**Ex. No: 1**  **Date: 12.08.24**

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**Basic C Programming**

## **1.a.**

**Aim**: Given two numbers, write a C program to swap the given numbers.

**Algorithm:**

DECLARE a, b, temp as INTEGER

READ a

READ b

// Swap values of a and b

temp = a

a = b

b = temp

PRINT a, b

**Program:**

#include<stdio.h>

int main(){

int a;

int b;

int temp;

scanf("%d",&a);

scanf("%d",&b);

temp=a;

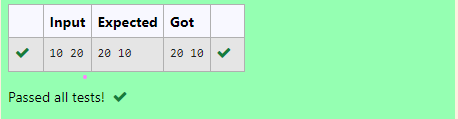
a=b;

b=temp;

printf("%d %d",a,b);

}

**Output:**



**Ex. No: 2**  **Date: 20.08.24**

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**Finding Time Complexity of Algorithms**

## **2.a.** **Finding Complexity using Counter Method**

**Aim**: Convert the following algorithm into a program and find its time complexity using the counter method.  
void function (int n)  
{  
 int i= 1; int s =1;

while(s <= n)  
 {  
 i++;  
 s += i;  
 }   
}   
**Note:** No need of counter increment for declarations and scanf() and count variable printf() statements.  
  
**Input:**  
 A positive Integer n  
**Output:**  
Print the value of the counter variable

**Algorithm:**

void function(int n){

set count = 0

set i = 1

increment count by 1

set s = 1

increment count by 1

while (s <=n){

increment count by 1

increment i by 1

increment count by 1

set s = s + i

increment count by 1

}

increment count by 1

print count

}

**Program:**

#include<stdio.h>

void function(int n){

int count=0;

int i=1;

count++;

int s=1;

count++;

while(s<=n){

count++;

i++;

count++;

s+=i;

count++;

}

count++;

printf("%d",count);

}

int main(){

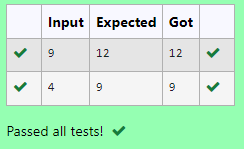
int n;

scanf("%d",&n);

function(n);

}

**Output:**



## **2.b.** **Finding Complexity using Counter Method**

**Aim**: Convert the following algorithm into a program and find its time complexity using the counter method.  
void func(int n)  
{  
 if(n==1)  
 {  
 printf("\*");  
 }  
 else  
 {  
 for(int i=1; i<=n; i++)  
 {  
 for(int j=1; j<=n; j++)  
 {  
 printf("\*");  
 printf("\*");  
 break;  
 }  
 }  
 }   
 }  
  
**Note:** No need of counter increment for declarations and scanf() and count variable printf() statements.  
**Input:**  
 A positive Integer n  
**Output:**  
Print the value of the counter variable

**Algorithm:**

void func(int n){

initialize count to 0

if n = 1{

increment count by 1

print "\*"

}

else{

increment count by 1

// outer loop from 1 to n

for each i from 1 to n{

increment count by 1

// inner loop from 1 to n

for each j from 1 to n {

increment count by 1

// simulate print statements with count increments

increment count by 1 // first simulated printf("\*")

increment count by 1 // second simulated printf("\*")

// exit inner loop immediately

increment count by 1 // break statement

}

increment count by 1

}

increment count by 1

}

print count

}

**Program:**

#include<stdio.h>

void func(int n)

{ int count=0;

if(n==1)

{ count++;

printf("\*");

}

else

{count++;

for(int i=1; i<=n; i++)

{ count++;

for(int j=1; j<=n; j++)

{ count++;

//printf("\*");

count++;

//printf("\*");

count++;

break;

}

count++;

}

count++;

}

printf("%d",count);

}

int main(){

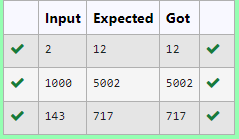
int n;

scanf("%d",&n);

func(n);

}

**Output:**



## **2.c.** **Finding Complexity using Counter Method**

**Aim**: Convert the following algorithm into a program and find its time complexity using counter method.  
 Factor(num) {  
 {  
 for (i = 1; i <= num;++i)  
 {  
 if (num % i== 0)  
 {  
 printf("%d ", i);  
 }   
 }   
 }  
  
  
**Note:** No need of counter increment for declarations and scanf() and counter variable printf() statement.  
  
**Input:**  
 A positive Integer n  
**Output:**  
Print the value of the counter variable

**Algorithm:**

function Factor(num) {

initialize count to 0

// loop from 1 to num

for each i from 1 to num {

increment count by 1

// check if i is a factor of num

if num modulo i equals 0 {

increment count by 1

// simulate printing i (e.g., printf("%d ", i);)

}

increment count by 1 // end of inner if-statement

}

increment count by 1 // after loop completion

print count

}

**Program:**

#include<stdio.h>

void Factor(int num)

{ int count=0;

for (int i = 1; i <= num;++i)

{

count++;

if (num % i== 0)

{

count++;

//printf("%d ", i);

}

count++;

}

count++;

printf("%d",count);

}

int main(){

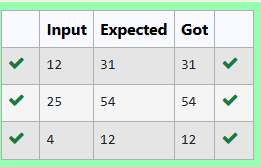
int n;

scanf("%d",&n);

Factor(n);

}

**Output:**



## **2.d.** **Finding Complexity using Counter Method**

**Aim**: Convert the following algorithm into a program and find its timecomplexity using counter method.  
   
void function(int n)  
{  
 int c= 0;  
 for(int i=n/2; i<n; i++)  
 for(int j=1; j<n; j = 2 \* j)  
 for(int k=1; k<n; k = k \* 2)  
 c++;  
}  
   
**Note:** No need of counter increment for declarations and scanf() and count variable printf() statements.  
  
**Input:**  
 A positive Integer n  
**Output:**  
Print the value of the counter variable

**Algorithm:**

function(n) {

initialize count to 0

initialize c to 0

increment count by 1

// outer loop: i goes from n/2 to n-1

for each i from n/2 to n-1 {

increment count by 1

// middle loop: j starts at 1 and doubles each iteration until j < n

for each j starting from 1 and doubling each time (j = 2 \* j) until j < n {

increment count by 1

// inner loop: k starts at 1 and doubles each iteration until k < n

for each k starting from 1 and doubling each time (k = k \* 2) until k < n {

increment count by 1

increment c by 1

increment count by 1

}

increment count by 1 // after inner loop ends

}

increment count by 1 // after middle loop ends

}

increment count by 1 // after outer loop ends

print count

}

**Program:**

#include<stdio.h>

void function(int n)

{

int count=0;

int c= 0;

count++;

for(int i=n/2; i<n; i++){

count++;

for(int j=1; j<n; j = 2 \* j){

count++;

for(int k=1; k<n; k = k \* 2){

count++;

c++;

count++;

}

count++;

}

count++;

}

count++;

printf("%d",count);

}

int main(){

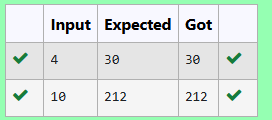
int n;

scanf("%d",&n);

function(n);

}

**Output:**



## **2.e.** **Finding Complexity using Counter Method**

**Aim**: Convert the following algorithm into a program and find its time complexity using counter method.

void reverse(int n)  
{  
 int rev = 0, remainder;  
 while (n != 0)   
 {  
 remainder = n % 10;  
 rev = rev \* 10 + remainder;  
 n/= 10;  
   
 }  
print(rev);  
}  
   
**Note:** No need of counter increment for declarations and scanf() and count variable printf() statements.  
  
**Input:**  
 A positive Integer n  
**Output:**  
Print the value of the counter variable

**Algorithm:**

function reverse(n) {

initialize count to 0

initialize rev to 0

initialize remainder

increment count by 1 // for initialization

// loop until n is not equal to 0

while n is not equal to 0 {

increment count by 1 // start of loop

remainder = n modulo 10

increment count by 1 // after calculating remainder

rev = rev \* 10 + remainder

increment count by 1 // after updating rev

n = n divided by 10

increment count by 1 // after updating n

}

increment count by 1 // after loop ends

// simulate printing rev (e.g., print(rev))

increment count by 1 // for print statement

print count

}

**Program:**

#include<stdio.h>

void reverse(int n)

{

int count=0;

int rev = 0, remainder;

count++;

while (n != 0)

{

count++;

remainder = n % 10;

count++;

rev = rev \* 10 + remainder;

count++;

n/= 10;

count++;

}

count++;

//print(rev);

count++;

printf("%d",count);

}

int main(){

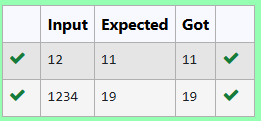
int n;

scanf("%d",&n);

reverse(n);

}

**Output:**



**Ex. No: 3**  **Date: 26.08.24**

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**Greedy Algorithm**

## **3.a.** **1-G-Coin Problem**

**Aim**: Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

64

Output:

4

Explanaton:

We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.

**Algorithm:**

Int main() {

initialize amt

initialize count to 0

read amt from user

// array of currency denominations

initialize arr as {1, 2, 5, 10, 20, 50, 100, 500, 1000}

// loop through currency denominations from highest to lowest

for i from 8 down to 0 {

count = count + (amt divided by arr[i]) // calculate number of notes of current denomination

amt = amt modulo arr[i] // update amt to the remaining amount

}

print count // output the total count of notes

}

**Program:**

#include <stdio.h>

int main()

{

int amt,count=0;

scanf("%d",&amt);

int arr[]={ 1, 2, 5, 10, 20, 50, 100, 500, 1000} ;

for (int i=8;i>=0;i--)

{

count+=amt/arr[i];

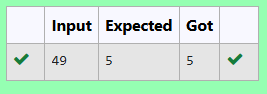
amt%=arr[i];

}

printf("%d",count);

}

**Output:**



## **3.b.** 2**-G-Cookies Problem**

**Aim:**

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Constraints:

1 <= g.length <= 3 \* 10^4

0 <= s.length <= 3 \* 10^4

1 <= g[i], s[j] <= 2^31 - 1

**Algorithm:**

function main() {

initialize n // number of children

read n

initialize greed array of size n // array to hold children's greed factors

// read greed factors for each child

for i from 0 to n-1 {

read greed[i] from user

}

initialize c // number of cookie sizes

read c from user

initialize csize array of size c // array to hold cookie sizes

// read cookie sizes

for j from 0 to c-1 {

read csize[j] from user

}

initialize count to 0 // counter for satisfied children

// check each child's greed against available cookie sizes

for i from 0 to n-1 {

for j from 0 to c-1 {

if csize[j] is greater than or equal to greed[i] {

increment count by 1 // child is satisfied

break // exit inner loop after satisfying this child

}

}

}

print count // output the total count of satisfied children

}

**Program:**

#include<stdio.h>

#include<string.h>

int main(){

int n;

scanf("%d",&n);

int greed[n];

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

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

}

int c;

scanf("%d",&c);

int csize[c];

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

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

}

int count=0;

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

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

if (csize[j]>=greed[i]){

count++;

break;

}

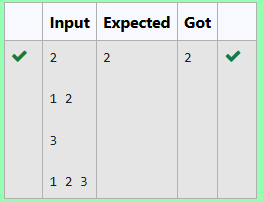
}

}

printf("%d",count);

}

**Output:**



## **3.c. 3-G-Burger Problem**

**Aim:**

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories.   
If he has eaten *i* burgers with c calories each, then he has to run at least *3i \* c* kilometers to burn out the calories. For example, if he ate 3  
 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are (30 \* 1) + (31 \* 3) + (32 \* 2) = 1 + 9 + 18 = 28.  
 But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance  
 he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve the problem.  
**Input Format**  
First Line contains the number of burgers  
Second line contains calories of each burger which is n space-separate integers   
   
**Output Format**  
  
Print: Minimum number of kilometers needed to run to burn out the calories  
   
**Sample Input**  
   
3  
5 10 7  
  
**Sample Output**  
76

**Algorithm:**

int main() {

initialize n // number of elements

read n from user

initialize cal array of size n // array to hold integers

// read values into the cal array

for i from 0 to n-1 {

read cal[i] from user

}

// sorting the array using bubble sort

for i from 0 to n-2 {

for j from 0 to n-i-2 {

if cal[j] is greater than cal[j+1] {

// swap cal[j] and cal[j+1]

initialize temp as cal[j]

cal[j] = cal[j+1]

cal[j+1] = temp

}

}

}

initialize mulfact // variable to hold power value

initialize sum to 0 // variable to hold the final sum

initialize h to n-1 // index for the last element

// compute the weighted sum

for i from 0 to n-1 {

mulfact = n raised to the power of i // compute n^i

sum = sum + (mulfact \* cal[h]) // accumulate the weighted sum

h = h - 1 // move to the next element

}

print sum // output the final result

}

**Program:**

#include<stdio.h>

#include<math.h>

int main(){

int n;

scanf("%d",&n);

int cal[n];

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

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

}

//sorting the array

int i, j, temp;

for (i = 0; i < n-1; i++) {

for (j = 0; j < n-i-1; j++) {

if (cal[j] > cal[j+1]) {

temp = cal[j];

cal[j] = cal[j+1];

cal[j+1] = temp;

}

}

}

int mulfact;

int sum=0;

int h=n-1;

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

{

mulfact=pow(n,i);

sum+=mulfact\*cal[h];

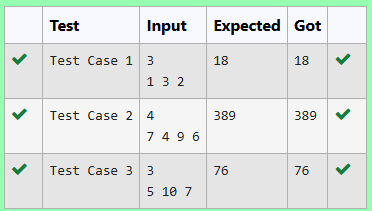
h--;

}

printf("%d",sum);

}

**Output:**



## **3.d. 4-G-Array Sum Max Problem**

**Aim:**

Given an array of N integer, we have to maximize the sum of arr[i] \* i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

Sample Input:

5

2 5 3 4 0

Sample output:

40

**Algorithm:**

function main() {

initialize n // number of elements

read n from user

initialize arr array of size n // array to hold integers

// read values into the arr array

for i from 0 to n-1 {

read arr[i] from user

}

// sorting the array using bubble sort

for i from 0 to n-2 {

for j from 0 to n-i-2 {

if arr[j] is greater than arr[j+1] {

// swap arr[j] and arr[j+1]

initialize temp as arr[j]

arr[j] = arr[j+1]

arr[j+1] = temp

}

}

}

initialize prod to 0 // variable to hold the weighted sum

// compute the weighted sum

for i from 0 to n-1 {

prod = prod + (arr[i] \* i) // accumulate the weighted sum

}

print prod // output the final result

}

**Program:**

#include<stdio.h>

int main(){

int n;

scanf("%d",&n);

int arr[n];

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

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

}

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 prod=0;

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

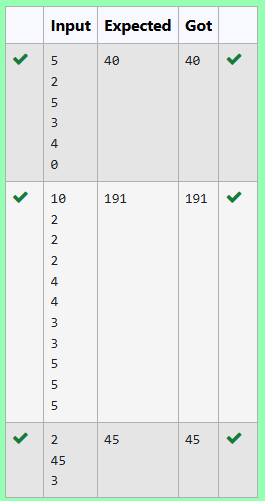
prod+=(arr[i]\*i);

}

printf("%d",prod);

}

**Output:**



## **3.e. 5-G-Product of Array Elements-Minimum**

**Aim:**

Given two arrays array\_One[] and array\_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs( 1 element from each) is minimum. That is SUM (A[i] \* B[i]) for all i is minimum.

**Algorithm:**

function main() {

initialize n // number of elements

read n from user

initialize array\_One of size n // first array

initialize array\_Two of size n // second array

// read values into array\_One

for i from 0 to n-1 {

read array\_One[i] from user

}

// read values into array\_Two

for i from 0 to n-1 {

read array\_Two[i] from user

}

// sorting both arrays

for i from 0 to n-2 {

for j from 0 to n-i-2 {

// sort array\_One in ascending order

if array\_One[j+1] is less than array\_One[j] {

// swap array\_One[j] and array\_One[j+1]

initialize temp as array\_One[j]

array\_One[j] = array\_One[j+1]

array\_One[j+1] = temp

}

// sort array\_Two in descending order

if array\_Two[j+1] is greater than array\_Two[j] {

// swap array\_Two[j] and array\_Two[j+1]

initialize temp as array\_Two[j]

array\_Two[j] = array\_Two[j+1]

array\_Two[j+1] = temp

}

}

}

initialize sum to 0 // variable to hold the final sum

// calculate the sum of products of corresponding elements

for i from 0 to n-1 {

sum = sum + (array\_One[i] \* array\_Two[i]) // accumulate the product

}

print sum // output the final result

}

**Program:**

#include<stdio.h>

int main(){

int n;

scanf("%d",&n);

int array\_One[n];

int array\_Two[n];

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

scanf("%d ",&array\_One[i]);

}

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

scanf("%d ",&array\_Two[i]);

}

for(int i=0;i<n-1;i++){

for(int j=0;j<n-i-1;j++){

if(array\_One[j+1]<array\_One[j]){

int temp=array\_One[j];

array\_One[j]=array\_One[j+1];

array\_One[j+1]=temp;

}

if(array\_Two[j+1]>array\_Two[j]){

int temp=array\_Two[j];

array\_Two[j]=array\_Two[j+1];

array\_Two[j+1]=temp;

}

}

}

int sum=0;

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

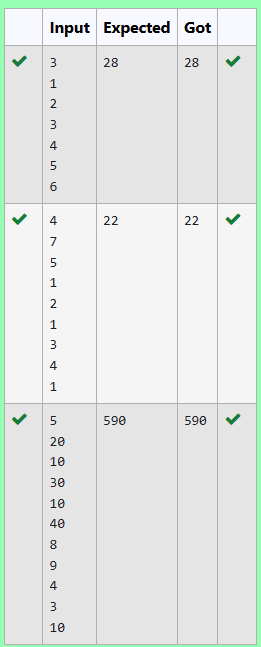
sum+=(array\_One[i]\*array\_Two[i]);

}

printf("%d",sum);

}

**Output:**



**Ex. No: 4**  **Date: 03.09.24**

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**Divide and Conquer**

## **4.a.** **Number of Zeros in a Given Array**

**Aim:** 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.

**Algorithm:**

function count(a, left, right) {

// base case: if left index exceeds right index

if left is greater than right {

return 0

}

initialize mid as (left + right) / 2 // find the middle index

// check if the middle element is 1

if a[mid] is equal to 1 {

// check if the next element is 0

if a[mid + 1] is equal to 0 {

// count zeros from mid + 1 to right

initialize c as (right - (mid + 1)) + 1

return c

} else {

// search in the right half

return count(a, mid + 1, right)

}

}

// check if both ends are 0

else if a[left] is equal to 0 and a[right] is equal to 0 {

return right + 1 // return total count of elements

}

// search in the left half

else {

return count(a, left, mid - 1)

}

}

function main() {

initialize n // number of elements

read n from user

initialize arr array of size n // array to hold binary values

// read values into the arr array

for i from 0 to n - 1 {

read arr[i] from user

}

initialize left as 0 // left index

initialize right as n - 1 // ri

**Program:**

#include <stdio.h>

int count(int a[],int left,int right)

{

if(left>right)

{

return 0;

}

int mid=(left+right)/2;

if(a[mid]==1)

{

if(a[mid+1]==0)

{

int c= (right-(mid+1))+1;

return c;

}

else{

return count(a,mid+1,right);

}

}

else if(a[left]==0 && a[right]==0)

{

return right+1;

}

else

{

return count(a,left,mid-1);

}

}

int main()

{

int n;

scanf("%d",&n);

int arr[n];

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

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

}

int left=0;

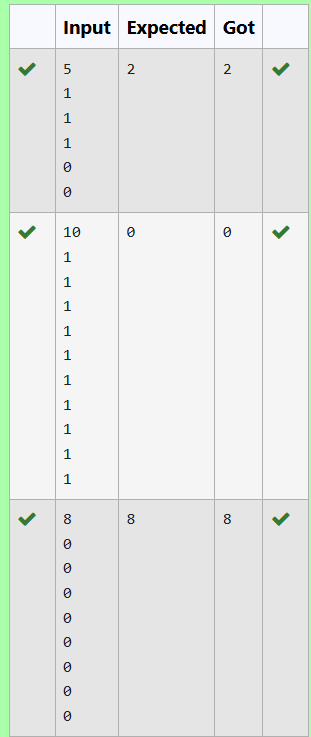
int right=n-1;

int result=count(arr,left,right);

printf("%d",result);

}

**Output:**



## **4.b.** **Majority Element**

**Aim:** 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 \* 104
* -231 <= nums[i] <= 231 - 1

**Algorithm:**

int divide(a, l, r, n) {

// base case: if left index equals right index

if l is equal to r {

return a[l] // return the only element

}

initialize mid as (l + r) / 2 // find the middle index

// recursively divide the array

initialize min as divide(a, l, mid, n) // find min in left half

initialize max as divide(a, mid + 1, r, n) // find max in right half

initialize leftc as 0 // counter for min occurrences

initialize rightc as 0 // counter for max occurrences

// count occurrences of min and max in the entire array

for i from 0 to n - 1 {

if a[i] is equal to min {

increment leftc by 1 // count occurrences of min

} else {

increment rightc by 1 // count occurrences of max

}

}

// check if min occurs more than n/2 times

if leftc is greater than (n / 2) {

return min // return min if it is the majority element

} else {

return max // return max otherwise

}

}

int main() {

initialize n // number of elements

read n from user

initialize a array of size n // array to hold input values

// read values into the array

for j from 0 to n - 1 {

read a[j] from user

}

initialize l as 0 // left index

initialize r as n - 1 // right index

// call the divide function

initialize result as divide(a, l, r, n)

print result // output the final majority element

}

**Program:**

#include<stdio.h>

int divide(int a[],int l,int r,int n){

if(l==r)

{

return a[l];

}

int mid=(l+r)/2;

int min=divide(a,l,mid,n);

int max=divide(a,mid+1,r,n);

int leftc=0,rightc=0;

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

{

if(a[i]==min)

{

leftc++;

}

else

{

rightc++;

}

}

if(leftc>(n/2))

{

return min;

}

else

{

return max;

}

}

int main(){

int n;

scanf("%d",&n);

int a[n];

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

scanf("%d",&a[j]);

}

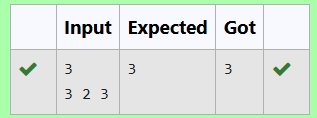
int l=0,r=n-1;

int result=divide(a,l,r,n);

printf("%d",result);

}

**Output:**



## **4.c.** **Finding Floor Value**

**Aim:** 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

**Algorithm:**

int large(arr, l, r, x){

// Base case: if the range is invalid

if r < l

return 0 // return 0 when there is no valid element

// Calculate the middle index

mid = (l + r) / 2

// Check if the middle element is equal to x

if arr[mid] is equal to x

return mid // return the index of x if found

// If the middle element is less than x

else if arr[mid] < x

// Recursively search in the right half

floorIndex = large(arr, mid + 1, r, x)

// Check if a valid floor index is found

if floorIndex is not equal to 0

return floorIndex // return the found index

else

return mid // return mid as the largest element less than x

// If the middle element is greater than x, search in the left half

else

return large(arr, l, mid - 1, x) // search in the left half

}

Int main()

initialize n // number of elements in the array

read n from user

initialize arr of size n // array to hold input values

// Read values into the array

for i from 0 to n - 1

read arr[i] from user

initialize l as 0 // left index

initialize r as n - 1 // right index

initialize x // the value for which we want to find the largest element less than or equal to x

read x from user

// Call the large function

result = large(arr, l, r, x)

// Check the result

if result is equal to 0

print x // if no valid element, print x

else

print arr[result] // print the largest element less than or equal to x

**Program:**

#include<stdio.h>

int large(int arr[],int l,int r,int x){

if (r < l) {

return 0;

}

int mid=(l+r)/2;

if (arr[mid]==x)

{

return mid;

}

else if (arr[mid]<x)

{

int floorIndex=large(arr,mid+1,r,x);

if(floorIndex!=0)

{

return floorIndex;

}

else

{

return floorIndex=mid;

}

}

else

{

return large(arr,l,mid-1,x);

}

}

int main(){

int n;

scanf("%d",&n);

int arr[n];

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

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

}

int l=0;

int r=n-1;

int x;

scanf("%d",&x);

int result=large(arr,l,r,x);

if (result == 0)

{

printf( "%d",x);

}

else

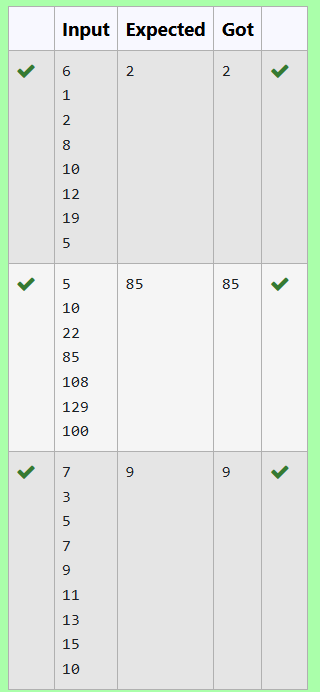
{

printf( "%d",arr[result]);

}

}

**Output:**



## **4.d.** **Two Elements Sum to X**

**Aim:** 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”)

**Algorithm:**

int findPairWithSum(arr, left, right, x){

// Base case: if there are no more pairs to check

if left >= right

print "No" // No pair found

return

// Calculate the sum of the elements at the left and right indices

sum = arr[left] + arr[right]

// Check if the sum is equal to x

if sum is equal to x

print arr[left] // Print the first element of the pair

print arr[right] // Print the second element of the pair

return

// If the sum is less than x, move the left index up

if sum < x

findPairWithSum(arr, left + 1, right, x) // Recursive call with increased left index

else

findPairWithSum(arr, left, right - 1, x) // Recursive call with decreased right index

}

function main()

initialize n // number of elements in the array

read n from user

initialize arr of size n // array to hold input values

// Read values into the array

for i from 0 to n - 1

read arr[i] from user

initialize x // the target sum value

read x from user

// Call the findPairWithSum function

findPairWithSum(arr, 0, n - 1, x)

**Program:**

#include <stdio.h>

void findPairWithSum(int arr[], int left, int right, int x) {

if (left >= right) {

//No pair found

printf("No\n");

return;

}

int sum = arr[left] + arr[right];

if (sum == x){

// If the pair is found

printf("%d\n%d\n", arr[left], arr[right]);

return;

}

if (sum < x){

findPairWithSum(arr, left + 1, right, x);

}

else{

findPairWithSum(arr, left, right - 1, x);

}

}

int main() {

int n;

scanf("%d", &n);

int arr[n];

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

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

}

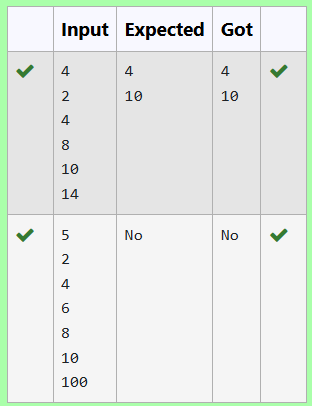
int x;

scanf("%d", &x);

findPairWithSum(arr, 0, n - 1, x);

}

**Output:**



## **4.e.** **Implementation of Quick Sort**

**Aim:** 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

**Algorithm:**

int partition(a, left, right)

{

pivot = right // Choose the last element as pivot

i = left - 1 // Index of smaller element

for j from left to right - 1

{

if a[j] < a[pivot]

{

i++

// Swap a[i] and a[j]

temp = a[i]

a[i] = a[j]

a[j] = temp

}

}

// Swap a[i + 1] and a[right]

temp = a[i + 1]

a[i + 1] = a[right]

a[right] = temp

return (i + 1) // Return the partition index

}

function quick(a, left, right)

{

if left < right

{

p = partition(a, left, right) // Partition the array

quick(a, left, p - 1) // Recursively sort the left sub-array

quick(a, p + 1, right) // Recursively sort the right sub-array

}

}

int main()

{

initialize n // number of elements

read n from user

initialize a of size n // array to hold input values

for i from 0 to n - 1

{

read a[i] from user

}

quick(a, 0, n - 1) // Call the quicksort function

// Print the sorted array

for i from 0 to n - 1

{

print a[i]

}

}

**Program:**

#include <stdio.h>

int partition(int a[], int left, int right) {

int pivot = right;

int i = left-1;

for (int j = left; j < right; j++) {

if (a[j] < a[pivot]) {

i++;

int temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

int temp = a[i + 1];

a[i + 1] = a[right];

a[right] = temp;

return (i + 1);

}

void quick(int a[], int left, int right) {

if (left < right) {

int p = partition(a, left, right);

quick(a, left, p - 1);

quick(a, p + 1, right);

}

}

int main() {

int n;

scanf("%d", &n);

int a[n];

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

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

}

quick(a, 0, n - 1);

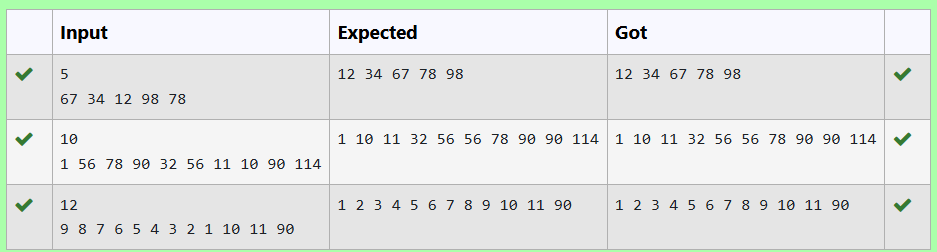
for (int i = 0; i < n; i++) {

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

}

}

**Output:**



**Ex. No: 5**  **Date: 10.09.24**

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**Dynamic Programming**

**5.a. Playing with Numbers**

**Aim:** Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer ‘n’ and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

*Input: 6*  
*Output:6*  
*Explanation: There are 6 ways to 6 represent number with 1 and 3*  
 *1+1+1+1+1+1*  
 *3+3*  
 *1+1+1+3*  
 *1+1+3+1*  
 *1+3+1+1*  
 *3+1+1+1*  
Input Format  
First Line contains the number n  
   
Output Format

Print: The number of possible ways ‘n’ can be represented using 1 and 3

Sample Input  
   
6

Sample Output

6

**Algorithm:**

function countWays(n)

{

initialize a of size n + 1 // Array to store the number of ways

a[0] = 1 // Base case: 1 way to climb 0 stairs

a[1] = 1 // Base case: 1 way to climb 1 stair

if n >= 2

{

a[2] = 1 // Base case: 1 way to climb 2 stairs

}

if n >= 3

{

a[3] = 2 // Base case: 2 ways to climb 3 stairs

}

// Fill the array for all stairs from 4 to n

for i from 4 to n

{

a[i] = a[i - 1] + a[i - 3] // Total ways to climb i stairs

}

return a[n] // Return the number of ways to climb n stairs

}

function main()

{

initialize n // Number of stairs

read n from user

result = countWays(n) // Calculate the number of ways

print result // Print the result

return 0

}

**Program:**

#include <stdio.h>

long long int countWays(int n) {

long long int a[n + 1];

a[0] = 1;

a[1] = 1;

if (n >= 2) {

a[2] = 1;

}

if (n >= 3) {

a[3] = 2;

}

for (int i = 4; i <= n; i++) {

a[i] = a[i - 1] + a[i - 3];

}

return a[n];

}

int main() {

int n;

scanf("%d", &n);

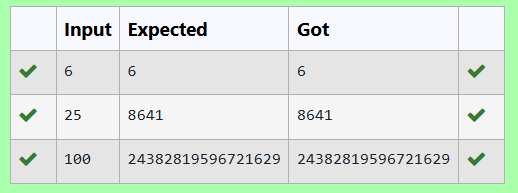
long long int result = countWays(n);

printf("%lld",result);

return 0;

}

**Output:**



## **5.b. Playing with chessboard**

**Aim:** Ram is given with an n\*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by traveling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:  
Input  
3  
1 2 4  
2 3 4  
8 7 1  
Output:  
19

Explanation:  
Totally there will be 6 paths among that the optimal is  
 Optimal path value:1+2+8+7+1=19

Input Format  
First Line contains the integer n  
The next n lines contain the n\*n chessboard values  
   
Output Format

Print Maximum monetary value of the path

**Algorithm:**

function max(a, b)

{

return (a > b) ? a : b // Return the maximum of a and b

}

function maxMonetaryPath(n, board)

{

initialize dp[n][n] // Array to store maximum monetary path sums

dp[0][0] = board[0][0] // Starting point

// Fill the first row

for j from 1 to n - 1

{

dp[0][j] = dp[0][j - 1] + board[0][j]

}

// Fill the first column

for i from 1 to n - 1

{

dp[i][0] = dp[i - 1][0] + board[i][0]

}

// Fill the rest of the dp table

for i from 1 to n - 1

{

for j from 1 to n - 1

{

dp[i][j] = board[i][j] + max(dp[i - 1][j], dp[i][j - 1])

}

}

return dp[n - 1][n - 1] // Return the maximum monetary path to the bottom-right corner

}

function main()

{

initialize n // Size of the board

read n from user

initialize board[n][n] // Create the board array

for i from 0 to n - 1

{

for j from 0 to n - 1

{

read board[i][j] from user

}

}

result = maxMonetaryPath(n, board) // Calculate the maximum monetary path

print result // Print the result

}

**Program:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int maxMonetaryPath(int n, int board[n][n]) {

int dp[n][n];

dp[0][0] = board[0][0];

for (int j = 1; j < n; j++) {

dp[0][j] = dp[0][j - 1] + board[0][j];

}

for (int i = 1; i < n; i++) {

dp[i][0] = dp[i - 1][0] + board[i][0];

}

for (int i = 1; i < n; i++) {

for (int j = 1; j < n; j++) {

dp[i][j] = board[i][j] + max(dp[i - 1][j], dp[i][j - 1]);

}

}

return dp[n - 1][n - 1];

}

int main() {

int n;

scanf("%d", &n);

int board[n][n];

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

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

scanf("%d", &board[i][j]);

}

}

int result = maxMonetaryPath(n, board);

printf("%d\n", result);

}

**Output:**



## **5.c. Longest Common Subsequence**

**Aim:** Given two strings find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb

| s1 |  | a | g | **g** | **t** | **a** | **b** |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s2 |  | **g** | x | **t** | x | **a** | y | **b** |

The length is 4

Solveing it using Dynamic Programming

For example:

| **Input** | **Result** |
| --- | --- |
| aab azb | 2 |

**Algorithm:**

int longestCommonSubsequence(s1, s2)

{

m = length of s1 // Length of first string

n = length of s2 // Length of second string

initialize dp[m + 1][n + 1] // DP table

// Initialize the DP table with base cases

for i from 0 to m

{

for j from 0 to n

{

if i == 0 or j == 0

{

dp[i][j] = 0 // Base case: LCS of an empty string

}

else if s1[i - 1] == s2[j - 1]

{

dp[i][j] = dp[i - 1][j - 1] + 1 // Characters match

}

else

{

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) // Characters do not match

}

}

}

return dp[m][n] // Return length of LCS

}

function main()

{

initialize s1[100], s2[100] // Arrays to hold the strings

read s1 from user

read s2 from user

result = longestCommonSubsequence(s1, s2) // Calculate LCS

print result // Print the result

}

**Program:**

#include <stdio.h>

#include <string.h>

int longestCommonSubsequence(char s1[], char s2[]) {

int m = strlen(s1);

int n = strlen(s2);

int dp[m + 1][n + 1];

// Initialize the DP table with base cases

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

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

if (i == 0 || j == 0) {

dp[i][j] = 0;

}

else if (s1[i - 1] == s2[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else {

dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];

}

}

}

return dp[m][n];

}

int main() {

char s1[100], s2[100];

scanf("%s", s1);

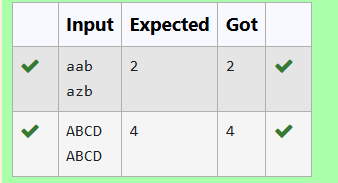
scanf("%s", s2);

int result = longestCommonSubsequence(s1, s2);

printf("%d", result);

}

**Output:**



## **5.d. Longest non-decreasing Subsequence**

**Aim:** Problem statement:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

Eg:

Input:9

Sequence:[-1,3,4,5,2,2,2,2,3]

the subsequence is [-1,2,2,2,2,3]

Output:6

**Algorithm:**

int longestNonDecreasingSubsequence(n, sequence)

{

initialize dp[n] // Array to hold the lengths of subsequences

maxLength = 1 // Initialize the maximum length

// Initialize dp array where each element is 1

for i from 0 to n - 1

{

dp[i] = 1

}

// Calculate the length of the longest non-decreasing subsequence

for i from 1 to n - 1

{

for j from 0 to i - 1

{

if sequence[j] <= sequence[i]

{

dp[i] = max(dp[i], dp[j] + 1) // Update dp[i] if a longer subsequence is found

}

}

maxLength = max(maxLength, dp[i]) // Update the maximum length found

}

return maxLength // Return the length of the longest non-decreasing subsequence

}

function main()

{

initialize n // Number of elements in the sequence

read n from user

initialize sequence[n] // Array to hold the sequence

// Read values into the sequence

for i from 0 to n - 1

{

read sequence[i] from user

}

result = longestNonDecreasingSubsequence(n, sequence) // Calculate result

print result // Print the result

}

**Program:**

#include <stdio.h>

int longestNonDecreasingSubsequence(int n, int sequence[]) {

int dp[n];

int maxLength = 1;

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

dp[i] = 1;

}

for (int i = 1; i < n; i++) {

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

if (sequence[j] <= sequence[i]) {

dp[i] = (dp[i] > dp[j] + 1) ? dp[i] : dp[j] + 1;

}

}

maxLength = (maxLength > dp[i]) ? maxLength : dp[i];

}

return maxLength;

}

int main() {

int n;

scanf("%d", &n);

int sequence[n];

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

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

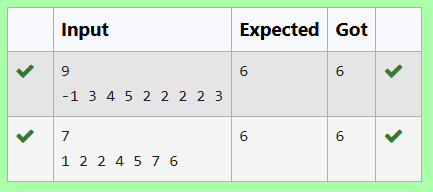
}

int result = longestNonDecreasingSubsequence(n, sequence);

printf("%d", result);

}

**Output:**



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

**Register No.: 230701517**  **Name: Ashwin R**

Shape

**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:**

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

}

flag = 0 // Initialize a flag to indicate if a duplicate is found

// Search for the first duplicate element

for i from 0 to n - 1

{

el1 = arr[i] // Current element

for j from 0 to n - 1

{

// Check for duplicates and ensure indices are different

if el1 == arr[j] and i != j

{

print el1 // Print the duplicate element

flag = 1 // Set flag to indicate a duplicate was found

break // Exit inner loop

}

}

if flag

break // Exit outer loop if a duplicate was found

}

}

**Program:**

#include<stdio.h>

int main(){

int n;

scanf("%d",&n);

int arr[n];

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

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

}

int flag=0;

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

int el1=arr[i];

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

if (el1==arr[j] && i!=j){

printf("%d",el1);

flag=1;

break;

}

}

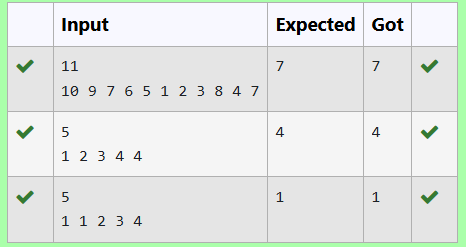
if(flag)

break;

}

}

**Output:**



## **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;

}

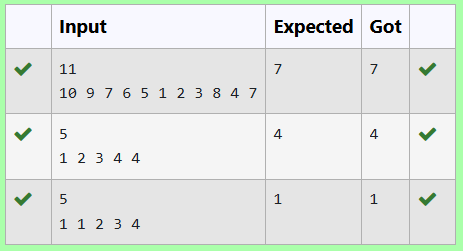
else

b[a[i]]=1;

}

}

**Output:**



## **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:**

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:**

#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]);

}

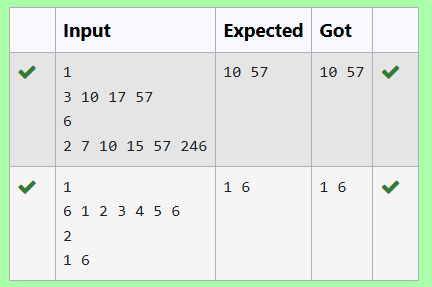
}

}

}

}

**Output:**



## **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:**

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++;

}

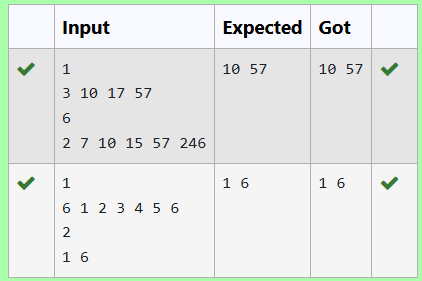
}

printf("\n");

}

}

**Output:**



## **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 != 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:**



## **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:**

