Ex:5 **Date**:15.08.24

Name: Veronica Regina Paul **Register number**:230701377

Competitive Programming

1-Finding Duplicates-O(n^2) Time Complexity,O(1) Space Complexity

AIM:

To find a duplicate element in an array of integers, where the integers range from 1 to n and there is at least one duplicate.

- **Time Complexity**: O(n^2)
- **Space Complexity**: O(1)

ALGORITHM:

1. Input Parsing:
    - Read the number of elements num in the array.
    - Read the num elements of the array into a list.
2. Duplicate Search:
    - Use two nested loops:
        - The outer loop starts from the first element and moves to the second last element.
        - The inner loop compares the current element from the outer loop with every subsequent element.
    - If a match is found (i.e., a duplicate), print the duplicate element.
3. Output:
    - The first duplicate element found is printed.
4. Time Complexity:
    - The outer loop runs n times, and for each iteration, the inner loop runs n-1 times, resulting in a time complexity of O(n^2).
5. Space Complexity:
    - Since the solution only uses a constant amount of extra space (no additional data structures like sets or hashmaps), the space complexity is O(1).

PROBLEM:

Given an array of n integers, each integer is between 1 and n. Find one number that repeats.

Input Format:

- The first line contains the number of elements, n.
- The next n lines contain n integers.

Output Format:

- Print the repeated element.

PROBLEM

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

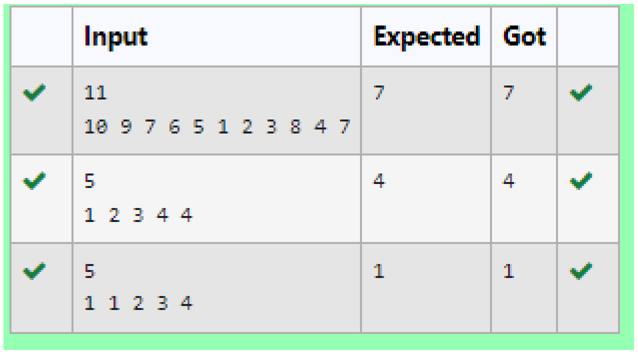
For example:

Input	Result
5 1 1 2 3 4	1

PROGRAM

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

2-Finding Duplicates-O(n) Time Complexity,O(1) Space Complexity

AIM:

To find a duplicate element in an array of integers, where the integers range from 1 to n and there is at least one duplicate, using an efficient approach with:

- Time Complexity: O(n)
- Space Complexity: O(1)

ALGORITHM:

1. Input Parsing:
    - Read the number of elements num in the array.
    - Read the num elements of the array.
2. Duplicate Detection:
    - Use the modulo operation (m % num) to map each element to an index within the array. This ensures that the indices correspond to the value ranges between 1 and n.
    - Check if the element at the mapped index (arr[ind]) is equal to the current element m. If true, that means we have found a duplicate, so print m and break the loop.
    - Otherwise, mark the index (arr[ind]) by storing the element in it.
3. Output:
    - The first duplicate element is printed when found.
4. Time Complexity:
    - The array is traversed only once, so the time complexity is O(n).
5. Space Complexity:
    - The solution doesn't use any extra space for storing values, apart from the input array, so the space complexity is O(1).

PROBLEM

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

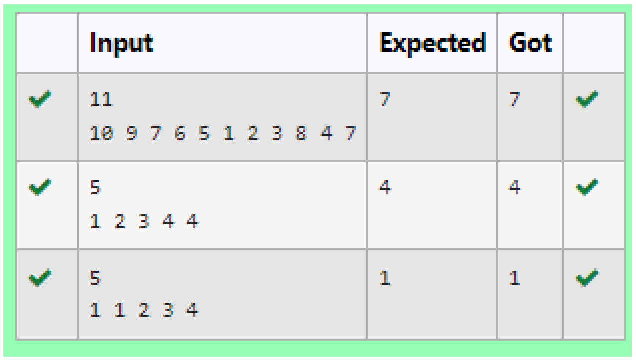
For example:

Input	Result
5 1 1 2 3 4	1

PROGRAM

```
#include<stdio.h>

int main(){

   int num,ind,m;

   scanf("%d",&num);

   int arr[num];

   for(int j=0;j<num;j++){

      scanf("%d",&m);

      ind=m%num;

      if (arr[ind]==m)

      {

         printf("%d",m);

         break;

      }

      else

      arr[ind]=m;

   }


}
```

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	✓

3-Print Intersection of 2 sorted arrays-O(m*n)Time Complexity,O(1) Space Complexity

AIM:

To find the intersection of two sorted arrays, i.e., the common elements that exist in both arrays. The task is to output all the elements that appear in both arrays for each test case.

ALGORITHM:

1. Input Parsing:
   - Read the number of test cases, T.
   - For each test case:
     - Read the size of the first array N1 and the elements of the first array.
     - Read the size of the second array N2 and the elements of the second array.
2. Intersection Logic:
   - Initialize two pointers, i for the first array and j for the second array, both starting from 0.
   - Compare the current elements of both arrays:
     - If the elements are equal, print the element and move both pointers forward.
     - If the element in the first array is smaller, move the pointer i forward.
     - If the element in the second array is smaller, move the pointer j forward.
   - Continue this process until one of the arrays is completely traversed.
3. Edge Cases:
   - If no common elements exist, no output will be printed for that test case.
   - Handle multiple test cases and ensure the outputs are printed correctly.
4. Time Complexity:
   - For each test case, the complexity is O(N1 + N2), where N1 and N2 are the lengths of the two arrays. This is optimal since we are iterating through both arrays once.

PROBLEM

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

For example:

Input	**Result**

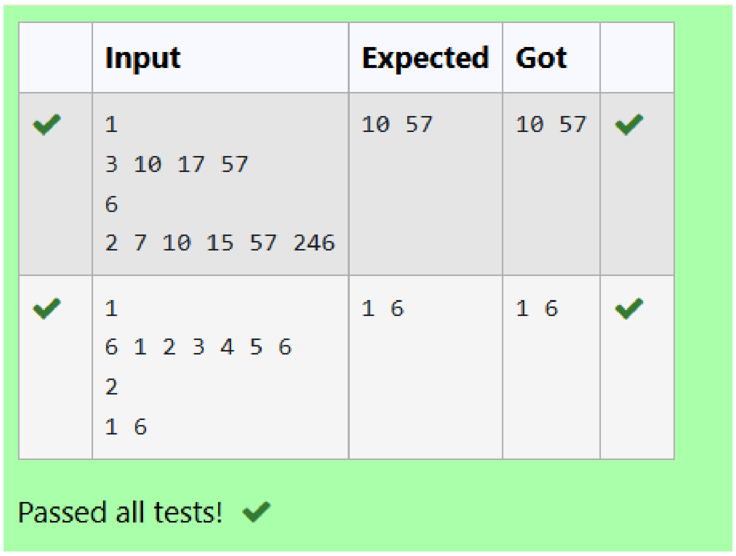
1
 	
3 10 17 57
 	
6	

Input	Result
2 7 10 15 57 246	

PROGRAM

```
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int first = 1;

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

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);
    }

    return 0;
}
```

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	✓

Passed all tests! ✓

4-Print Intersection of 2 sorted arrays-O(m+n)Time Complexity,O(1) Space Complexity

AIM:

To find the intersection of two sorted arrays, that is, the elements which are present in both arrays.

- **Time Complexity**: O(n1 + n2) where n1 and n2 are the sizes of the two arrays.
- **Space Complexity**: O(1) since we're using the input arrays and performing in-place operations.

ALGORITHM:

1. **Input Parsing**:
    - Read the number of test cases T.
    - For each test case:
        - Read the first array arr1 of size n1.
        - Read the second array arr2 of size n2.
2. **Find Intersection**:
    - Initialize two pointers i and j to traverse arr1 and arr2 respectively.
    - Use a first flag to handle spacing in the output.
    - Traverse both arrays simultaneously:

        - If arr1[i] == arr2[j], print the common element and move both pointers forward.
        - If arr1[i] < arr2[j], move the pointer i to the right (i.e., increment i).
        - If arr1[i] > arr2[j], move the pointer j to the right (i.e., increment j).
    - Print the common elements in a single line, separated by spaces.

3. **Output**:
    - For each test case, print the intersection elements in a single line.

PROBLEM

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

For example:

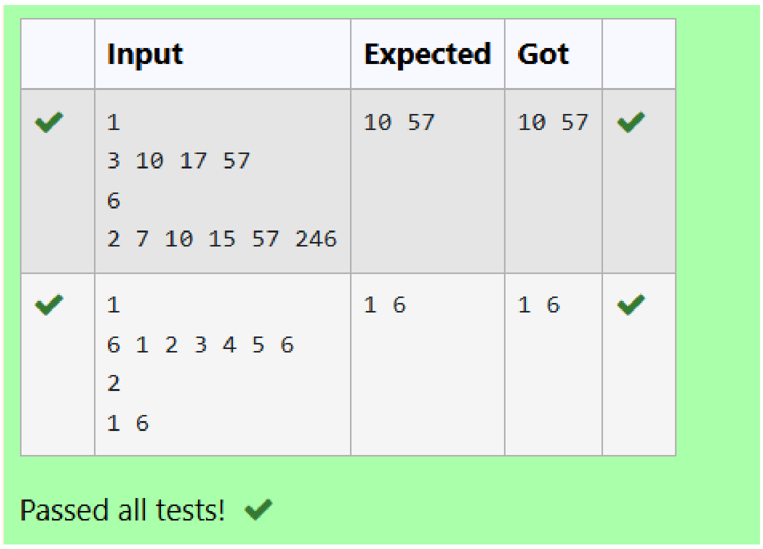
Input	**Result**

1
 	
3 10 17 57
 	
6
 	
2 7 10 15 57 246
	10 57

PROGRAM

```
#include <stdio.h>

void findIntersection(int arr1[], int n1, int arr2[], int n2) {
    int i = 0, j = 0;
    int first = 1;

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

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);
    }

    return 0;
}
```

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	✓

Passed all tests! ✓

5-Pair with Difference-O(n^2)Time Complexity,O(1) Space Complexity

AIM:

To determine if there exist two distinct indices i and j in the sorted array A such that A[j] - A[i] = k, where k is a non-negative integer. The goal is to return 1 if such a pair exists and 0 otherwise.

- **Time Complexity**: O(n^2) (due to the nested loop approach).
- **Space Complexity**: O(1) (since we use only a few variables for the check).

ALGORITHM:

1. Input Parsing:
   - Read the integer n representing the number of elements in the array.
   - Read the n elements of the sorted array A.
   - Read the non-negative integer k.
2. Find Pair with Given Difference:
   - Loop through the array with two pointers:
     - The outer loop (i) runs from 0 to n-1 and represents the first element.

        - The inner loop (j) runs from i+1 to n and represents the second element.
    - For each pair (i, j), calculate A[j] - A[i].
    - If the difference equals k, return 1 (pair found).
    - If the difference exceeds k, break the inner loop early (because the array is sorted and further differences will only increase).
    - If no pair is found after checking all pairs, return 0.

3. Output:
    - Print 1 if a pair with the given difference k is found.
    - Print 0 otherwise.

PROBLEM

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

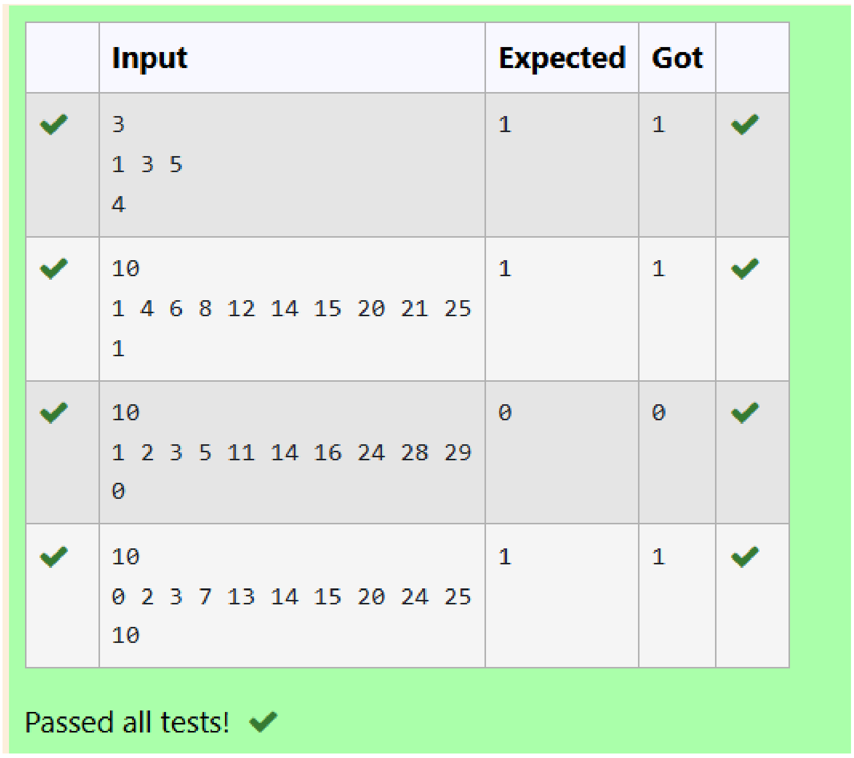
So Return 1.

PROGRAM

```
#include <stdio.h>


int findPairWithDifference(int arr[], int n, int k) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
      if (arr[j] - arr[i] == k) {
        return 1;
```

```
            } else if (arr[j] - arr[i] > k) {
                break;
            }
        }
    }
    return 0;
}

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);
    printf("%d\n", findPairWithDifference(arr, n, k));
    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	✓

Passed all tests! ✓

6-Pair with Difference -O(n) Time Complexity,O(1) Space Complexity

AIM:

To determine if there exists two distinct indices i and j in a sorted array A such that A[j] - A[i] = k, where k is a non-negative integer. If such a pair exists, the program should return 1; otherwise, return 0.

ALGORITHM:

1. Input Parsing:
   - Read the integer n, which denotes the number of elements in the array A.
   - Read the next n elements to populate the array A.
   - Read the non-negative integer k.
2. Find Pair with Given Difference:
   - Use two pointers approach:
     - i represents the first pointer and starts at index 0.
     - j represents the second pointer and starts at index 1.
   - If arr[j] - arr[i] == k, return 1 (pair found).
   - If arr[j] - arr[i] < k, move pointer j forward (increase j).
   - If arr[j] - arr[i] > k, move pointer i forward (increase i).
     - Ensure that i != j, if i == j, move j forward.
   - Repeat the process until you find a valid pair or exhaust the array.
3. Output:
   - If a pair with the difference k is found, print 1.
   - If no such pair exists, print 0.
   -

PROBLEM

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:

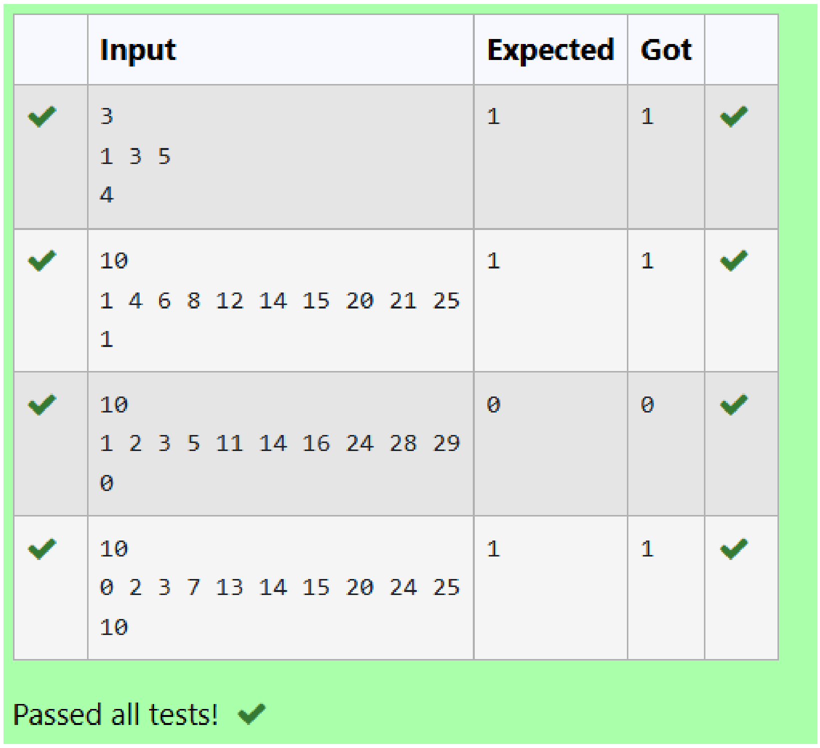
Input	Result

3
 	
1 3 5
 	
4
	1

PROGRAM

```
#include <stdio.h>


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


    while (j < n) {
        if (arr[j] - arr[i] == k) {
            return 1;
        } else if (arr[j] - arr[i] < k) {
            j++;
        } else {
            i++;
            if (i == j) {
                j++;
            }
        }
    }
    return 0;
}

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);
    printf("%d\n", findPairWithDifference(arr, n, k));
    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	✓

Passed all tests! ✓