NAME: SREYA G **REGISTER NO.:** 230701334

CLASS : CSE F **DATE:** 10/09/2024

EX – 5:

DYNAMIC PROGRAMMING:

PROBLEM 1:

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
```

ALGORITM:

1. Input n:
   - Read the integer n from the user.
2. Initialize an array dp:
   - Create an array dp of size n + 1.
   - Set dp[0] = 1, dp[1] = 1, dp[2] = 1, and dp[3] = 2.
3. Calculate values for dp[i] (for i = 4 to n):
   - Use the formula:
     - dp[i] = dp[i - 1] + dp[i - 3].
4. Return dp[n]:
   - The value at dp[n] gives the result.
5. Output the result:
   - Print the value returned by the function ways(n).
6. End.

CODE:

```
#include<stdio.h>

long long ways(int n){
    long long dp[n + 1];
    dp[0] = 1;
    dp[1] = 1;
    dp[2] = 1;
    dp[3] = 2;
    int i;
```

```
    for(i = 4;i <= n;i++){
        dp[i] = dp[i - 1] + dp[i - 3];
    }
    return dp[n];
}

int main(){
    int n;
    scanf("%d",&n);
    printf("%lld\n",ways(n));
    return 0;
}
```

OUTPUT:

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

Passed all tests! ✔

Correct

Marks for this submission: 10.00/10.00.

RESULT:

Thus the code is executed successfully and gives the expected output.

PROBLEM 2:

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:

1. Input n.
2. Initialize a 2D array cb[n][n] and read its values.
3. Initialize a 2D array dp[n][n] with dp[0][0] = cb[0][0].
4. Fill the first row and column of dp using the values from cb.
5. For each i and j, update dp[i][j] as the maximum sum from the top or left cell.
6. Output dp[n-1][n-1].
7. End.

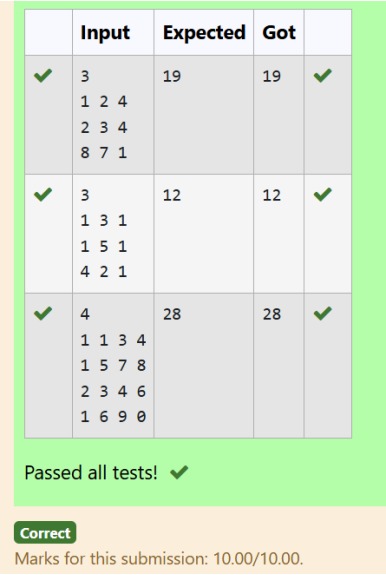
CODE:

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

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

    int cb[n][n];
    int i,j;

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

    for(i = 1;i < n;i++){
        dp[i][0] = dp[i - 1][0] + cb[i][0];
        dp[0][i] = dp[0][i - 1] + cb[0][i];
    }

    for(i = 1;i < n;i++){
        for(j = 1;j < n;j++){
            if(dp[i - 1][j] > dp[i][j - 1]){
                dp[i][j] = cb[i][j] + dp[i - 1][j];
            }else{
                dp[i][j] = cb[i][j] + dp[i][j -1];
```

```
            }
        }
    }
    printf("%d\n",dp[n - 1][n - 1]);
    return 0;
}
```

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! ✔

Correct

Marks for this submission: 10.00/10.00.

RESULT:

Thus the code is executed successfully and gives the expected output.

PROBLEM 3:

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:

1. Input strings s1 and s2.
2. Initialize m = length of s1 and n = length of s2, and create a 2D array dp[m+1][n+1].
3. Set base cases: dp[i][0] = 0 and dp[0][j] = 0.
4. For each i and j, if s1[i-1] == s2[j-1], set dp[i][j] = dp[i-1][j-1] + 1; else, set dp[i][j] = max(dp[i-1][j], dp[i][j-1]).
5. Return dp[m][n] as the LCS length.
6. Output the result.

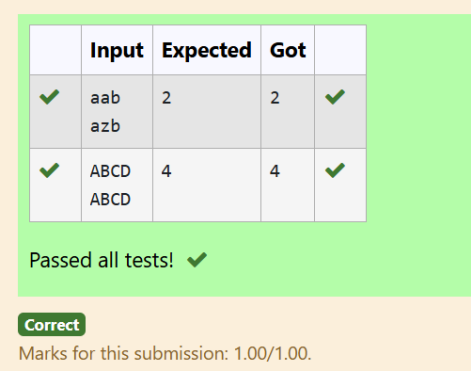
CODE:

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

int clsLen(char * s1,char * s2){
    int m = strlen(s1);
    int n = strlen(s2);
```

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

    int i,j;

    for(i = 0;i <= m;i++){
        for(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{
                    if(dp[i - 1][j] > dp[i][j - 1]){
                        dp[i][j] = dp[i - 1][j];
                    }else{
                        dp[i][j] = dp[i][j - 1];
                    }
                }
            }
        }
    }
    return dp[m][n];
}

int main(){
    char s1[100],s2[100];
```

```
    scanf("%s",s1);
    scanf("%s",s2);

    int answer = clsLen(s1,s2);
    printf("%d\n",answer);

    return 0;
}
```

OUTPUT:



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

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

RESULT:

Thus the code is executed successfully and gives the expected output.

PROBLEM 4:

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:

1. Input array arr[] and size n.
2. Initialize a DP array dp[] with all values set to 1.
3. For each element `i` from 1 to n-1, and for each element `j` from 0 to `i-1`:
   - If `arr[i] >= arr[j]`, update `dp[i] = max(dp[i], dp[j] + 1)`.
4. Find the maximum value in `dp[]` to get the length of the longest non-decreasing subsequence.
5. Output the result.

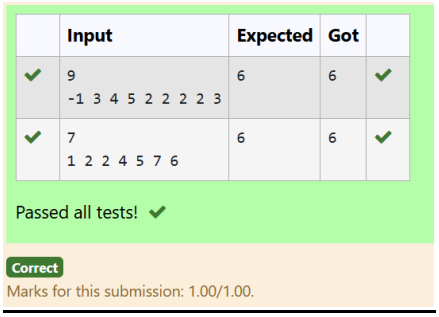
CODE:

```
#include<stdio.h>
int lonNonDecSubseq(int arr[],int n){
    int dp[n];
    int i,j,maxlen = 1;

    for(i = 0;i < n;i++){
        dp[i] = 1;
    }
```

```
    for(i = 1;i < n;i++){
        for(j = 0;j < i;j++){
            if(arr[i] >= arr[j]){
                if(dp[i] < dp[j] + 1){
                    dp[i] = dp[j] + 1;
                }
            }
        }
    }

    for(i = 0;i < n;i++){
        if(dp[i] > maxlen){
            maxlen = dp[i];
        }
    }
    return maxlen;
}

int main(){
    int arr[] = {-1,3,4,5,2,2,2,2,3};
    int n = sizeof(arr) / sizeof(arr[0]);

    int answer = lonNonDecSubseq(arr,n);
    printf("%d\n",answer);

    return 0;
```

}

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! ✔

Correct

Marks for this submission: 1.00/1.00.

RESULT:

Thus the code is executed successfully and gives the expected output.