

**Ex. No: 6**

**Date: 17.09.24**

**Register No.: 230701346**

**Name: Subasri V**

---

## **Competitive Programming**

### **6.a. Finding Duplicates- $O(n^2)$ Time Complexity (1) Space Complexity**

**Aim:** 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

**Algorithm:**

1. Read the integer  $n$  (size of array)
2. Initialize array  $a[]$  of size  $n$
3. For  $i = 0$  to  $n-1$  do:  
     $a[i] = \text{input value}$  (read each element of the array)
4. For  $i = 0$  to  $n-1$  do:  
    For  $j = i+1$  to  $n-1$  do:  
        If  $a[i] == a[j]$  then:

Print a[i]

**Program:**

```
#include<stdio.h>

int main()
{
    int n,b;

    scanf("%d",&n);

    int a[n];

    for(int i=0;i<n;i++)
    {
        scanf("%d",&b);

        a[i]=b;
    }

    for(int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(a[i]==a[j])
            {
                printf("%d",a[j]);
            }
        }
    }
}
```

**Output:**

	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	✓

## 6.b. Finding Duplicates- $O(n)$ Time Complexity (1) Space Complexity

**Aim:** 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 **Algorithm:**

```
function main()
```

```
{
```

```
    initialize n // Number of elements in the array
```

```
    read n from user
```

```
    initialize a[n] // Array to hold input values
```

```
    // Read values into the array
```

```
    for i from 0 to n - 1
```

```
    {
```

```
        read a[i] from user
```

```
    }
```

```
    initialize b[n] // Array to keep track of seen elements
```

```
    for i from 0 to n - 1
```

```
    {
```

```
        b[i] = 0 // Initialize the tracking array
```

```

    }

    // Search for the first duplicate element
    for i from 0 to n - 1
    {
        // If the element is already present, i.e., b[a[i]] = 1
        if b[a[i]]
        {
            print a[i] // Print the duplicate element
            break // Exit the loop
        }
    }
    else
    {
        b[a[i]] = 1 // Mark the element as seen
    }
}

```

### **Program:**

```

#include <stdio.h> int
main(){
    int n;

    scanf("%d",&n);

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

    }

```

```

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

    }

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

        //if el already present i.e, b[i]=1
if(b[a[i]]){        printf("%d",a[i]);
break;    }    else
b[a[i]]=1;

    }

}

```

**Output:**

	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	✓

### 6.c. Print Intersection of 2 sorted arrays- $O(m*n)$ Time Complexity, $O(1)$ Space Complexity

**Aim:**

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
```

### Algorithm

1. Read the integer

n (size of array)

2. Initialize array a[]

of size n

3. For i = 0 to n-1

do:

    Read a[i] from  
input

4. For i = 0 to n-2

do:

    If a[i] == a[i+1]  
then:

        Print a[i]

        Break the loop

5. If no duplicate is  
found, do nothing  
or handle it as  
needed

**Program:**

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

```
int main()
```

```
{
```

```
    int n,b;
```

```
    scanf("%d",&n);
```

```
    int a[n];
```

```
    for(int i=0;i<=n;i++)
```



```

{
    scanf("%d",&b);

    a[i]=b;
}
for(int i=0;i<n;i++)
{
    if(a[i]==a[i+1])
    {
        printf("%d",a[i+1]);

        break;
    }
}
}

```

**Output:**

	Input	Expected	Got	
✓	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✓
✓	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✓

#### 6.d. Print Intersection of 2 sorted arrays- $O(m+n)$ Time Complexity, $O(1)$ Space Complexity

**Aim:**

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
```

**Algorithm:**

1. Read the integer T  
(number of test cases).

2. For each test case

(T times):

a. Read  $n_1$  (size of  $arr_1$ ) and  $n_2$  (size of  $arr_2$ ).

b. Initialize array  $arr_1[]$  of size  $n_1$ .

c. Read  $n_1$  integers into  $arr_1[]$ .

d. Initialize array  $arr_2[]$  of size  $n_2$ .

e. Read  $n_2$  integers into  $arr_2[]$ .

f. Initialize indices  $i = 0$  and  $j = 0$ .

g. While  $i < n_1$  and  $j < n_2$ :

i. If  $arr_1[i] < arr_2[j]$ , increment  $i$ .

ii. If  $arr_1[i] > arr_2[j]$ , increment  $j$ .

iii. If arr1[i] ==  
arr2[j], print arr1[i],  
increment both i  
and j.

h. Print newline  
after the results of  
each test case.

3. End.

**Program:**

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

```
int main() {  
    int T;  
    scanf("%d", &T);  
    while (T-->0) {  
        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]);
    }

    int i = 0, j = 0;
    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            i++;
        }
        else if (arr2[j] < arr1[i]) {
            j++;
        }
        else {
            printf("%d ", arr1[i]);
            i++;
            j++;
        }
    }
    printf("\n");
}
}

```

**Output:**

	Input	Expected	Got	
✓	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✓
✓	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✓

## 6.e. Pair with Difference- $O(n^2)$ Time Complexity, $O(1)$ Space Complexity

### Aim:

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

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.

### Algorithm:

1. Read the integer n (size of array).
2. Initialize array arr[] of size n.
3. For i = 0 to n-1:

- a. Read arr[i] from input.
4. Read the integer t (target absolute difference).
5. Initialize flag = 0.
6. Initialize i = 0 and j = 1.
7. While i < n and j < n:
  - a. Compute diff = abs(arr[i] - arr[j]).
  - b. If i != j and diff == t:
    - i. Set flag = 1.
    - ii. Break the loop.
  - c. Else if diff < t, increment j.
  - d. Else, increment i.
8. If flag is set to 1:
  - a. Print 1.
9. Else:
  - a. Print 0.

**Program:**

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

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

```
int main() {
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    int arr[n];
```

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

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

```
int t;  
scanf("%d", &t);
```

```
int flag = 0;
```

```
int i=0;  
int j=1;  
while(i<n && j<n){  
    int diff = abs(arr[i] - arr[j]);  
    if(i!=j && diff==t){  
        flag=1;  
        break;  
    }  
    else if(diff<t){  
        j++;  
    }  
    else{  
        i++;  
    }  
}
```

```
if (flag) {
```



```

        printf("%d\n", 1);
    } else {
        printf("%d\n", 0);
    }

    return 0;
}

```

### Output:

	Input	Expected	Got	
✓	3 1 3 5 4	1	1	✓
✓	10 1 4 6 8 12 14 15 20 21 25 1	1	1	✓
✓	10 1 2 3 5 11 14 16 24 28 29 0	0	0	✓
✓	10 0 2 3 7 13 14 15 20 24 25 10	1	1	✓

## 6.f. Pair with Difference - $O(n)$ Time Complexity, $O(1)$ Space Complexity

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

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.

### Algorithm:

1. Read the integer n (size of array).
2. Initialize array arr[] of size n.
3. For i = 0 to n-1:
  - a. Read arr[i] from input.
4. Read the integer t (target absolute difference).
5. Initialize flag = 0.
6. For i = 0 to n-1:
  - a. For j = 0 to n-1:
    - i. If  $i \neq j$  and  $\text{abs}(\text{arr}[i] - \text{arr}[j]) == t$ :
      - A. Set flag = 1.
      - B. Break the inner loop.

b. If flag = 1, break the outer loop.

7. If flag is set to 1:

a. Print 1.

8. Else:

a. Print 0.

**Program:**

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

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

```
int main() {
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    int arr[n];
```

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

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

```
    }
```

```
    int t;
```

```
    scanf("%d", &t);
```

```
    int flag = 0;
```

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

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

```
        if (i!=j && abs(arr[i] - arr[j]) == t) {  
            flag = 1;  
            break;  
        }  
    }  
    if (flag) {  
        break;  
    }  
}  
  
if (flag) {  
    printf("%d\n", 1);  
} else {  
    printf("%d\n", 0);  
}  
  
return 0;  
}
```

**Output:**

	Input	Expected	Got	
✓	3 1 3 5 4	1	1	✓
✓	10 1 4 6 8 12 14 15 20 21 25 1	1	1	✓
✓	10 1 2 3 5 11 14 16 24 28 29 0	0	0	✓
✓	10 0 2 3 7 13 14 15 20 24 25 10	1	1	✓