# Recursion

Basics :

## Print 1 to N numbers

Using recursion method1

void f**(**int i **,** int N **){**

**if(**i**>** N **)**

**return** **;**

cout **<<** i **<<** endl **;**

f**(**i**+**1 **,** N **)** **;**

**}**

int main**()** **{**

init\_code**()** **;**

//Solution sol ;

f**(**1 **,** 5 **)** **;**

**return** 0 **;**

**}**

Using recursion method 2

void f**(**int i **,** int j **){**

**if(**j **<** i **)**

**return** **;**

f**(**i **,** j**-**1 **)** **;**

cout **<<** j **<<** endl **;**

**}**

int main**()** **{**

init\_code**()** **;**

//Solution sol ;

f**(** 1 **,** 5 **)** **;**

**return** 0 **;**

**}**

## Sum of the first N natural numbers

Parameterized recursion

void f**(**int N **,** int sum **){**

**if(**N **==** 0 **){**

cout **<<** sum **;**

**return** **;**

**}**

f**(**N**-**1 **,** sum**+**N **)** **;**

**}**

int main**()** **{**

init\_code**()** **;**

//Solution sol ;

f**(** 5 **,** 0 **)** **;**

**return** 0 **;**

**}**

Functional recursion (return )

int f**(**int N **){**

**if(**N **==** 0 **){**

**return** 0**;**

**}**

**return** N **+** f**(**N**-**1 **)** **;**

**}**

int main**()** **{**

init\_code**()** **;**

//Solution sol ;

cout **<<** f**(** 5 **)** **;**

**return** 0 **;**

**}**

## Reversing Array

Using two pointer

void rev**(**int l **,** int r **,** int arr**[]** **)** **{**

**if(**l **>=** r **)**

**return** **;**

swap**(**arr**[**l**]** **,** arr**[**r**])** **;**

rev**(**l**+**1 **,** r**-**1 **,** arr **)** **;**

**}**

int main**()** **{**

init\_code**()** **;**

//Solution sol ;

int arr**[]** **=** **{**1 **,** 2 **,** 3 **,** 4 **,** 5 **}** **;**

**for(**int i**=**0 **;** i**<**5 **;** i**++** **)**

cout **<<** arr**[**i**]** **<<** " " **;**

cout **<<** endl **;**

rev**(**0 **,** 4 **,** arr **)** **;**

**for(**int i**=**0 **;** i**<**5 **;** i**++** **)**

cout **<<** arr**[**i**]** **<<** " " **;**

cout **<<** endl **;**

**return** 0 **;**

**}**

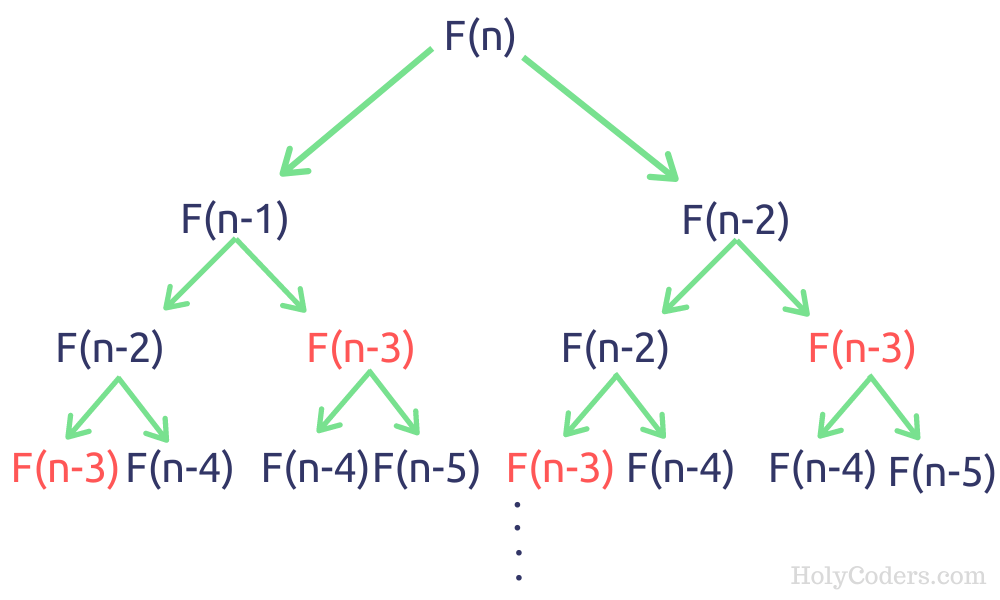
(We can consider slightly changing above two pointer to ‘i’ , ‘n-1-i’ , i>=n/2 will be the basecondition)

## Time Complexity of the Fibonnaci Series :

It is basically how many recursion calls we are doing overall.

2^0+2^1+...+2^n = 2^n+1-1 , here basically there will be n-1 levels.

Simply , at each node in the tree there are 2 splits,hence it will be 2^n



# Printing Subsequences

Let us say given arr[] = {3 , 2 , 1 } , then all the subsequnces of that would be

{}

3

2

1

3 2

3 1

2 1

3 2 1

Recursion on Subsequnces

This can be achived using the take/not-take approach , in one recursion call we will add the current element and in the another recursion call we will not add that.

**Time complexity :** 2^n x n as for every element there are tow choices either to pick or not to pick , n is nearly for printing the answer in the for loop.

## Printing Subsequences whose sum is k