REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

SAMPLE PRACTICE PROGRAM

QUESTION 1.A

AIM :

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

For example:

Input	Result
10 20	20 10

ALGORITHM :

Step 1: Start

Step 2: Input integers x and y

Step 3: Store the value of x in temp

Step 4: Assign the value of y to x

Step 5: Assign the value of temp to y

Step 6: Print x and y

Step 7: Stop

PROGRAM :

```c
#include<stdio.h>
int main ()
{
    int a,b,temp;
    scanf("%d",&a);
    scanf("%d",&b);
    temp=a;
    a=b;
    b=temp;
    printf("%d %d",a,b);

}
```

OUTPUT:

	Input	Expected	Got	
✔	10 20	20 10	20 10	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

AIM:

QUESTION 1.B

Write a C program to find the eligibility of admission for a professional course based on the following criteria:

Marks in Maths >= 65
Marks in Physics >= 55
Marks in Chemistry >= 50
Or
Total in all three subjects >= 180

Sample Test Cases

Test Case 1

Input

70 60 80

Output

The candidate is eligible

Test Case 2

Input

50 80 80

Output

The candidate is eligible

Test Case 3

Input

50 60 40

Output

The candidate is not eligible

ALGORITHM:

Step 1: Start

Step 2: Input marks for Physics (p), Chemistry (c), and Math (m)

Step 3: Check if m >= 65, p >= 55, c >= 50 or if the total marks m + p + c >= 180

Step 4: If true, print "The candidate is eligible"; else, print "The candidate is not eligible" Step 5: Stop

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int m,p,c,t;
    scanf("%d %d %d",&m,&p,&c);
    t=m+p+c;
    if(m>=65 && p>=55 && c>=50){
        printf("The candidate is eligible");
    }
    else if(t>=180) {
        printf("The candidate is eligible");
    }
    else{
        printf("The candidate is not eligible");
    }

}
```

OUTPUT:

	Input	Expected	Got	
✔	70 60 80	The candidate is eligible	The candidate is eligible	✔
✔	50 80 80	The candidate is eligible	The candidate is eligible	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

AIM:

QUESTION 1.C

Malini goes to BestSave hyper market to buy grocery items. BestSave hyper market provides 10% discount on the bill amount B when ever the bill amount B is more than Rs.2000.

The bill amount B is passed as the input to the program. The program must print the final amount A payable by Malini.

Input Format:

The first line denotes the value of B.

Output Format:

The first line contains the value of the final payable amount A.

Example Input/Output 1:

Input:

1900

Output:

1900

Example Input/Output 2:

Input:

3000

Output:

2700

ALGORITHM:

Step 1: Start

Step 2: Input the bill amount b

Step 3: If b > 2000, calculate a discount of 10% and subtract it from b to get the final amount f Step 4: If b <= 2000, set f = b

Step 5: Print f

Step 6: Stop

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int b;
    scanf("%d",&b);
    if(b>2000)
    {
        int p=(0.1*b);
        int pay=(b-p);
        printf("%d",pay);
    }
    else{
        printf("%d",b);
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	1900	1900	1900	✔
✔	3000	2700	2700	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

AIM:

QUESTION 1.D

Baba is very kind to beggars and every day Baba donates half of the amount he has when ever a beggar requests him. The money M left in Baba's hand is passed as the input and the number of beggars B who received the alms are passed as the input. The program must print the money Baba had in the beginning of the day.

Input Format:

The first line denotes the value of M.
The second line denotes the value of B.

Output Format:

The first line denotes the value of money with Baba in the beginning of the day.

Example Input/Output:

Input:

100
2

Output:

400

Explanation:

Baba donated to two beggars. So when he encountered second beggar he had 100*2 = Rs.200 and when he encountered 1st he had 200*2 = Rs.400.

ALGORITHM:

Step 1: Start

Step 2: Input integers m and b

Step 3: While b is not zero, double the value of m and decrement b by 1 Step

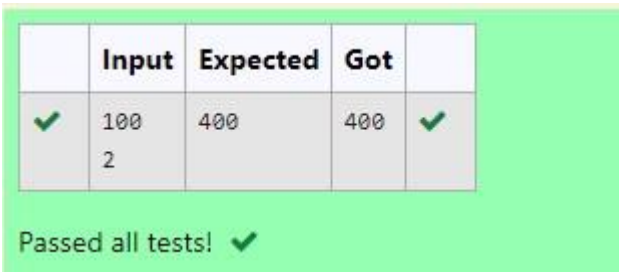
4: Print the value of m

Step 5: Stop

PROGRAM:

```
#include<stdio.h>
int main(){
    int m,b;
    scanf("%d %d",&m,&b);
    int i=0;
    while(i<b){
        m=m*2;
        i++;
    }
    printf("%d",m);
}
```

OUTPUT:



	Input	Expected	Got	
✔	100 2	400	400	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.E AIM:

The CEO of company ABC Inc wanted to encourage the employees coming on time to the office. So he announced that for every consecutive day an employee comes on time in a week (starting from Monday to Saturday), he will be awarded Rs.200 more than the previous day as "Punctuality Incentive". The incentive I for the starting day (ie on Monday) is passed as the input to the program. The number of days N an employee came on time consecutively starting from Monday is also passed as the input. The program must calculate and print the "Punctuality Incentive" P of the employee.

Input Format:

The first line denotes the value of I.
The second line denotes the value of N.

Output Format:

The first line denotes the value of P.

Example Input/Output:

Input:

500
3

Output:

2100

Explanation:

On Monday the employee receives Rs.500, on Tuesday Rs.700, on Wednesday Rs.900

So total = Rs.2100

ALGORITHM:

Step 1: Start

Step 2: Input integers i and d Step 3:

Initialize s with the value of i

Step 4: While d > 1, add 200 to i, add i to s, and decrement d by 1

Step 5: Print the value of s

Step 6: Stop

PROGRAM:

```
#include<stdio.h>
int main(){
    int i,n,a=0,t=0;
    scanf("%d %d",&i,&n);
    while(a<n){
        t=t+i;
        i=i+200;
        a++;
    }
    printf("%d",t);
}
```

OUTPUT:

	Input	Expected	Got	
✔	500 3	2100	2100	✔
✔	100 3	900	900	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.F

AIM:

Two numbers M and N are passed as the input. A number X is also passed as the input. The program must print the numbers divisible by X from N to M (inclusive of M and N).

Input Format:

The first line denotes the value of M
The second line denotes the value of N
The third line denotes the value of X

Output Format:

Numbers divisible by X from N to M, with each number separated by a space.

Boundary Conditions:

1 <= M <= 9999999
M < N <= 9999999
1 <= X <= 9999

Example Input/Output 1:

Input:
2
40
7

Output:
35 28 21 14 7

Example Input/Output 2:

Input:
66
121
11

Output:
121 110 99 88 77 66

ALGORITHM:

Step 1: Start

Step 2: Input integers m, n, and x Step

3: Initialize i with the value of n

Step 4: While i >= m, if i is divisible by x, print i

Step 5: Decrement i by 1

Step 6: Stop

PROGRAM:

```
#include<stdio.h>
int main()
{
    int n,m,x;
    scanf("%d %d %d",&n,&m,&x);
    while(m>=n){
        if (m%x==0){
            printf("%d ", m);
        }
        m--;
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	2 40 7	35 28 21 14 7	35 28 21 14 7	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.G AIM:

Write a C program to find the quotient and reminder of given integers.

For example:

Input	Result
12 3	4 0

ALGORITHM:

Step 1: Start

Step 2: Input integers a and d

Step 3: Calculate the quotient q = a / d and remainder r = a % d

Step 4: Print q and r

Step 5: Stop

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int n,d,q,r;
    scanf("%d %d",&n,&d);
    q=n/d;
    r=n%d;
    printf("%d\n%d",q,r);
}
```

OUTPUT:

	Input	Expected	Got	
✔	12 3	4 0	4 0	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.H

ALGORITHM:

Step 1: Start

Step 2: Input three integers a, b, and c

Step 3: Check which of the three integers is the largest

Step 4: Print the largest integer

Step 5: Stop

Write a C program to find the biggest among the given 3 integers?

For example:

Input	Result
10 20 30	30

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int a,b,c;
    scanf("%d %d %d",&a,&b,&c);
    if(a>b && a>c){
        printf("%d",a);
    }
    else if(b>a && b>c){
        printf("%d",b);
    }
    else{
        printf("%d",c);
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	10 20 30	30	30	✔

Passed all tests! ✔

RESULT:

The above program

is executed

successfully.

QUESTION 1.I

AIM:

Write a C program to find whether the given integer is odd or even?

For example:

Input	Result
12	Even
11	Odd

ALGORITHM:

Step 1: Start

Step 2: Input an integer a

Step 3: Check if a is even or odd

Step 4: Print "Even" if a is even; otherwise, print "Odd"

Step 5: Stop

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    if(n%2==0)
    {
        printf("Even");
    }
    else{
        printf("Odd");
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	12	Even	Even	✔
✔	11	Odd	Odd	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

Question no:1.J

Write a C program to find the factorial of given n.

For example:

Input	Result
5	120

ALGORITHM:

Step 1: Start

Step 2: Input an integer a

Step 3: Set x = a

Step 4: While x > 1, decrement x by 1 and multiply it with a

Step 5: Print the final value of a

Step 6: Stop

QUESTION 1.K

Aim:

PROGRAM:

```
#include <stdio.h>
int main(){
    int n,f=1;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        f=f*i;
    }
    printf("%d",f);
}
```

OUTPUT:

	Input	Expected	Got	
✔	5	120	120	✔

Passed all tests! ✔

RESULT:

The **above** program is executed successfully.

Question no:1.K:

Write a C program to find the sum first N natural numbers.

For example:

Input	Result
3	6

ALGORITHM:

Step 1: Start

Step 2: Input an integer a

Step 3: Initialize b = 0

Step 4: While a != 0, add a to b and decrement a by 1

Step 5: Print the value of b

Step 6: Stop

PROGRAM:

```
#include<stdio.h>
int main()
{
    int n,a=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        a=a+i;
    }
    printf("%d",a);
}
```

:

OUTPUT:

	Input	Expected	Got	
✔	3	6	6	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.L

AIM:

Write a C program to find the Nth term in the fibonacci series.

For example:

Input	Result
0	0
1	1

ALGORITHM:

Step 1: Start

Step 2: Input an integer n

Step 3: Define a recursive function fib(n) that returns fib(n-1) + fib(n-2) for n > 1 and n for n <= 1

Step 4: Print the result of fib(n)

Step 5: Stop

PROGRAM:

```
#include<stdio.h>
int fib(int n)
{
    if(n<=1){
        return n;
    }
    else{
        return fib(n-1)+fib(n-2);
    }
}
int main()
{
    int n;
    scanf("%d",&n);
    printf("%d",fib(n));
    return 0;
}
```

OUTPUT:

	Input	Expected	Got	
✔	0	0	0	✔
✔	1	1	1	✔
✔	4	3	3	✔

RESULT:

The above program is executed successfully.

QUESTION 1.M

AIM:

Write a C program to find the power of integers.

input:

a b

output:

a^b value

ALGORITHM:

Step 1: Start

Step 2: Input integers a and b

Step 3: Initialize i = 0 and p = 1

Step 4: While i < b, multiply p with a and increment i by 1

Step 5: Print the value of p

Step 6: Stop

PROGRAM:

```
#include<stdio.h>
int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    int i=0;
    int p=1;
    while(i<b){
        p=p*a;
        i++;
    }
    printf("%d",p);
}
```

OUTPUT:

	Input	Expected	Got	
✔	2 5	32	32	✔

RESULT:

The above program is executed successfully.

QUESTION 1.N

AIM:

Write a C program to find Whether the given integer is prime or not.

For example:

Input	Result
7	Prime
9	No Prime

ALGORITHM:

Step 1: Start

Step 2: Input an integer n

Step 3: For each number i from 2 to n-1, check if n % i == 0 Step 4: If

divisible, set flag = 1 and break; else, set flag = 0

Step 5: If flag == 0, print "Prime"; else, print "No Prime"

Step 6: Stop

PROGRAM:

```c
#include<stdio.h>
int main()
{
    int n,flag;
    scanf("%d",&n);


    for(int i=2;i<n;i++){
        if(n%i==0){
            flag=1;
            break;
        }
        else{
            flag=0;
        }
    }
    if(flag==0){
        printf("Prime");
    }
    else{
        printf("No Prime");
    }
}
```

OUTPUT:

	Input	Expected	Got	
✔	7	Prime	Prime	✔
✔	9	No Prime	No Prime	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 1.O

AIM:

Write a C program to find the reverse of the given integer?

ALGORITHM:

Step 1: Start

Step 2: Input an integer n

Step 3: Initialize rev = 0

Step 4: While n != 0, calculate the remainder rem = n % 10

Step 5: Update rev = rev * 10 + rem and divide n by 10

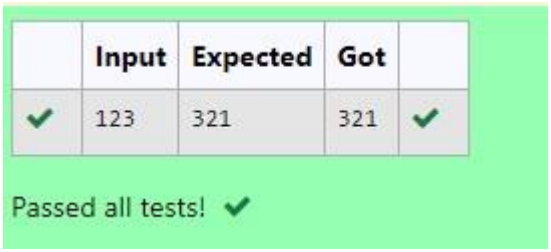
Step 6: Print rev

Step 7: Stop

PROGRAM:

```
#include<stdio.h>
int main()
{
    int n,rem,rev=0;
    scanf("%d",&n);
    while(n!=0)
    {
        rem=n%10;
        rev=rev*10+rem;
        n/=10;
    }
    printf("%d",rev);
}
```

OUTPUT:



	Input	Expected	Got	
✔	123	321	321	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

TIME COMPLEXITY

QUESTION 2.A

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

For example:

Input	Result
9	12

AIM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize c to 0 to count operations

Step 4: Initialize i to 1

Step 5: Increment c by 1

Step 6: Initialize s to 1

Step 7: Increment c by 1

Step 8: While s is less than or equal to n, do Steps 8.1 to 8.5

Step 8.1: Increment c by 1

Step 8.2: Increment i by 1

Step 8.3: Increment c by 1 Step
8.4: Add i to s (s += i)

Step 8.5: Increment c by 1

Step 9: Increment c by 1

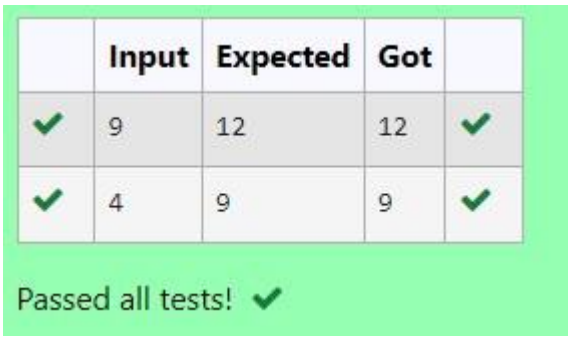
Step 10: Print the value of c

Step 11: Stop

PROGRAM:

```c
#include<stdio.h>
void function(int n)
{
    int c=0;
    int i=1;
    c++;
    int s=1;
    c++;
    while(s<=n)
    {
        c++;
        i++;
        c++;
        s+=i;
        c++;

    }
    c++;
    printf("%d",c);

}
int main()
{
    int n;
    scanf("%d",&n);
    function(n);
    return 0;
}
```

OUTPUT :



	Input	Expected	Got	
✔	9	12	12	✔
✔	4	9	9	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

AIM:

QUESTION 2.b

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:

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize c to 0 to count operations
Step 4: If n is equal to 1, go to Step 5, else go to Step 7
Step 5: Increment c by 1
Step 6: Print "*" and go to Step 12
Step 7: Increment c by 1
Step 8: For each integer i from 1 to n, do Steps 9 to 11
Step 9: Increment c by 1
Step 10: For each integer j from 1 to n, do Steps 10.1 to 10.4
Step 10.1: Increment c by 1
Step 10.2: Increment c by 1
Step 10.3: Increment c by 1
Step 10.4: Break out of the inner loop
Step 11: Increment c by 1
Step 12: Increment c by 1
Step 13: Print the value of c
Step 14: Stop

PROGRAM:

```
#include<stdio.h>
void func(int n)
{   int c=0;
    if(n==1)
    { c++;
      printf("*");
    }
    else
    {
        c++;
        for(int i=1; i<=n; i++)
        {
            c++;
            for(int j=1; j<=n; j++)
            {
                c++;
                //printf("*");
                c++;
               //printf("*");
                c++;
                break;
            }
            c++;
        }
        c++;
    }
    printf("%d",c);
}


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

OUTPUT:

	Input	Expected	Got	
✔	2	12	12	✔
✔	1000	5002	5002	✔
✔	143	717	717	✔

RESULT:

AIM:

The above code is executed successfully and gives expected output.

QUESTION 2.C

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:

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize c to 0 to count operations
Step 4: For each integer i from 1 to n, do Steps 5 to 7
Step 5: Increment c by 1
Step 6: If n is divisible by i (n % i == 0), increment c by 1
Step 7: Increment c by 1
Step 8: Increment c by 1
Step 9: Print the value of c

Step 10: Stop

PROGRAM:

```c
#include <stdio.h>

void Factor(int num) {
    int c = 0;
    for (int i = 1; i <= num; ++i)
    {
        c++;
        if (num % i == 0)
        {
           c++;
        }
        c++;
    }
    c++;
    printf("%d", c);
}

int main() {
    int n;
    scanf("%d", &n);
    Factor(n);
    return 0;
}
```

OUTPUT:

	Input	Expected	Got	
✔	12	31	31	✔
✔	25	54	54	✔
✔	4	12	12	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

AIM:

QUESTION 2.D

Convert the following algorithm into a program and find its time

complexity 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

ALGORITM:

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize count to 0 to count operations
Step 4: Initialize c to 0
Step 5: Increment count by 1
Step 6: For each integer i from n/2 to n - 1, do Steps 7 to 9
Step 7: Increment count by 1
Step 8: Initialize j to 1 and while j is less than n, do Steps 8.1 to 8.5
Step 8.1: Increment count by 1
Step 8.2: Initialize k to 1 and while k is less than n, do Steps 8.2.1 to 8.2.4
Step 8.2.1: Increment count by 1
Step 8.2.2: Increment c by 1
Step 8.2.3: Increment count by 1
Step 8.2.4: Multiply k by 2 (k = k * 2)
Step 8.3: Increment count by 1
Step 8.4: Multiply j by 2 (j = j * 2)
Step 9: Increment count by 1
Step 10: Increment count by 1
Step 11: Print the value of count
Step 12: Stop

PROGRAM:

```c
#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:

	Input	Expected	Got	
✔	4	30	30	✔
✔	10	212	212	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

AIM:

QUESTION 2.E

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:

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize counter to 0 to count operations
Step 4: Initialize rev to 0 and remainder as unassigned
Step 5: Increment counter by 1
Step 6: While n is not equal to 0, do Steps 6.1 to 6.7
Step 6.1: Increment counter by 1
Step 6.2: Calculate remainder as n % 10
Step 6.3: Increment counter by 1
Step 6.4: Update rev to rev * 10 + remainder
Step 6.5: Increment counter by 1
Step 6.6: Divide n by 10 (n /= 10)
Step 6.7: Increment counter by 1
Step 7: Increment counter by 1
Step 8: Increment counter by 1
Step 9: Print the value of counter
Step 10: Stop

PROGRAM:

```c
#include <stdio.h>
void reverse(int n)
{
    int counter=0;
    int rev = 0, remainder;
    counter++;
    while (n != 0)
     { counter++;
         remainder = n % 10;
         counter++;
         rev = rev * 10 + remainder;
         counter++;
         n/= 10;
         counter++;

     }counter++;
  counter++;
//print(rev);
printf("%d",counter);
}
int main(){
     int n;
     scanf("%d",&n);
     reverse(n);
}
```

OUTPUT:

	Input	Expected	Got	
✔	12	11	11	✔
✔	1234	19	19	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

GREEDY ALGORITHM

QUESTION 3.A 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:

Step 1: Start

Step 2: Input the integer v, the amount for which denominations are needed.

Step 3: Initialize an array denominations with values {1000, 500, 100, 50, 20, 10, 5, 2, 1}.

Step 4: Initialize count to 0 to keep track of the total number of denominations. **Step 5:** For each denomination in denominations:

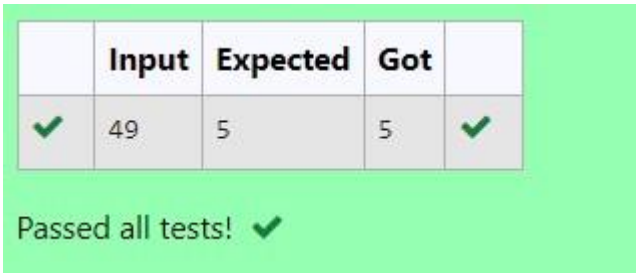
- Divide v by the current denomination to find how many of that denomination are needed and add the result to count.
- Update v to the remainder after division.

Step 6: Print the value of count. **Step 7: Stop**

PROGRAM :

```c
#include <stdio.h>
int main() {
    int v;
    scanf("%d", &v);
    int denominations[] = {1000, 500, 100, 50, 20, 10, 5, 2, 1};
    int count = 0;
    for (int i = 0; i < sizeof(denominations) / sizeof(denominations[0]); i++) {
        count += v / denominations[i];
        v %= denominations[i];
    }
    printf("%d\n", count);
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✔	49	5	5	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

QUESTION 3.B 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:

Step 1: Start

Step 2: Input the integer n, the number of elements in array g. **Step**

3: Input n integers into array g.

Step 4: Input the integer m, the number of elements in array c.

Step 5: Input m integers into array c.

Step 6: Initialize co to 0 to count compatible pairs.

Step 7: For each element in g, check if there exists an element in c such that c[i] <= g[j]:

- If a compatible element is found, increment co and stop checking further for that g[j]. **Step 8:** Print the value of co.

  Step 9:
  Stop

PROGRAM :

```c
#include <stdio.h>
int main() {
    int n, m,co=0;
    scanf("%d", &n);
    int g[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &g[i]);
    }
    scanf("%d", &m);
    int c[m];
    for (int i = 0; i < m; i++) {
        scanf("%d", &c[i]);
    }
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(c[i]<=g[j])
            {
                co++;
                break;
            }
        }

    }
    printf("%d\n", co);
}
```

OUTPUT:

	Input	Expected	Got	
✔	2 1 2 3 1 2 3	2	2	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

QUESTION 3.C

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 $3^i * c$ 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 $(3^0 * 1) + (3^1 * 3) + (3^2 * 2) = 1 + 9 + 18 = 28$. 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
```

For example:

Test	Input	Result
Test Case 1	3 1 3 2	18

ALGORITHM:

Step 1: Start

Step 2: Input the integer n, the number of elements in array c.

Step 3: Input n integers into array c.

Step 4: Sort the array c in descending order.

Step 5: Initialize k to 0 to store the weighted sum.

Step 6: For each element c[i], calculate c[i] * n^i and add it to k.

Step 7: Print the value of k.

Step 8: Stop

PROGRAM:

```c
#include <stdio.h>
#include<math.h>
int main()
{
    int n;
    scanf("%d",&n);
    int c[n];
    for(int i=0;i<n;i++){
        scanf("%d",&c[i]);
    }
    int temp = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
           if(c[i] < c[j]) {
               temp = c[i];
               c[i] = c[j];
               c[j] = temp;
           }
        }
    }
    int k=0;
    for(int i=0;i<n;i++)
    {
        k+=(pow(n,i)*c[i]);
    }
    printf("%d",k);
}
```

OUTPUT:

	Test	Input	Expected	Got	
✔	Test Case 1	3 1 3 2	18	18	✔
✔	Test Case 2	4 7 4 9 6	389	389	✔
✔	Test Case 3	3 5 10 7	76	76	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

QUESTION 3.D

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:

Step 1: Start

Step 2: Input the integer n, the number of elements in array a.

Step 3: Input n integers into array a.

Step 4: Sort the array a in ascending order.

Step 5: Initialize sum to 0 to store the weighted sum.

Step 6: For each element a[i], multiply it by its index i and add it to sum.

Step 7: Print the value of sum.

Step 8: Stop

PROGRAM:

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

OUTPUT:

	Input	Expected	Got	
✔	5 2 5 3 4 0	40	40	✔
✔	10 2 2 2 4 4 3 3 5 5 5	191	191	✔
✔	2 45 3	45	45	✔

Passed all tests! ✔

QUESTION 3.E

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.

For example:

Input	Result
3 1 2 3 4 5 6	28

ALGORITHM :

Step 1: Start

Step 2: Input the integer n, the number of elements in arrays a and b.

Step 3: Input n integers into array a.

Step 4: Input n integers into array b.

Step 5: Sort array a in ascending order.

Step 6: Sort array b in descending order.

Step 7: Initialize min to 0 to store the minimum weighted sum.

Step 8: For each index i, multiply a[i] and b[i] and add the result to min.

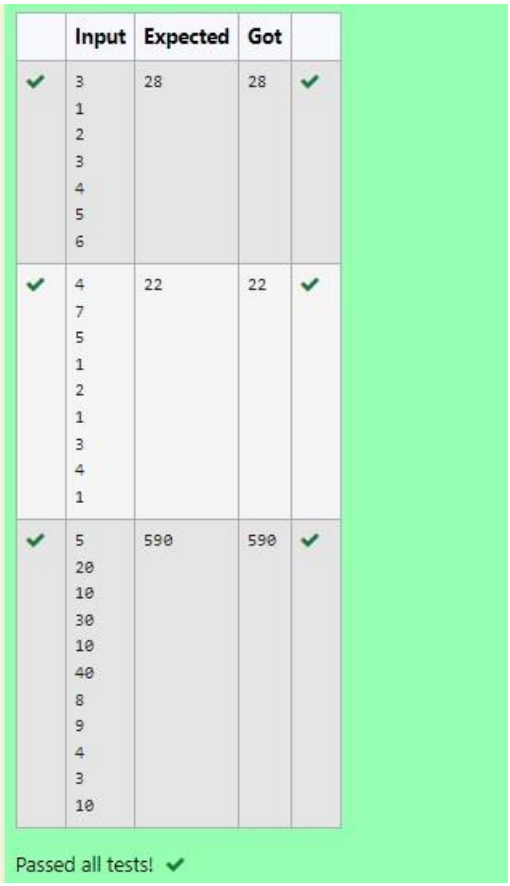
Step 9: Print the value of min.

Step 10: Stop

PROGRAM :

```c
#include<stdio.h>

int main()
{
    int n;
    scanf("%d",&n);
    int a[n],b[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(int i=0;i<n;i++)
    {
        scanf("%d",&b[i]);
    }
    int temp = 0;
    for (int i=0;i<n;i++)
    {
        for(int j=i+1;j<n;j++)
        {
            if(a[i]>a[j])
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
    }
    for (int i= 0; i < n; i++)
    {
        for (int j=i+1;j<n;j++)
        {
           if(b[i]<b[j])
           {
               temp=b[i];
               b[i]=b[j];
               b[j]=temp;
           }
        }
    }
    int min=0;
    for(int i=0;i<n;i++)
    {
        min+=(a[i]*b[i]);
    }
    printf("%d",min);
}
```

OUTPUT:



	Input	Expected	Got	
✔	3 1 2 3 4 5 6	28	28	✔
✔	4 7 5 1 2 1 3 4 1	22	22	✔
✔	5 20 10 30 10 40 8 9 4 3 10	590	590	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

DIVIDE AND CONQUER

QUESTION 4.A AIM:

Problem Statement

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 :

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array a of size n
Step 4: For each index i from 0 to n-1, input a[i]
Step 5: Call the function countz(a, 0, n - 1) and store its result in count
Step 6: Print the value of count Step
7: Stop

Function **countz(a[], l, r)**:
Step 1: If l > r, return 0
Step 2: Calculate mid as l + (r - l) / 2 Step
3: Initialize count to 0
Step 4: If a[mid] == 0, set count = 1
Step 5: Return count + countz(a, l, mid - 1) + countz(a, mid + 1, r)

PROGRAM :

```c
#include <stdio.h>

int countz(int a[],int l,int r);

int main()
{
    int n;
    scanf("%d",&n);
    int a[n];
    for (int i=0;i<n;i++) {
        scanf("%d",&a[i]);
    }
    int count=countz(a,0,n-1);
    printf("%d",count);

    return 0;
}

int countz(int a[],int l,int r)
{
    if (l>r)
    {
        return 0;
    }
    int mid=l+(r-l)/2;
    int count=0;
    if (a[mid]==0)
    {
        count=1;
    }
    return count + countz(a, l, mid - 1) + countz(a, mid + 1, r);
}
```

OUTPUT:

	Input	Expected	Got	
✔	5 1 1 1 0 0	2	2	✔
✔	10 1 1 1 1 1 1 1 1 1 1	0	0	✔

RESULT :

The above program is executed successfully .

QUESTION 4.B

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 * 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

ALOGORITHM :

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array a of size n
Step 4: For each index i from 0 to n-1, input a[i]
Step 5: Call the function majority(a, 0, n - 1) and store its result in majoele Step 6: If majoele is not -1, print majoele; otherwise, print "No Majority Element" Step 7: Stop

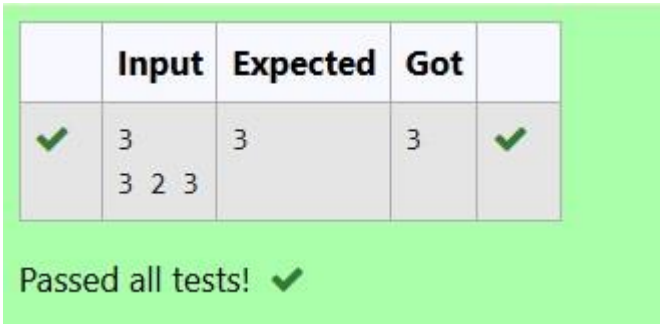
Function majority(a[], l, r):
Step 1: If l == r, return a[l]
Step 2: Calculate mid as (l + r) / 2
Step 3: Call majority(a, l, mid) and store its result in leftmajo
Step 4: Call majority(a, mid + 1, r) and store its result in rightmajo
Step 5: Initialize lc and rc to 0
Step 6: For each index i from l to r, if a[i] == leftmajo, increment lc; if a[i] == rightmajo, increment rc

Step 7: If lc > (r - l + 1) / 2, return leftmajo
Step 8: If rc > (r - l + 1) / 2, return rightmajo
Step 9: Return -1

PROGRAM :

```c
#include <stdio.h>

int majority(int a[], int l, int r)
{
    if (l == r)
    {
        return a[l];
    }
    int mid = (l + r) / 2;
    int leftmajo = majority(a, l, mid);
    int rightmajo = majority(a, mid + 1, r);

    int lc = 0, rc = 0;
    for (int i = l; i <= r; i++)
    {
        if (a[i] == leftmajo) lc++;
        if (a[i] == rightmajo) rc++;
    }
    if (lc > (r - l + 1) / 2)
    {
        return leftmajo;
    }
    if (rc > (r - l + 1) / 2)
    {
        return rightmajo;
    }
    return -1;
}

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

    int majoele = majority(a, 0, n - 1);

    if (majoele != -1)
    {
        printf("%d\n",majoele);
    }
    else
    {
        printf("No Majority Element\n");
    }
}
```

OUTPUT :



	Input	Expected	Got	
✔	3 3 2 3	3	3	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

QUESTION 4.C

AIM :

Problem Statement:
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 :

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array a of size n
Step 4: For each index i from 0 to n-1, input a[i]
Step 5: Input integer k
Step 6: Call findfloor(a, 0, n - 1, k) Step
7: Stop

Function findfloor(a[], l, r, key):
Step 1: If a[r] <= key, print a[r] and return
Step 2: If l < r, do Steps 3 and 4
Step 3: Calculate mid as (l + r) / 2
Step 4: Call findfloor(a, mid + 1, r, key)
Step 5: Call findfloor(a, l, mid, key)

PROGRAM :

```c
#include<stdio.h>
int search(int[],int,int,int);
int search(int arr[],int x,int left,int right)
{
    int mid=left+(right-left)/2;
     if(arr[mid]<=x)
        {
            int max = arr[mid];
            for(int i=0;i<mid;i++){
                if(arr[i]>=max)
                    max=arr[i];
            }
            return max;
        }
      else if(arr[mid]>x)
      {
        return search(arr,x,left,mid);
        }
    else
        return search(arr,x,mid+1,right);
}

int main()
{
    int n,x,floor;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    scanf("%d",&x);
    floor = search(arr,x,0,n-1);
    printf("%d",floor);
    return 0;
}
```

OUTPUT:

	Input	Expected	Got	
✔	6 1 2 8 10 12 19 5	2	2	✔

RESULT:

The above program is executed successfully.

QUESTION 4.B

AIM :

Problem Statement:
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 :

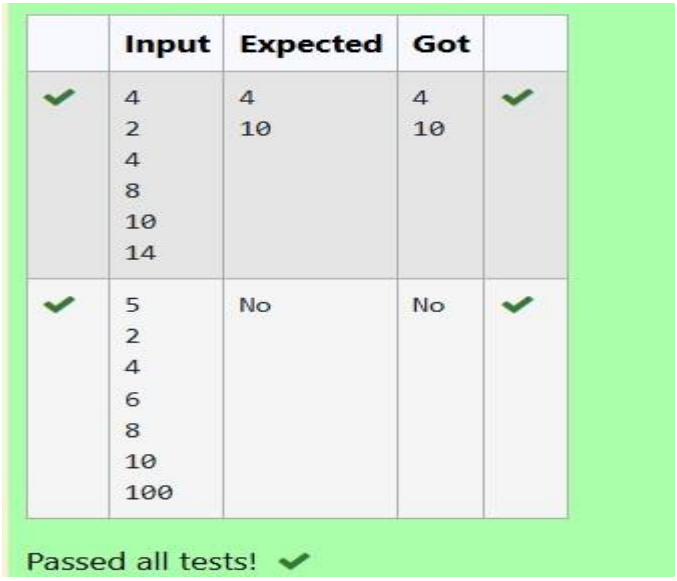
Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array arr of size n
Step 4: For each index i from 0 to n-1, input arr[i]
Step 5: Input integer x
Step 6: Call findPair(arr, 0, n - 1, x) Step
7: Stop

Function findPair(arr[], left, right, x):
Step 1: If left >= right, print "No" and return
Step 2: Calculate sum as arr[left] + arr[right]
Step 3: If sum == x, print arr[left] and arr[right], and return
Step 4: If sum < x, call findPair(arr, left + 1, right, x)
Step 5: Otherwise, call findPair(arr, left, right - 1, x)

PROGRAM :

```c
#include<stdio.h>
void twosum(int arr[],int left,int right,int x){
    if (left >= right){
        printf("No");
        return;
    }
    int sum=arr[left]+arr[right];
    if (sum==x){
        printf("%d\n",arr[left]);
        printf("%d\n",arr[right]);
    }
    else if(sum<x){
        twosum(arr,left+1,right,x);
    }
    else{
        twosum(arr,left,right-1,x);
    }
}
int main(){
    int n,x;
    scanf("%d",&n);
    int arr[n];
    for (int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    scanf("%d",&x);
    twosum(arr,0,n-1,x);
    return 0;
}
```

OUTPUT:



	Input	Expected	Got	
✔	4 2 4 8 10 14	4 10	4 10	✔
✔	5 2 4 6 8 10 100	No	No	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 4.E

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

For example:

Input	Result
5 67 34 12 98 78	12 34 67 78 98

ALGORITHM :

Step 1: Start
Step 2: Input the integer n
Step 3: Initialize array arr of size n
Step 4: For each index i from 0 to n-1, input arr[i]

Step 5: Call quickSort(arr, 0, n - 1)
Step 6: For each index i from 0 to n-1, print arr[i] Step 7: Stop

Function quickSort(arr[], left, right):
Step 1: If left < right, do Steps 2 to 7
Step 2: Set pivot to (left + right) / 2
Step 3: Initialize i to left and j to right
Step 4: While i < j, do Steps 5.1 to 5.4
Step 5.1: While arr[pivot] >= arr[i], increment i
Step 5.2: While arr[pivot] < arr[j], decrement j
Step 5.3: If i <= j, swap arr[i] and arr[j]
Step 6: Swap arr[j] and arr[pivot]
Step 7: Call quickSort(arr, left + 1, right)

PROGRAM :

```c
#include<stdio.h>
void quicksort(int arr[],int left,int right){
    if(left<right){
        int j=right;
        int i=left;
        int pivot=left;
        while(i<j){
            while(arr[i]<=arr[pivot]){
                i++;
            }
            while(arr[j]>arr[pivot]){
                j--;
            }
            if(i<j){
                int temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
        int temp=arr[j];
        arr[j]=arr[pivot];
        arr[pivot]=temp;
        quicksort(arr,left,j-1);
        quicksort(arr,j+1,right);
    }
}

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]);
    }
}
```

OUTPUT :

	Input	Expected	Got	
✔	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98	✔
✔	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56 78 90 90 114	✔
✔	12 9 8 7 6 5 4 3 2 1 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully .

REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

DYNAMIC PROGRAMMING

QUESTION 5.A AIM

:

Playing with Numbers:

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 :

Step 1: Start
Step 2: Input an integer n
Step 3: Initialize an array dp of size n+1
Step 4: Set dp[0] to 1
Step 5: For each index i from 1 to n, set dp[i] to 0
Step 6: For each index i from 1 to n, do Steps 7 and 8

PROGRAM :

Step 7: Add dp[i - 1] to dp[i]
Step 8: If i >= 3, add dp[i - 3] to dp[i]
Step 9: Print dp[n] Step
10: Stop

```c
#include<stdio.h>
int main() {
    int n;
    scanf("%d", &n);
    long dp[n+1];
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        dp[i] = 0;
    }
    for (int i = 1; i <= n; i++) {
        dp[i] += dp[i - 1];
        if (i >= 3) {
            dp[i] += dp[i - 3];
        }
    }
    printf("%ld\n", dp[n]);
    return 0;
}
```

OUTPUT :

	Input	Expected	Got	
✔	6	6	6	✔
✔	25	8641	8641	✔
✔	100	24382819596721629	24382819596721629	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 5.B

AIM:

Playing with Chessboard:

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 :

Step 1: Start
Step 2: Input an integer n
Step 3: Initialize a 2D array board of size n x n
Step 4: For each row i from 0 to n-1, and each column j from 0 to n-1, input board[i][j]
Step 5: Call maxMonetaryPath(n, board) and store the result in result
Step 6: Print result Step
7: Stop

Function maxMonetaryPath(n, board):

Step 1: Initialize a 2D array dp of size n x n
Step 2: Set dp[0][0] to board[0][0]
Step 3: For each column j from 1 to n-1, set dp[0][j] = dp[0][j-1] + board[0][j]
Step 4: For each row i from 1 to n-1, set dp[i][0] = dp[i-1][0] + board[i][0]
Step 5: For each row i from 1 to n-1, and each column j from 1 to n-1, set dp[i][j] = board[i][j] + max(dp[i-1][j], dp[i][j-1])
Step 6: Return dp[n-1][n-1]

PROGRAM :

```c
#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:

	Input	Expected	Got	
✔	3 1 2 4 2 3 4 8 7 1	19	19	✔
✔	3 1 3 1 1 5 1 4 2 1	12	12	✔
✔	4 1 1 3 4 1 5 7 8 2 3 4 6 1 6 9 0	28	28	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

QUESTION 5.C 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 :

Step 1: Start
Step 2: Input two strings `s1` and `s2`
Step 3: Calculate the lengths `len1` of `s1` and `len2` of `s2`
Step 4: Initialize a 2D array `dp` of size `(len1 + 1) x (len2 + 1)`
Step 5: For each index `i` from `0` to `len1`, and each index `j` from `0` to `len2`, do Steps 6-8
Step 6: If `i == 0` or `j == 0`, set `dp[i][j] = 0`
Step 7: If `s1[i-1] == s2[j-1]`, set `dp[i][j] = dp[i-1][j-1] + 1`
Step 8: Otherwise, set `dp[i][j]` to the maximum of `dp[i][j-1]` and `dp[i-1][j]`
Step 9: Print `dp[len1][len2]`
Step 10: Stop

PROGRAM :

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char s1[10],s2[10];
    scanf("%s",s1);
    scanf("%s",s2);
    int len1=strlen(s1);
    int len2=strlen(s2);
    int dp[len1 + 1][len2 + 1];
    for(int i=0;i<=len1;i++)
    {
        for(int j=0;j<=len2;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{
                if(dp[i][j-1]>dp[i-1][j])
                    dp[i][j]=dp[i][j-1];
                else
                    dp[i][j]=dp[i-1][j];
                }
        }
    }
    printf("%d",dp[len1][len2]);
}
```

OUTPUT :

	Input	Expected	Got	
✔	aab azb	2	2	✔
✔	ABCD ABCD	4	4	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

QUESTION 5.D 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 :

Step 1: Start
Step 2: Input an integer n
Step 3: Initialize an array arr of size n
Step 4: For each index i from 0 to n-1, input arr[i]
Step 5: Call subsequence(arr, n) and store the result in result
Step 6: Print result Step
7: Stop

Function subsequence(arr, n):
Step 1: Initialize an array dp of size n
Step 2: Set each element in dp to 1
Step 3: Initialize maxlen to 1
Step 4: For each index i from 1 to n-1, do Steps 5-7
Step 5: For each index j from 0 to i-1, if arr[i] >= arr[j] and dp[i] < dp[j] + 1, set dp[i] = dp[j] + 1
Step 6: If maxlen < dp[i], set maxlen = dp[i] Step
7: Return maxlen

PROGRAM :

```c
#include <stdio.h>
int subsequence(int arr[],int n){
    int dp[n];
    int maxlen=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(arr[i]>=arr[j] && dp[i]<dp[j]+1){
                dp[i]=dp[j]+1;
            }
        }
        if(maxlen<dp[i]){
            maxlen=dp[i];
        }
    }
    return maxlen;
}

int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    for (int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int result=subsequence(arr,n);
    printf("%d",result);
}
```

OUTPUT :

	Input	Expected	Got	
✔	9 -1 3 4 5 2 2 2 2 3	6	6	✔
✔	7 1 2 2 4 5 7 6	6	6	✔

Passed all tests! ✔

RESULT :

The above program is executed successfully.

REG NO: 230701161

NAME : Laksheta SV

DEPT : CSE - C

COMPETITIVE PROGRAMMING

QUESTION 6.A 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

For example:

Input	Result
5 1 1 2 3 4	1

ALGORITHM:

Step 1: Start

Step 2: Input the integer n, the number of elements in the array.

Step 3: Input n integers into an array a.

Step 4: Initialize r as -1 to store the repeated element.

Step 5: Use a nested loop to check if any element a[i] matches with subsequent elements a[j].

Step 6: If a match is found, set r to the repeated element.

Step 7: If a repeated element is found (r != -1), print the repeated element. **Step 8:**

Stop

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

}
```

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	✔

Passed all tests! ✔

RESULT:

The above progeam is executed successfully .

QUESTION 6.B 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

For example:

Input	Result
5 1 1 2 3 4	1

ALGORITHM:

Step 1: Start

Step 2: Input the integer n, the number of elements in the array.

Step 3: Input n integers into an array a.

Step 4: Initialize a boolean array r[100] to track whether a number has already been encountered.

Step 5: Iterate through the array a. For each element, check if it has already been seen.

Step 6: If the element is already seen, print it. If not, mark it as seen in r. **Step**

7: Stop

PROGRAM:

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

int main() {
    int n;
    scanf("%d", &n);
    int a[n];
    bool r[100] = {false};
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        if (r[a[i]]) {
            printf("%d ", a[i]);
        } else {
            r[a[i]] = true;
        }
    }
}
```

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	✔

Passed all tests! ✔

RESULT:

The above program is executed successfully.

Question 6.C 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

ALLGORITHM:

Step 1: Start

Step 2: Input the number of test cases t.

Step 3: For each test case, input the size n1 of the first array and input the array arr1.

Step 4: Input the size n2 of the second array and input the array arr2.

Step 5: For each element of arr1, check if it exists in arr2.

Step 6: If a match is found, print the element as part of the intersection.

Step 7: Stop

PROGRAM:

```c
#include <stdio.h>
void intersection(int arr1[],int n1,int arr2[],int n2){
    for (int i=0;i<n1;i++){
        int element=arr1[i];
        for (int j=0;j<n2;j++){
            if (arr2[j]==element) {
                printf("%d ",element);
                break;
            }
        }
    }
    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]);
        }
        intersection(arr1,n1,arr2,n2);
    }
}
```

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! ✔

RESULT:

The above program is executed successfully.

Question 6.D 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

ALGORITM:

Step 1: Start

Step 2: Input the number of test cases t.

Step 3: For each test case, input the size n1 of the first array and input the array arr1.

Step 4: Input the size n2 of the second array and input the array arr2.

Step 5: Initialize two indices i and j to 0 and use them to traverse both arrays.

Step 6: If arr1[i] < arr2[j], increment i. If arr2[j] < arr1[i], increment j.

Step 7: If arr1[i] == arr2[j], print the common element and increment both i and j.

Step 8: Continue until one of the arrays is completely traversed.

Step 9: Stop

PROGRAM:

```c
#include <stdio.h>
void intersection(int arr1[], int n1, int arr2[], int n2) {
    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");
}

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]);
        }
        intersection(arr1,n1,arr2,n2);
    }
}
```

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! ✔

RESULT:

The above program is executed successfully.

Question 6.E 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:

Step 1: Start

Step 2: Input the integer n (number of elements) and the array arr.

Step 3: Input the integer k (difference to check for).

Step 4: Use a nested loop to compare each pair of elements arr[i] and arr[j].

Step 5: If the difference arr[j] - arr[i] == k, return 1.

Step 6: If the difference exceeds k, break the inner loop.

Step 7: If no valid pair is found, return 0.

Step 8: Output the result.

Step 9: Stop

PROGRAM :

```c
#include <stdio.h>
int checkpair(int arr[],int n,int k){
    for (int i=0;i<n;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);
    int result=checkpair(arr,n,k);
    printf("%d\n",result);
}
```

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! ✔

RESULT:

The above program is executed successfully.

Question 6.F

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:

Step 1: Start

Step 2: Input the integer n (number of elements) and the array arr.

Step 3: Input the integer k (difference to check for).

Step 4: Initialize two indices i = 0 and j = 1.

Step 5: While j < n, calculate the difference arr[j] - arr[i].

Step 6: If the difference is k, return 1.

Step 7: If the difference is less than k, increment j. If the difference is greater, increment i.

Step 8: If i == j, increment j to avoid comparing the same element with itself.

Step 9: If no valid pair is found, return 0.

Step 10: Output the result.

Step 11: Stop

PROGRAM:

```c
#include <stdio.h>
int checkpair(int arr[],int n,int k){
    int i=0,j=1;
    while(j<n){
        int diff=arr[j]-arr[i];
        if (diff==k && i!=j){
            return 1;
        }
        else if(diff<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);
    int result=checkpair(arr,n,k);
    printf("%d\n",result);
}
```

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! ✔

RESULT :

The above program is executed successfully.