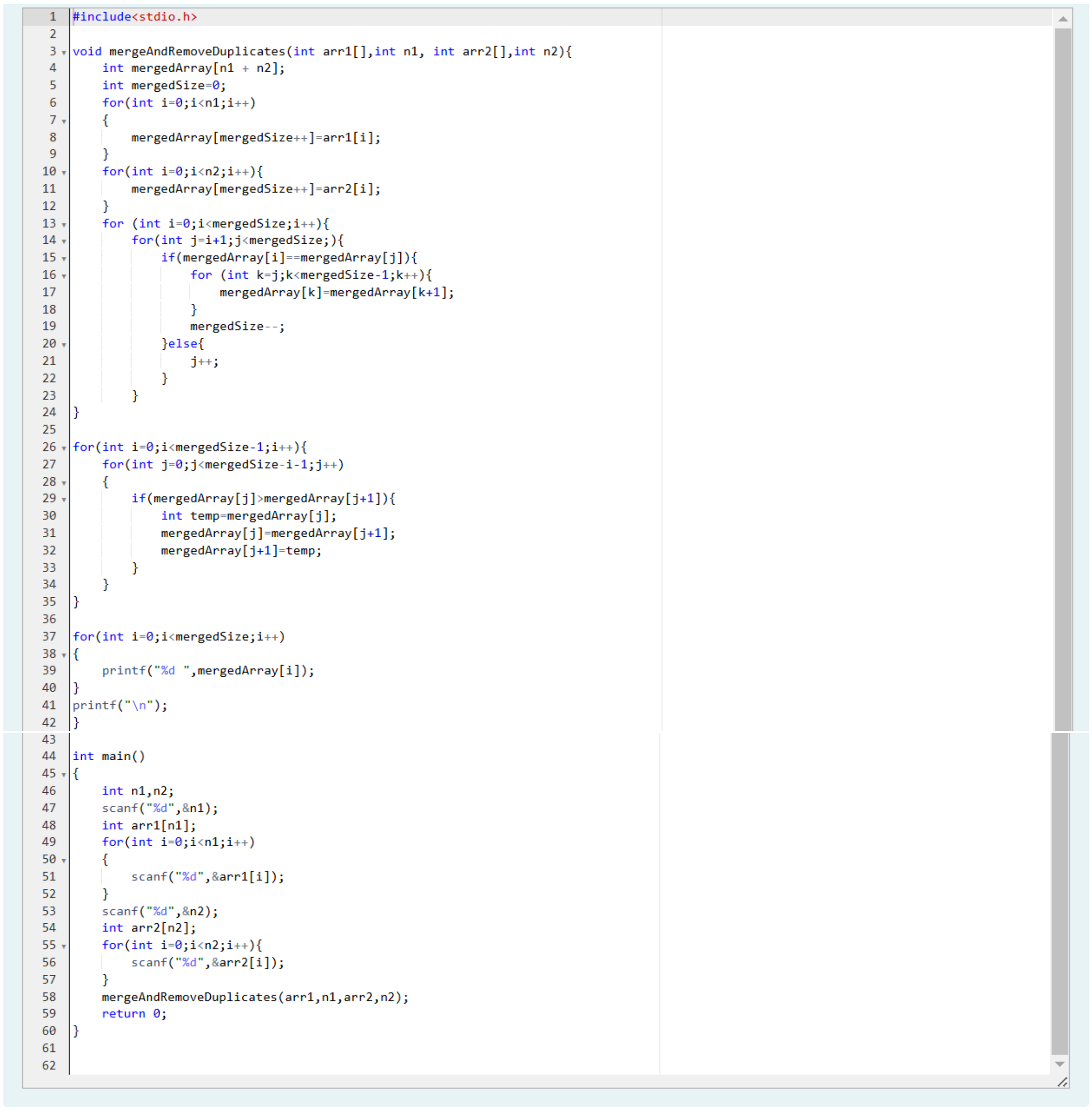
Sample Output 1

1 2 3 4 5 6 9 10

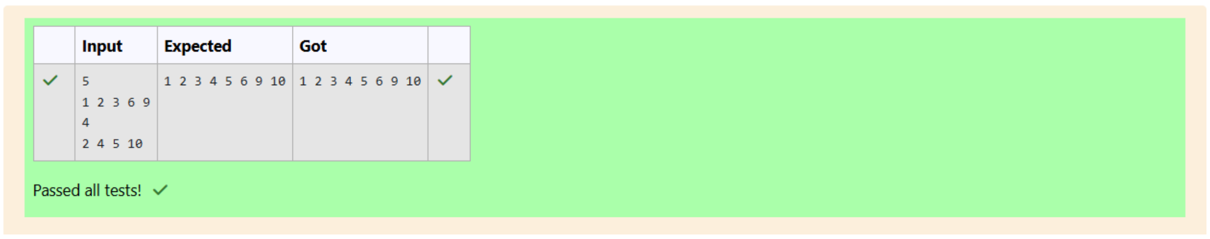
For example:

Input	Result

5
1 2 3 6 9
4
2 4 5 10
	1 2 3 4 5 6 9 10



```c
#include<stdio.h>

void mergeAndRemoveDuplicates(int arr1[],int n1, int arr2[],int n2){
    int mergedArray[n1 + n2];
    int mergedSize=0;
    for(int i=0;i<n1;i++)
    {
        mergedArray[mergedSize++]=arr1[i];
    }
    for(int i=0;i<n2;i++){
        mergedArray[mergedSize++]=arr2[i];
    }
    for (int i=0;i<mergedSize;i++){
        for(int j=i+1;j<mergedSize;){
            if(mergedArray[i]==mergedArray[j]){
                for (int k=j;k<mergedSize-1;k++){
                    mergedArray[k]=mergedArray[k+1];
                }
                mergedSize--;
            }else{
                j++;
            }
        }
}

for(int i=0;i<mergedSize-1;i++){
    for(int j=0;j<mergedSize-i-1;j++)
    {
        if(mergedArray[j]>mergedArray[j+1]){
            int temp=mergedArray[j];
            mergedArray[j]=mergedArray[j+1];
            mergedArray[j+1]=temp;
        }
    }
}

for(int i=0;i<mergedSize;i++)
{
    printf("%d ",mergedArray[i]);
}
printf("\n");
}

int main()
{
    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]);
    }
    mergeAndRemoveDuplicates(arr1,n1,arr2,n2);
    return 0;
}
```



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

Passed all tests! ✓