**RECURSION QUESTIONS**

Table of Contents

[**LEVEL 1: Easy** 2](#_Toc187681145)

[1. Find factorial of number using recursion. 2](#_Toc187681146)

[2. Find the nth term of Fibonacci series. 0,1,1,2,3,5,8… 2](#_Toc187681147)

[3. Recursive function to reverse the string. 2](#_Toc187681148)

[4. Print 1 to n numbers using recursion. 2](#_Toc187681149)

[5. Print 1 to n numbers in reverse order (descending order). 2](#_Toc187681150)

[6. Write recursive function to calculate sum of ranges of numbers 2](#_Toc187681151)

[7. Find gcd of two numbers. 2](#_Toc187681152)

[8. Recursive function to count occurrences of given character in string 2](#_Toc187681153)

[9. Write a recursive function to find sum of all digits in a number. 2](#_Toc187681154)

[10. Find maximum number in the list of integers. 2](#_Toc187681155)

[11. Recursive function to find power of number. 2](#_Toc187681156)

[13. Print a special series 2](#_Toc187681157)

[LEVEL 2: **Moderate** 3](#_Toc187681158)

[1. Print star pattern. 3](#_Toc187681159)

[2. Print Pattern 3](#_Toc187681160)

[3. Find ways to go to party. 3](#_Toc187681161)

[4. Recursive function to merge two sorted lists. 3](#_Toc187681162)

[5. Print all subsets of array. 3](#_Toc187681163)

[6. Count sss binary strings 3](#_Toc187681164)

[7. Print all n length binary strings not having adjacent 1’s. 3](#_Toc187681165)

[8. Print all permutations of a string. 3](#_Toc187681166)

[9. Is subsequence? 3](#_Toc187681167)

[10. Print all paths in 2D grid 3](#_Toc187681168)

[11. Print all paths in 2D grid with diagonal move too 3](#_Toc187681169)

[LEVEL 3: **Difficult** 5](#_Toc187681170)

[LEVEL 4: **Expert** 6](#_Toc187681171)

[**SOLUTIONS:** 7](#_Toc187681172)

[**LEVEL 1:** 7](#_Toc187681173)

[**LEVEL 2:** 11](#_Toc187681174)

# **LEVEL 1: Easy**

### Find factorial of number using recursion.

### Find the nth term of Fibonacci series. 0,1,1,2,3,5,8…

### Recursive function to reverse the string.

### Print 1 to n numbers using recursion.

### Print 1 to n numbers in reverse order (descending order).

### Write recursive function to calculate sum of ranges of numbers

Eg: sum of all numbers from a to b (a and b inclusive)

### Find gcd of two numbers.

### Recursive function to count occurrences of given character in string

### Write a recursive function to find sum of all digits in a number.

Eg. For number 101249, sum of digits is 17

### Find maximum number in the list of integers.

### Recursive function to find power of number.

Eg. Find num power of x, in most effective way

### Recursive function to check if string is palindrome.

### Print a special series

You are given a series defined by Tn= (Tn-2)2 – (Tn-1) .Here 1st and 2nd values are 0,1. Print the first N values.

[GFG question](https://practice.geeksforgeeks.org/problems/gf-series3535/1?page=1&difficulty%5b%5d=-2&difficulty%5b%5d=-1&difficulty%5b%5d=0&category%5b%5d=Recursion&sortBy=difficulty)

# LEVEL 2: **Moderate**

### Print star pattern.

Eg. For n = 4 pattern is \*\*\*\* \*\*\* \*\* \*

### Print Pattern

Print a sequence of numbers starting with n, without using a loop. Replace n with n - 5, n−5... until n≤0. Then, replace n with n+5, n + 5, n+5 until n regains its initial value.

[GFG question](https://www.geeksforgeeks.org/problems/print-pattern3549/1?page=1&category=Recursion&difficulty=Easy&sortBy=accuracy)

### Find ways to go to party.

There are N persons who want to go to party. There is constraint that any person can either go alone or go in a pair. Calculate number of ways in which n persons can go to the party. Eg for n=3 , persons A,B,C can go as [A,B,C] , [A,(B,C)] , [(A,B),C] , [(A,C),B] = total 4 ways

### Recursive function to merge two sorted lists.

### Print all subsets of array.

### Count of binary strings

Count number of ways to make a binary string of length n, where there is no adjacent 1.

### Print all n length binary strings not having adjacent 1’s.

### Print all permutations of a string.

### Is subsequence?

Given two strings s and t, return true if s is a subsequence of t, or false otherwise. A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

### Print all paths in 2D grid

You are given a 2D grid (m × n), you are at the top left point and you need to reach bottom right point. You can only go right or down. Count all possible ways and also print all the paths.

### Print all paths in 2D grid with diagonal move too

You are given a 2D grid (m × n), you are at the top left point and you need to reach bottom right point. You can only go right or down and diagonally down too. Print all the paths.

# LEVEL 3: **Difficult**

# LEVEL 4: **Expert**

# **SOLUTIONS:**

## **LEVEL 1:**

1. Factorial of number

def fact(n):

    if n==1:            *#base case*

        return 1

    sub = fact(n-1)     *#recursive subproblem*

    return n\*sub        *#self work*

print(fact(3))

1. Fibonacci Series

def fib(n):

    if(n<=1):          *#base case*

        return n

    sub1 = fib(n-1)    *#recursive subproblem*

    sub2 = fib(n-2)

    return sub1+sub2   *#self work*

print(fib(6))

1. Reverse strings

def rev(i,j,s):

    if i>=j:

        return ''.join(s)

    s[i],s[j] = s[j],s[i]

    return rev(i+1,j-1,s)

s=list("Palash")

print(rev(0,len(s)-1,s))

1. Print 1 to n

def print\_nums(n):

    if(n==0):          *#base case*

        return

    print\_nums(n-1)    *#recursive subproblem*

    print(n)           *#self work*

print\_nums(5)

1. Print 1 to n in descending order

def print\_nums\_desc(n):

    if(n==0):          *#base case*

        return

    print(n)           *#self work*

    print\_nums\_desc(n-1)    *#recursive subproblem*

print\_nums\_desc(5)

1. Range sum ( sum of all numbers from start to end )

def range\_sum(start,end):

    if start>end :                             *#base case*

        return 0

    return start + range\_sum(start+1,end)   *#self work,recursive subproblem*

print(range\_sum(3,5))

1. GCD(a,b) //here a>b. The Euclidean algorithm is a method for finding the greatest common divisor (GCD) of two numbers. It is based on the observation that if a and b are two positive integers with a > b, then the GCD of a and b is the same as the GCD of b and a % b, where % is the modulo operator.

The Euclidean algorithm is very efficient, with a worst-case time complexity of O(log n), where n is the larger of the two numbers.

def gcd(a,b):

    if(a==0):           *#base case*

        return b

    sub = gcd(b%a,a)   *#recursive subproblem*

    return sub          *#self work*

print(gcd(42,18))

1. Character occurrences in string

def count\_occurances(string,char):

    if len(string)==0:                             *#base case*

        return 0

    if string[0]==char:

        return 1 + count\_occurances(string[1:],char)  *#self work,recursive subproblem*

    else:

        return count\_occurances(string[1:],char)    *#self work,recursive subproblem*

print(count\_occurances("python application",'p'))

1. Sum of digits of a number

def sum\_digits(num):

    if num<10:                                  *#base case*

        return num

    return num%10 + sum\_digits(num//10)         *#self work , recursive case*

print(sum\_digits(101249))

def maxNum(m,l,i):

    if i==len(l):

        return m

    if l[i]>m:

        return maxNum(l[i],l,i+1)

    return maxNum(m,l,i+1)

print(maxNum(0,[21,5,2,5,9,12],0))

1. Max number in array

def find\_max(arr):

    if len(arr) == 1:

        return arr[0]

    else:

        return max(arr[0], find\_max(arr[1:]))

if \_\_name\_\_ == '\_\_main\_\_':

    print(find\_max([1, 2, 3, 9, 5, 6, 7, 8]))

1. Find N\*\*X

def power(n,x):

    if x==1:

        return 1

    else:

        if(x%2):

            return n\*n\*n\*power(n,x//2)

        else:

            return n\*n\*power(n,x//2)

print(power(2,4))

1. Check Palindrome

def is\_palindrome(s):

    if len(s) <= 1:

        return True

    else:

        return s[0] == s[-1] and is\_palindrome(s[1:-1])

print(is\_palindrome('abcba'))

1. GFG series

class Solution:

    def gfSeries(self, N : int) -> None:

        # code here

        ans = [0]\*N

        if N>1:

            ans[0]=0

        if N>2:

            ans[1]=1

        def rec(N):

            if N<=2:

                return N-1

            else:

                temp = rec(N-2)

                temp2 =  temp\*temp-rec(N-1)

                ans[N-1] = temp2

                return temp2

        rec(N)

## **LEVEL 2:**

1. Print star pattern

def pattern(n,j):

    if n==0:

       return

    if j==0:

        print("\r")

        pattern(n-1,n-1)

    else:

        print("\*",end="")

        pattern(n,j-1)

pattern(4,4)

*#Via loop (above one, j act as for loop, and reducing n value act as while loop)*

n=4

for i in range(n,0,-1):

    for j in range(i):

        print("\*",end="")

    print("\r")

*#Also try printing reverse of this pattern, will need to use pattern(n,i,j)*

1. Print Pattern

class Solution:    *#learn use of global variable*

    def pattern(*self*, *N*):

*# code here*

        def helper(*num*,*N*,*flag*):

            ans.append(*num*)

*if* *num*<=0:

*flag*=1

*if* *flag*==1:

*if*  *num*==*N*:

*return*

                helper(*num*+5,*N*,1)

*else*:

                helper(*num*-5,*N*,*flag*)

        global ans

        ans= []   *#remember to make global arr null before running this each time*

        helper(*N*,*N*,0)  *#flag=0 means increasing, 1 means decreasing*

*return* ans

1. Ways to go to party

Let 3 persons are there A B C

F(n) denotes total number of ways to go to party

F(n) depends on 2 cases:

If A goes alone: F(n) = number of ways B,C can go to party = F(n-1)

If A goes in pair: F(n) = number of ways to make pair \* (number of ways other n-2 can go to party )

= (n-1) \* F(n-2)

So

F(n) = F(n-1) + (n-1)\*F(n-2)

def ways\_to\_party(n):

    if n<=1:

        return 1

    else:

        return ways\_to\_party(n-1) + (n-1)\*ways\_to\_party(n-2)

print(ways\_to\_party(3))

1. Merge two sorted arrays

def merge\_sorted(*a*,*b*,*osf*): *#With output so far method*

*if* len(*a*)==0:

*return* *osf*+*b*

*if* len(*b*)==0:

*return* *osf*+*a*

*else*:

*if* *a*[0]<*b*[0]:

*osf*+=[*a*[0]]

            merge\_sorted(*a*[1:],*b*,*osf*)

*else*:

*osf*+=[*b*[0]]

            merge\_sorted(*a*,*b*[1:],*osf*)

*return* *osf*

a= [2,4,5,7,10]

b= [1,3,6,8,9]

print(merge\_sorted(a,b,[]))

def merge\_arr(*arr1*,*arr2*,*i*,*j*): *#Without osf way*

*if* *i*==len(*arr1*):

*return* *arr2*[*j*:]

*if* *j*==len(*arr2*):

*return* *arr1*[*i*:]

*if*(*arr1*[*i*]<*arr2*[*j*]):

*return* [*arr1*[*i*]]+merge\_arr(*arr1*,*arr2*,*i*+1,*j*)

*else*:

*return* [*arr2*[*j*]]+merge\_arr(*arr1*,*arr2*,*i*,*j*+1)

a= [2,4,5,7,10]

b= [1,3,6,8,9]

print(merge\_arr(a,b,0,0))

1. All subsets

def subsets(i,n,array,s):     *#s= output so far*

    if i==n:

        print('['+s+']')

        return

    subsets(i+1,n,array,s+array[i])

    subsets(i+1,n,array,s)

array=["A","B","C"]

subsets(0,len(array),array,"")

*#output: [['A', 'B', 'C'], ['A', 'B'], ['A', 'C'], ['A'], ['B', 'C'], ['B'], ['C'], []]*

1. Count of Binary strings

It forms a Fibonacci series, for n=1 count =2 (0,1) , for n=2 count = 3(00,01,10) , for n=3 count =5(000,100,010,001,101)

*#count of binary strings of length n with no consecutive 1's*

def count\_binary\_strings(n):

    if n<=2:

        return n+1

    else:

        return count\_binary\_strings(n-1)+count\_binary\_strings(n-2)

print(count\_binary\_strings(4))

1. Print binary strings

*#osf = output so far*

*#one\_flag = True if last value added was 1 else False*

def binary\_strings(*n*,*i*,*flag*,*osf*):

*if* *i*==*n*:

        ans.append(*osf*)

*return* 1

*if*(*flag*==0):

*return* binary\_strings(*n*,*i*+1,0,*osf*+"0") + binary\_strings(*n*,*i*+1,1,*osf*+"1")

*else*:

*return* binary\_strings(*n*,*i*+1,0,*osf*+"0")

global ans

ans=[]

print(binary\_strings(4,0,0,""))  *#this return count of strings*

print(ans)

1. Print all permutations

def permutations(i,s):

    if i == len(s):

        print(''.join(s))

    else:

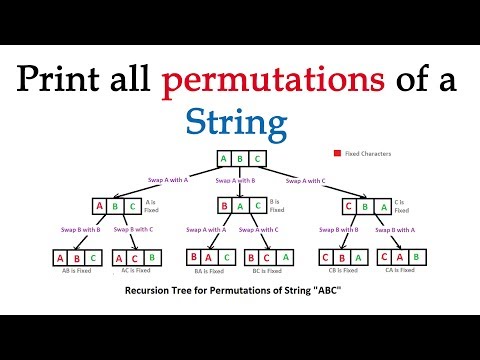
        for j in range(i,len(s)):

            s[i],s[j] = s[j],s[i]

            permutations(i+1,s)

            s[i],s[j] = s[j],s[i]

permutations(0,list('abc'))

[](https://www.youtube.com/watch?v=GuTPwotSdYw)

9. Is Subsequence

def sub(*a*,*b*,*i*,*j*):

*if* *j*==len(*b*):

*return* True

*if* *i*==len(*a*):

*return* False

*if*(*a*[*i*]==*b*[*j*]):

*return* sub(*a*,*b*,*i*+1,*j*+1)

*else*:

*return* sub(*a*,*b*,*i*+1,*j*)

print(sub("abcdepm","acfde",0,0))

10. All Paths

*#To print all the paths,*

*#l = total length of path, for any path taken l = n-1+m-1*

*#i,j = current position* *m,n = destination*

*#s = string to store the path*

def paths(i,j,m,n,s,l):

    if i==m:

        print(s+"D"\*(l-len(s)))

        return 1

    elif j==n:

        print(s+"R"\*(l-len(s)))

        return 1

    else:

        sub1 =  paths(i+1,j,m,n,s+"R",l)

        sub2 =  paths(i,j+1,m,n,s+"D",l)

        return sub1+sub2

n=2

m=3

print(paths(1,1,n,m,"",n-1+m-1))

To just get count of paths:

def count\_paths(m,n):

    if m==1 or n==1:

        return 1

    return count\_paths(m-1,n)+count\_paths(m,n-1)

print(count\_paths(2,3))

11. All paths including diagonal

*#i,j = current position m,n = destination*

*#s = string to store the path*

*#R - right, B - bottom, D - diagonal*

def paths(i,j,m,n,s):

    if i>m or j>n:

        return 0

    if i==m-1 and j==n-1:

        print(s)

        return 1

    else:

        sub1 =  paths(i+1,j,m,n,s+"R")

        sub2 =  paths(i,j+1,m,n,s+"B")

        sub3 =  paths(i+1,j+1,m,n,s+"D")

        return sub1+sub2+sub3

m=3

n=2

print(paths(0,0,m,n,""))