Ex. No: 6 **Date: 17.09.24**

Register No.: 230701368 **Name: AL UMA**

COMPETITIVE PROGRAMMING

6.A Finding Duplicates-O(n^2) Time Complexity ,O(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

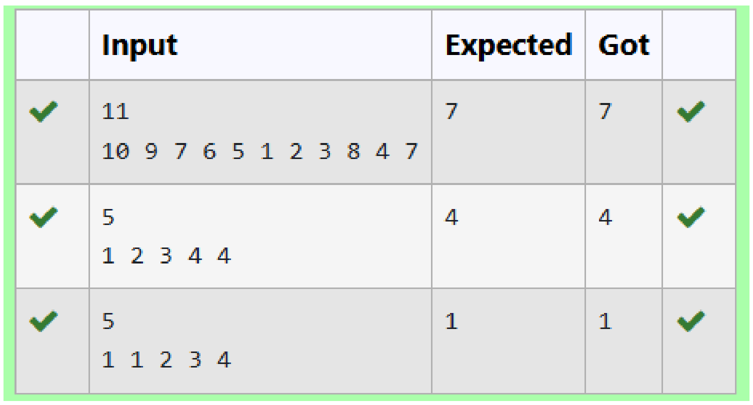
ALGORITHM:

```
Function main()
    // Step 1: Read the number of elements n
    Initialize n  // Number of elements
    Read n from user  // Input the value of n

    // Step 2: Initialize the array a with size n
    Initialize array a of size n
    // Step 3: Read the elements into the array a
    For i from 0 to n-1  // Loop over the range [0, n-1]
        Read a[i] from user  // Input each element into a[i]
    End For

    Initialize c = 0  // Counter to track if a duplicate is found

    // Step 4: Find and print the first duplicate element
    For i from 0 to n-1  // Loop over the array for the first element
        For j from 0 to n-1  // Loop over the array for the second element
            If i != j AND a[i] == a[j]  // Check if a duplicate is found
                Print a[i]  // Print the duplicate value
                Increment c by 1  // Mark that a duplicate was found
                Break the inner loop  // Break out of the inner loop once duplicate is
found
            End If
        End For

        If c == 1  // Check if a duplicate has already been found
            Break the outer loop  // Break out of the outer loop if duplicate is found
        End If
    End For
End Function
Detailed
```

PROGRAM:

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

    scanf("%d",&n);

    int a[n];

    int d,index;

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

    {

        scanf("%d",&d);

        index=d%n;

        if(a[index]!=0 && a[index]==d)

        {

            printf("%d",a[index]);

            break;


        }

        a[index]=d;


    }


}
```

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:

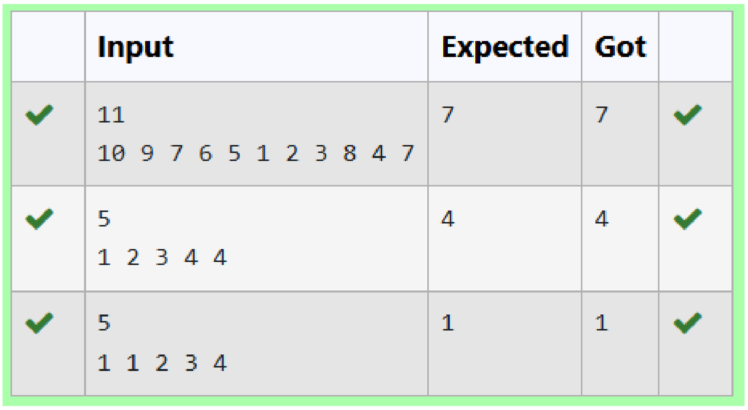
```
Function main()
   // Step 1: Read the number of elements n
   Initialize n  // Number of elements
   Read n from user  // Input the value of n

   // Step 2: Initialize an array a of size n with all elements set to 0
   Initialize array a of size n, where all elements are initially set to 0

   // Step 3: For each input element, check for the first duplicate
   For i from 0 to n-1  // Loop over the range [0, n-1]
      Read d from user  // Input the element d

      // Step 4: Compute the index as d % n
      Set index = d % n  // Find the index where the element should be stored
```

```
        // Step 5: Check if the element at a[index] has been set before and if it equals d
        If a[index] != 0 AND a[index] == d  // Check if there is a match (duplicate)
            Print a[index]  // Print the duplicate element
            Break the loop  // Exit the loop after printing the duplicate
        End If

        // Step 6: Set the element at a[index] to d
        Set a[index] = d  // Mark this element in the array

    End For
End Function
```

PROGRAM:

```
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    int d,index;
    for(int i=0;i<n;i++)
    {
        scanf("%d",&d);
        index=d%n;
        if(a[index]!=0 && a[index]==d)
        {
            printf("%d",a[index]);
            break;
        }
        a[index]=d;

    }

}
```

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 3-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

```
{
```

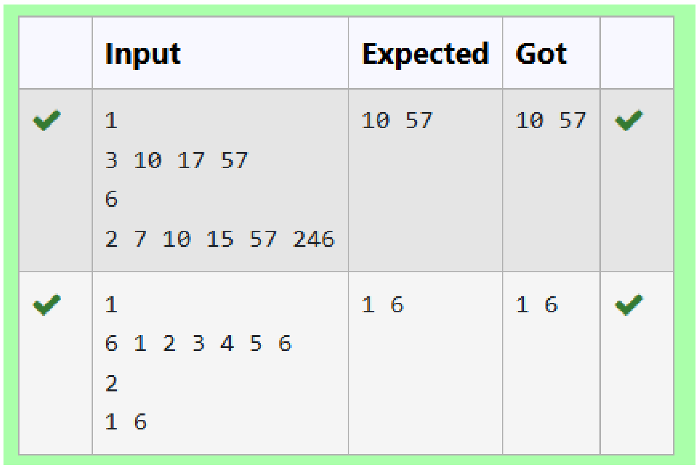
ALGORITHM

```
function main()
{
    initialize n  // Number of test cases
    read n from user

    for i from 0 to n - 1
    {
        initialize n1  // Size of the first array
        read n1 from user

        initialize arr1[n1]  // First array

        // Read values into the first array
        for j from 0 to n1 - 1
        {
            read arr1[j] from user
        }

        initialize n2  // Size of the second array
        read n2 from user

        initialize arr2[n2]  // Second array

        // Read values into the second array
        for j from 0 to n2 - 1
        {
            read arr2[j] from user
        }

        // Check for common elements in both arrays
        for j from 0 to n1 - 1
        {
            for k from 0 to n2 - 1
            {
                if arr1[j] == arr2[k]
                {
                    print arr1[j]  // Print the common element
                }
            }
        }
    }
}
```

PROGRAM:

```c
#include<stdio.h>
int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int n1;
        scanf("%d",&n1);
        int arr1[n1];
        for(int j=0;j<n1;j++){
            scanf("%d ",&arr1[j]);
        }
        int n2;
        scanf("%d",&n2);
        int arr2[n2];
        for(int j=0;j<n2;j++){
            scanf("%d ",&arr2[j]);
        }
        for(int j=0;j<n1;j++){
            for(int k=0;k<n2;k++){
                if(arr1[j]==arr2[k]){
                    printf("%d ",arr1[j]);
                }
            }
        }
    }

}
```

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:

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

.

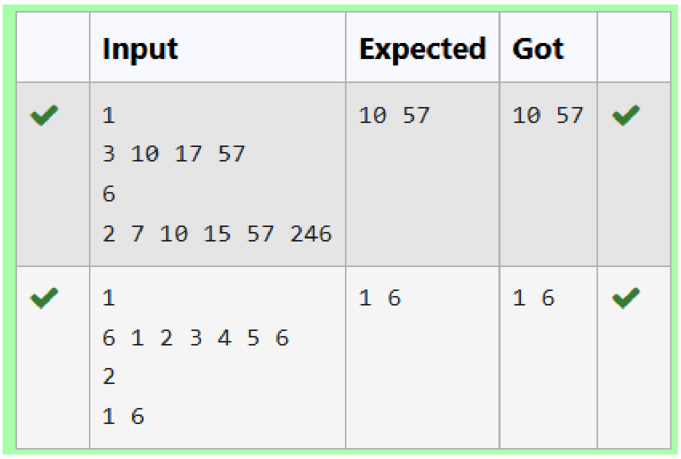
ALGORITHM:

```
function main()
{
    initialize T  // Number of test cases
    read T from user

    while T > 0
    {
        // Decrement the test case counter
        T--

        initialize n1, n2  // Sizes of the two arrays
        read n1 from user
        initialize arr1[n1]  // First array

        // Read values into the first array
        for i from 0 to n1 - 1
        {
            read arr1[i] from user
        }

        read n2 from user
        initialize arr2[n2]  // Second array

        // Read values into the second array
        for i from 0 to n2 - 1
        {
            read arr2[i] from user
        }

        initialize i = 0, j = 0  // Indices for both arrays

        // Iterate through both arrays to find common elements
        while i < n1 and j < n2
        {
            if arr1[i] < arr2[j]
            {
                i++  // Move to the next element in arr1
            }
            else if arr2[j] < arr1[i]
            {
                j++  // Move to the next element in arr2
            }
            else
            {
                print arr1[i]  // Print the common element
                i++  // Move to the next element in arr1
                j++  // Move to the next element in arr2
            }
        }

        print new line  // Move to the next line for output
    }
}
```

PROGRAM

```
#include <stdio.h>

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]);
        }

        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 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 != 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:

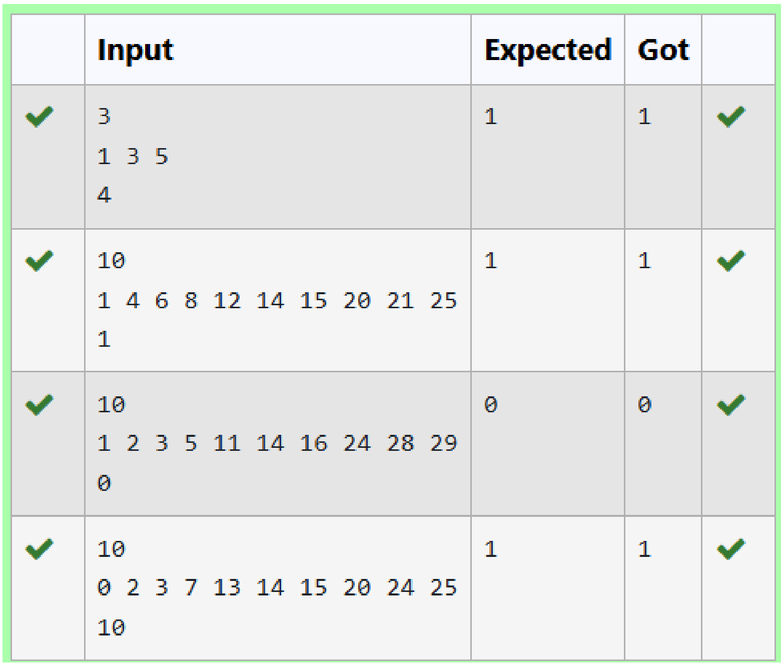
```
function main()
{
    initialize n  // Number of elements in the array
    read n from user

    initialize arr[n]  // Array to hold input values

    // Read values into the array
    for i from 0 to n - 1
```

```
{
    read arr[i] from user
}

initialize t  // Target difference
read t from user

initialize flag = 0  // Flag to indicate if a pair is found

// Check for pairs with the specified difference
for i from 0 to n - 1
{
    for j from 0 to n - 1
    {
        if i != j and abs(arr[i] - arr[j]) == t
        {
            flag = 1  // Pair found
            break
        }
    }
    if flag
    {
        break
    }
}

// Output the result based on the flag
if flag
{
    print 1  // Pair found
    }
    else
    {
        print 0  // No pair found
    }

    return 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	✓

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 != 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:

```
function main()
{
   initialize n  // Number of elements in the array
   read n from user

   initialize arr[n]  // Array to hold input values

   // Read values into the array
```

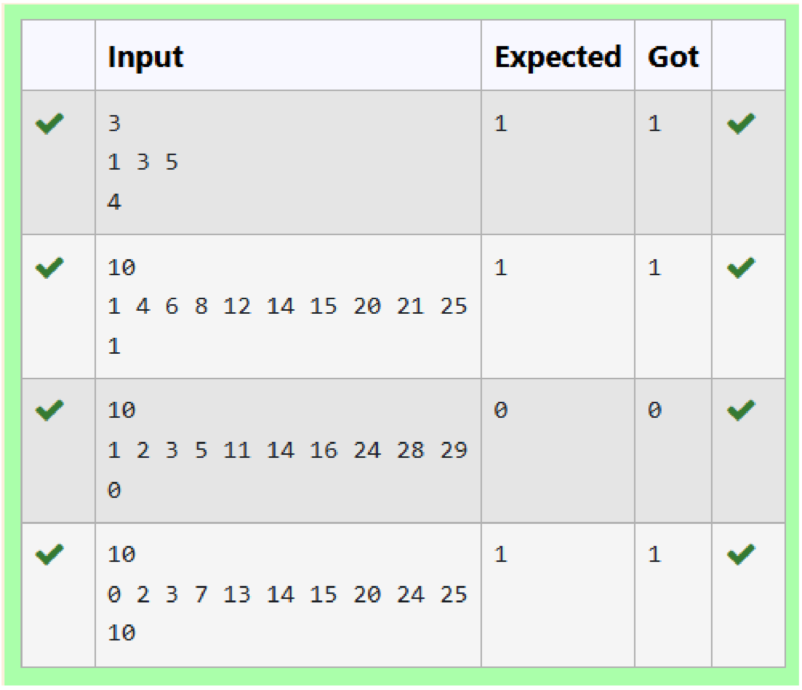
```
for i from 0 to n - 1
{
    read arr[i] from user
}

initialize t  // Target difference
read t from user

initialize flag = 0  // Flag to indicate if a pair is found

initialize i = 0  // First index
initialize j = 1  // Second index

// Loop to find pairs with the specified difference
while i < n and j < n
{
    diff = abs(arr[i] - arr[j])  // Calculate the difference

    if i != j and diff == t
    {
        flag = 1  // Pair found
        break
    }
    else if diff < t
    {
        j++  // Increment second index
    }
    else
    {
        i++  // Increment first index
        }
    }

    // Output the result based on the flag
    if flag
    {
        print 1  // Pair found
    }
    else
    {
        print 0  // No pair found
    }

    return 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	✓