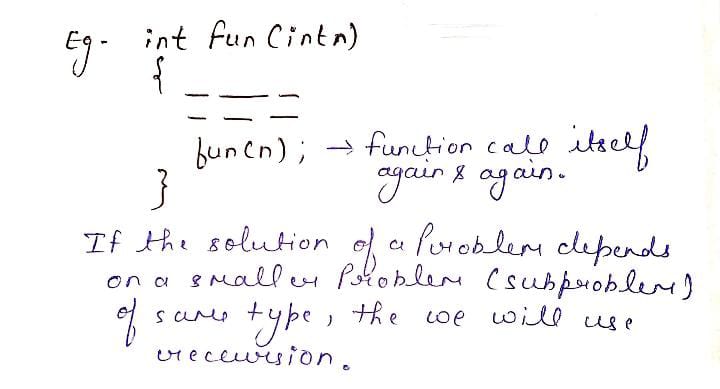
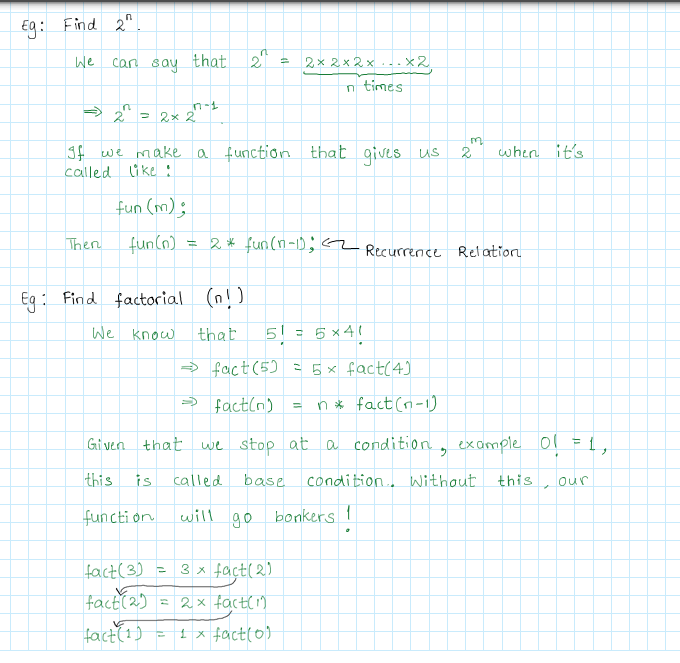
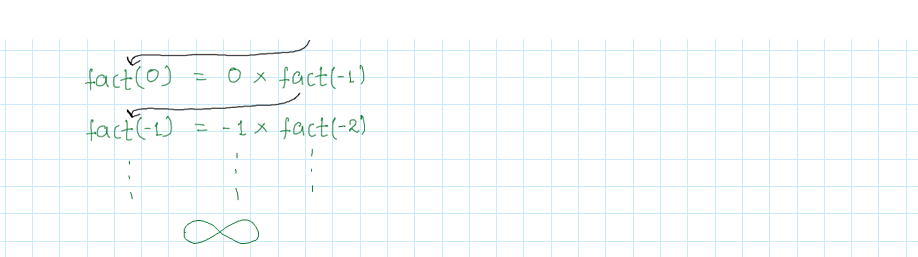
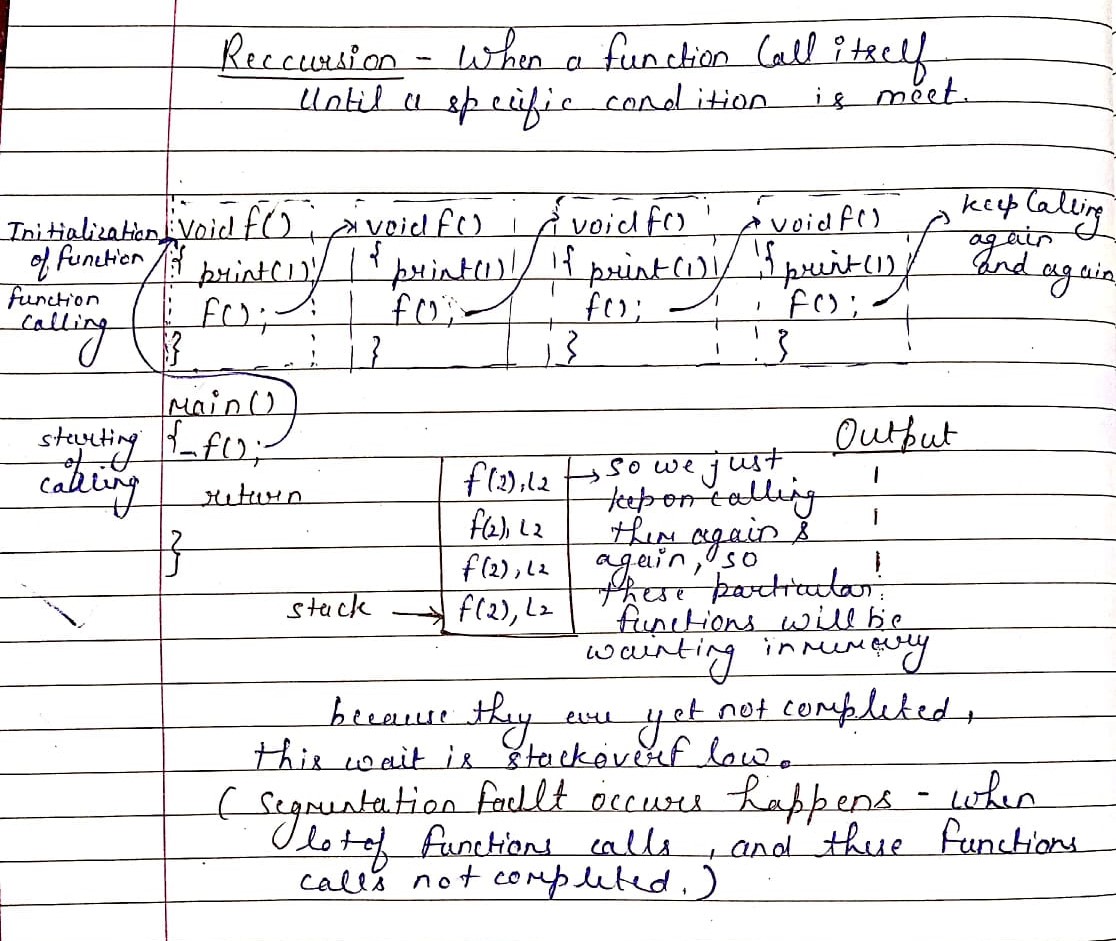
**Recursion -** The process in which a function calls itself directly or indirectly is called recursion and **the corresponding function is called a recursive function** . And This Process is Call **Reccursive Call .**

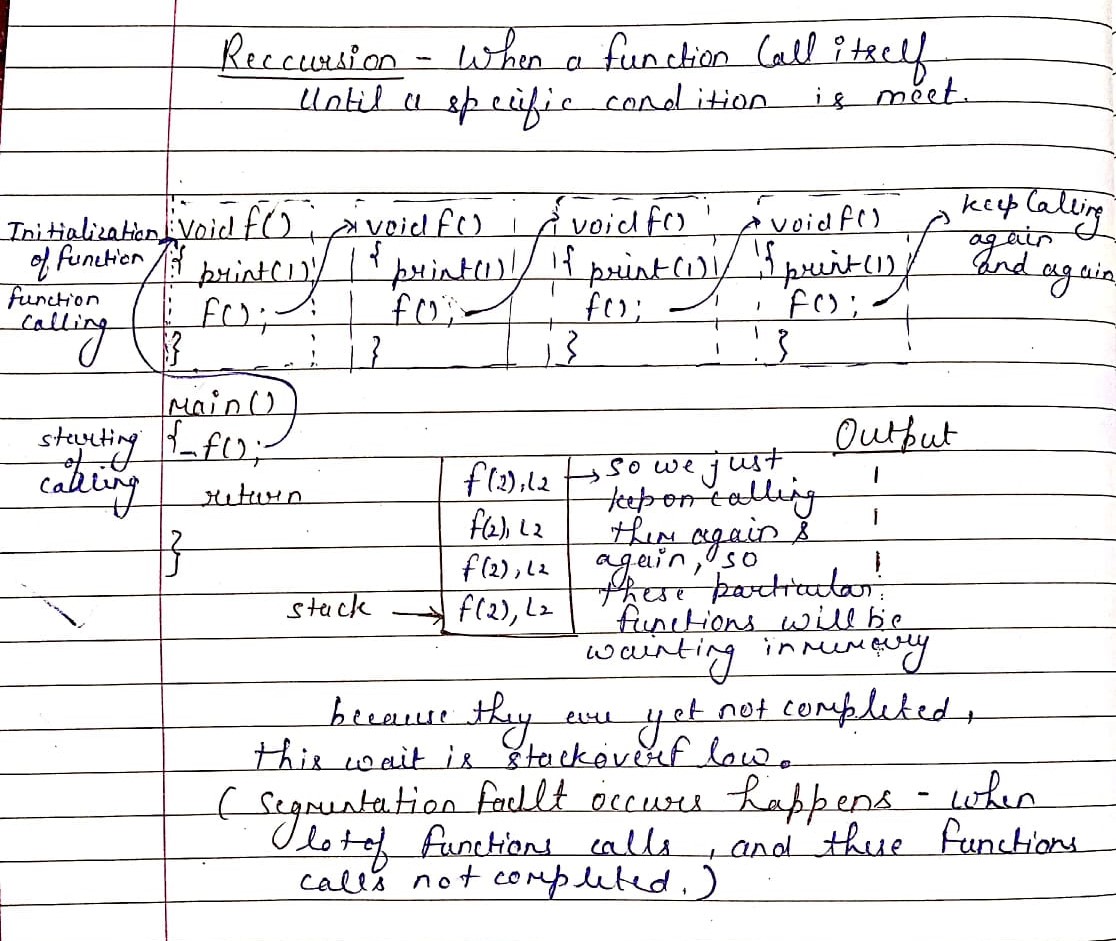
**Using a recursive algorithm, certain problems can be solved quite easily**. Examples of such problems are **Towers of Hanoi (TOH)**, **Inorder / Preorder / Postorder Tree Traversals, DFS of Graph,** etc

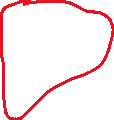
**A recursive function solves a particular problem by calling a copy of itself** and **solving smaller subproblems of the original problems**.



**Need of Recursion - Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.**

**This is Reccursion Code without Given Base Case , This will Print 1 again and again infinity**

****Example –



#include <bits/stdc++.h>

using namespace std;

void print()

{

    cout<<'1'<<endl;

    print();

}

int main()

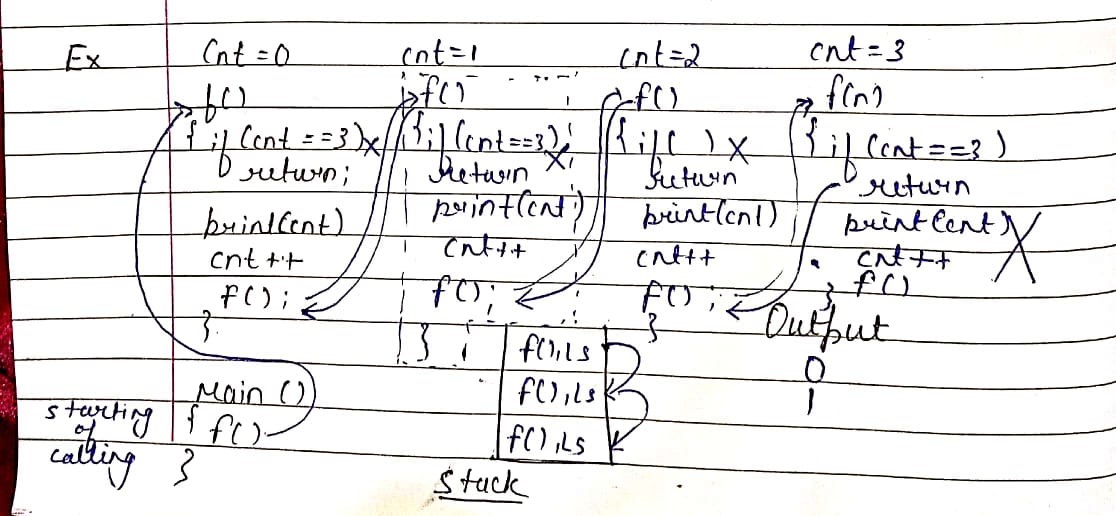
{

    print();

    return 0;

}

**This is Reccursion Code with Given Base Case , This will Print finite no of 1**

Example –

#include <bits/stdc++.h>

using namespace std;

int cnt = 0;  // Define Globally

void print()

{

    if(cnt==4)  // Base Condition

    return ;

    cout<<cnt<<endl;

    cnt++;

    print();   // Function Calling untill Reach The Base Condition

}

int main()

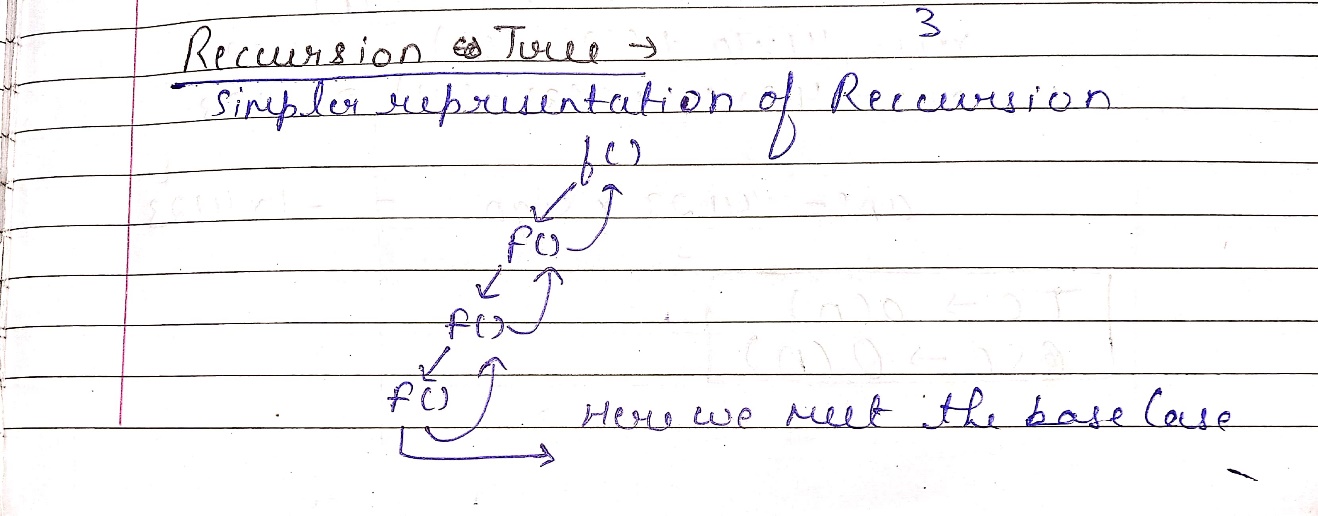
{

    print();

    return 0;

}

**OutPut - 0 1 2 3**

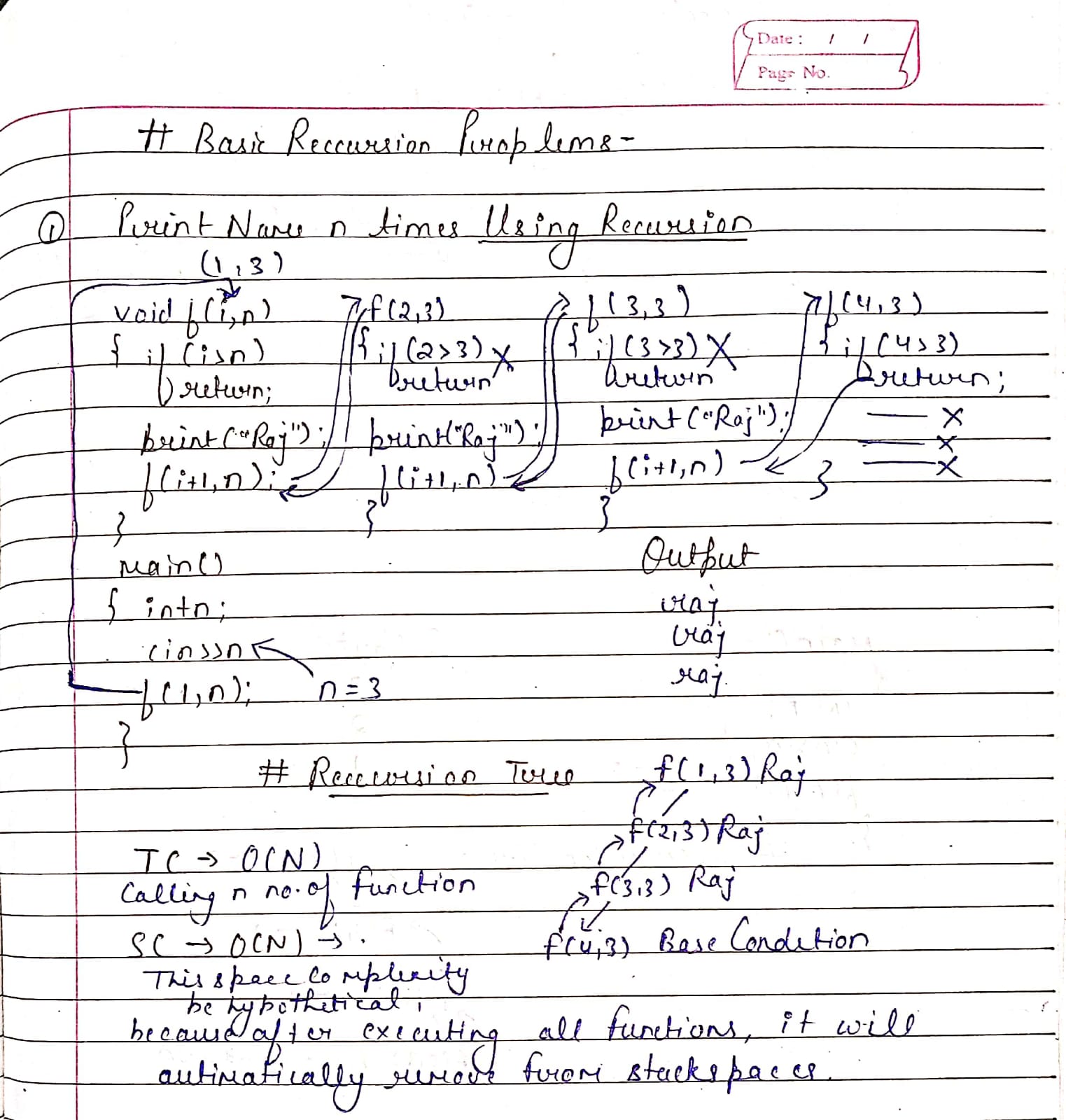
**Reccursion**

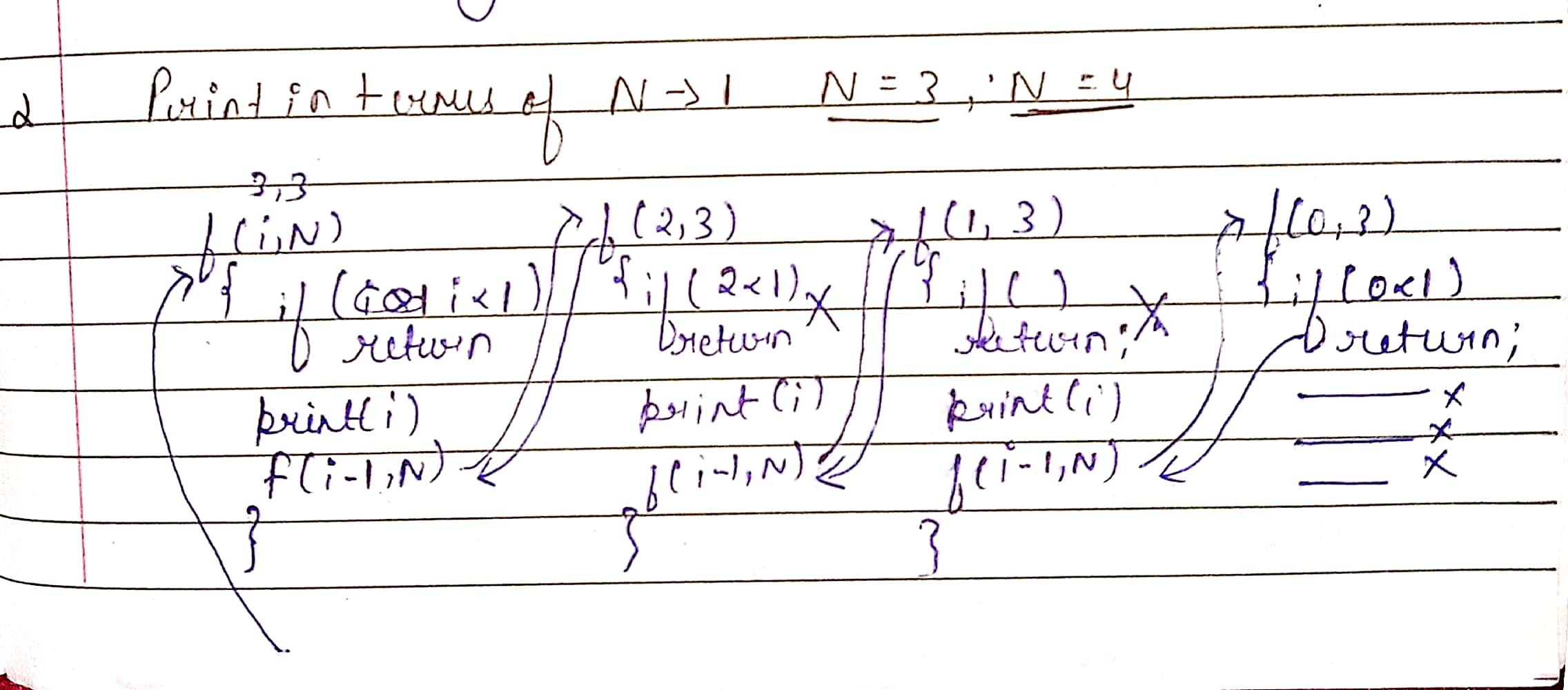
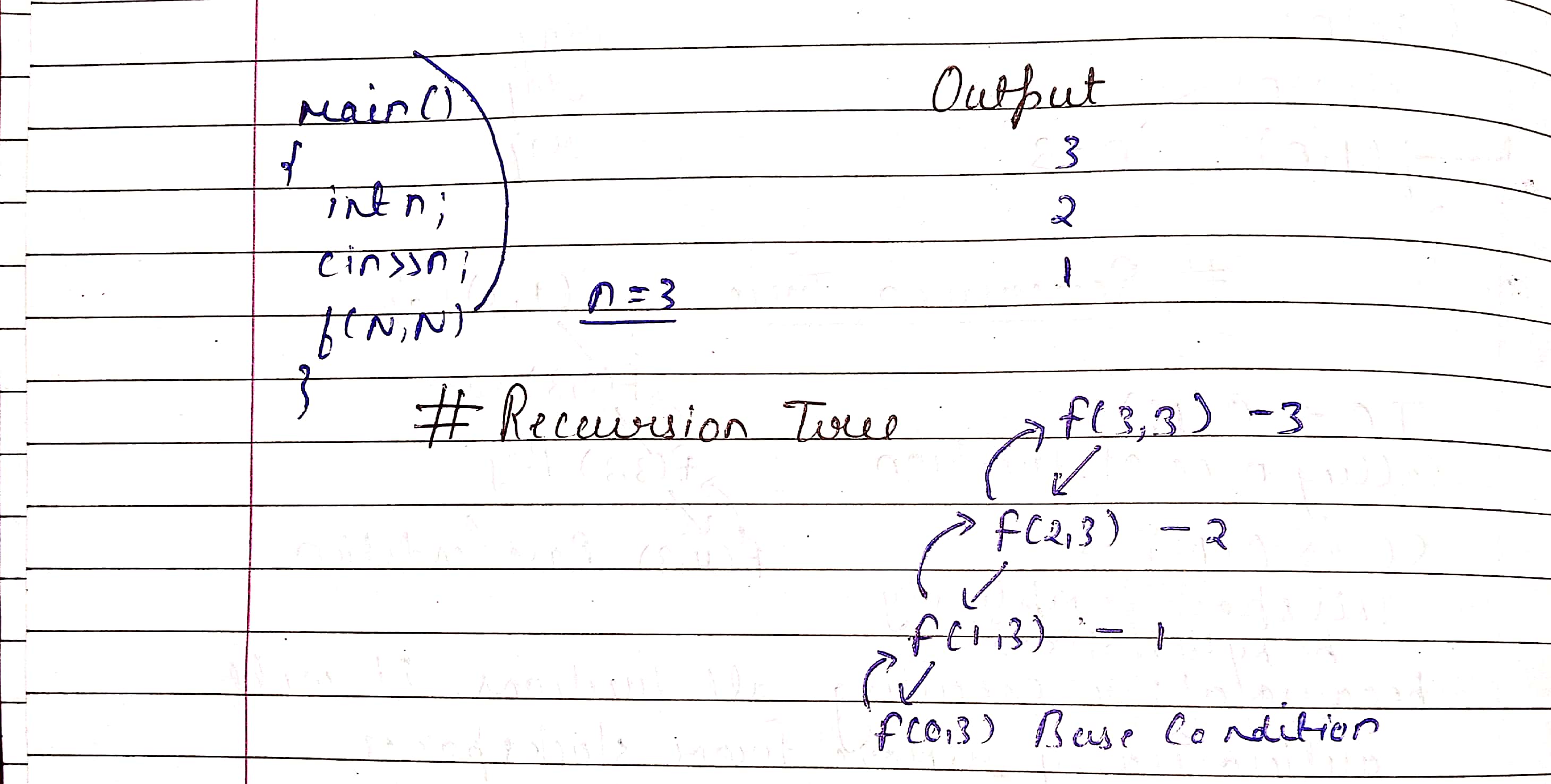
**Base Case**

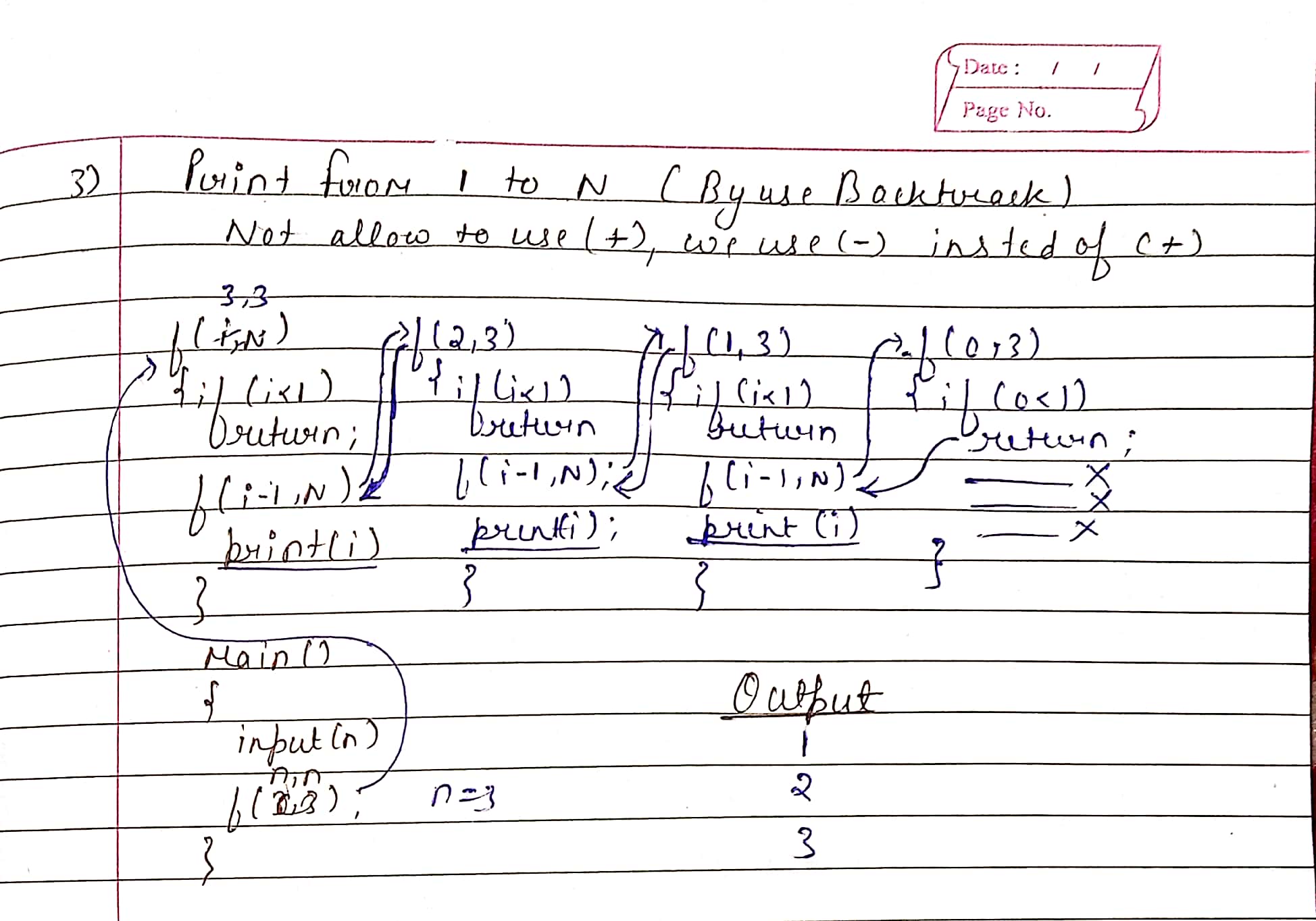
**Stack OverFlow / Stack Space**

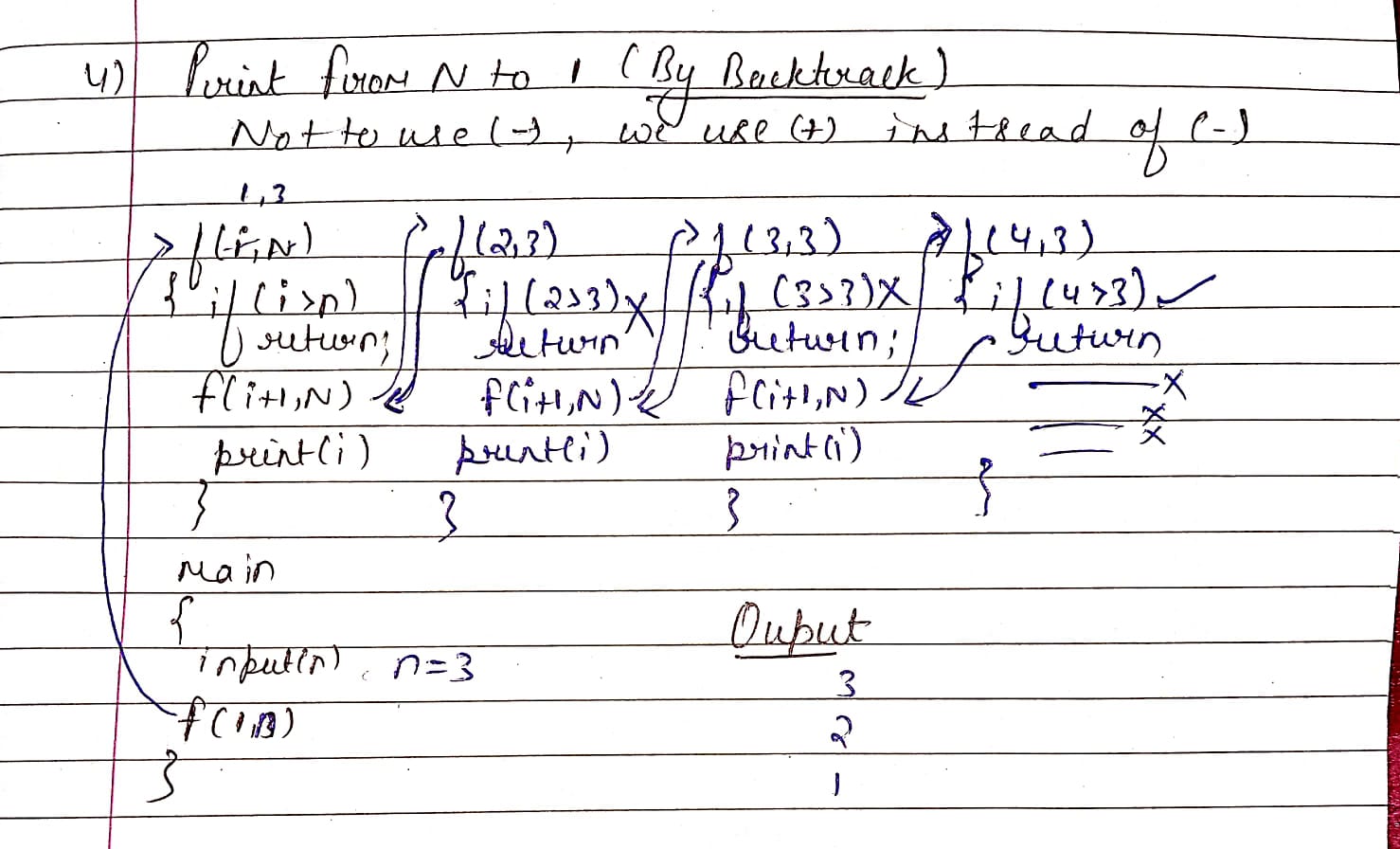
**Reccursion Tree**

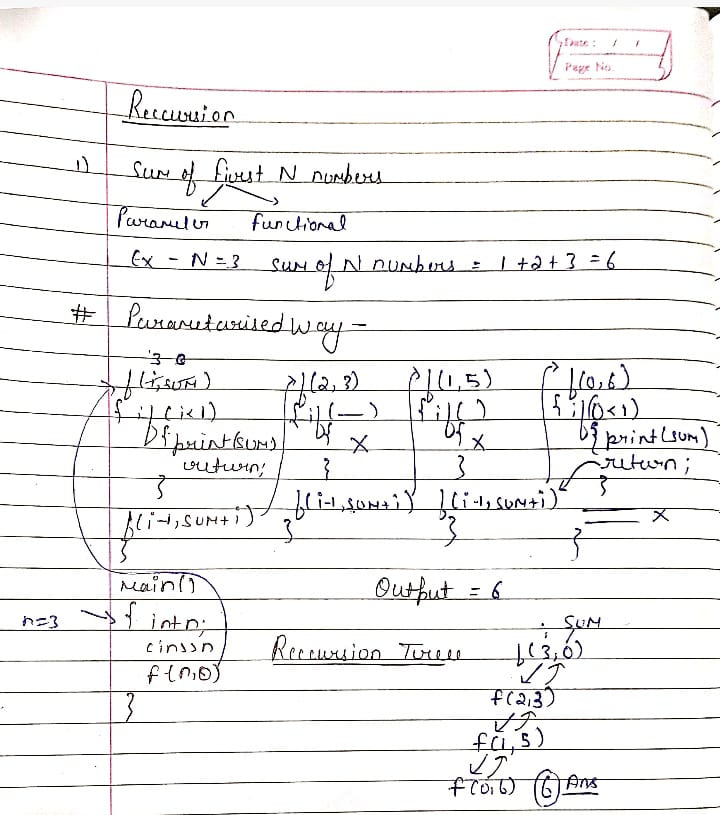
**Question - 1**



**Question – 2**

**Question – 3**

**Question – 4**



Sum **TC – O(N)**

#include<bits/stdc++.h>

using namespace std;

int sum(int N)

{

    if(N==0)return 0;

    return N+sum(N-1);

}

int main() {

    // Write C++ code here

    int n;

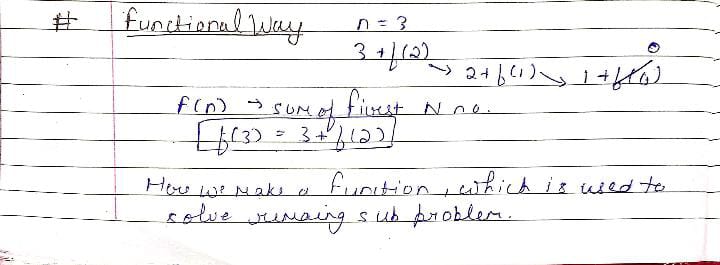
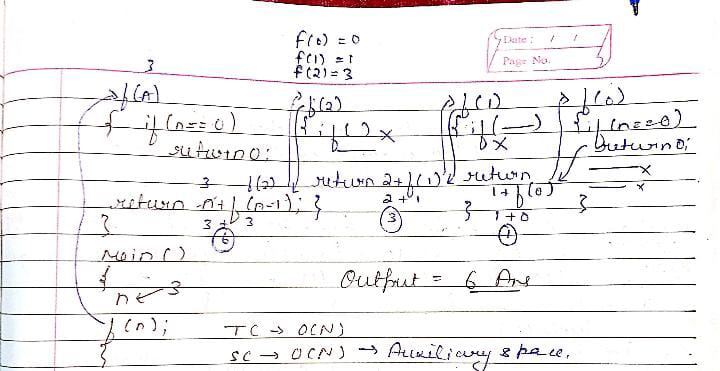
    cout<<"Enter the no : "<<endl;

    cin>>n;

    cout<<sum(n)<<endl;

    return 0;

}

Factorial **TC – O(N)**

#include<bits/stdc++.h>

using namespace std;

int sum(int N)

{

    if(N==0)return 1;

    return N\*sum(N-1);

}

int main() {

    int n;

    cout<<"Enter the no : "<<endl;

    cin>>n;

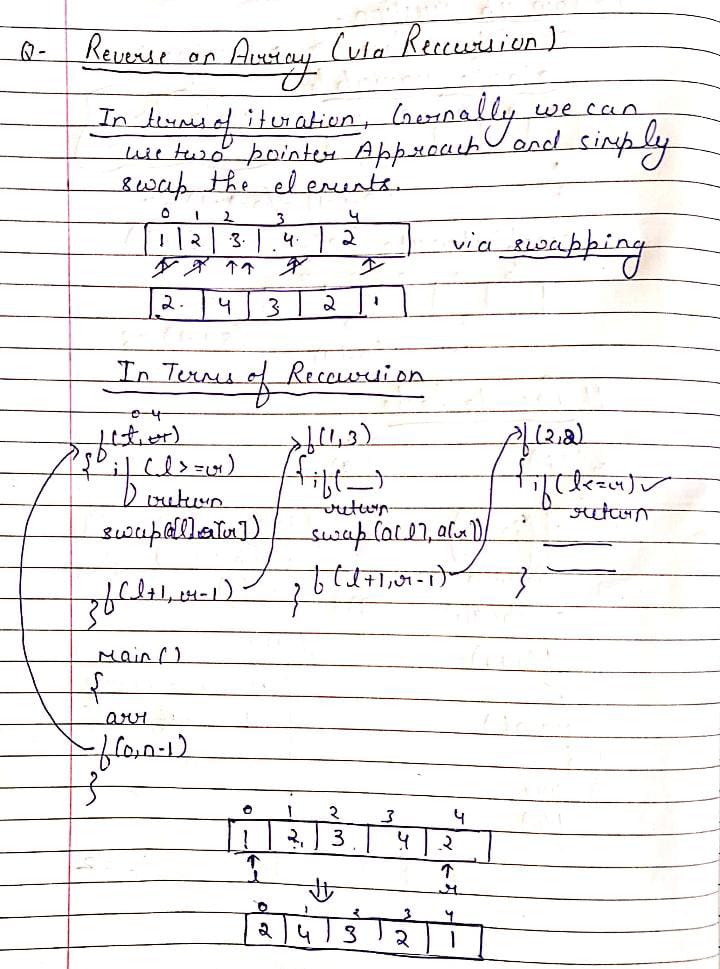
    cout<<sum(n)<<endl;

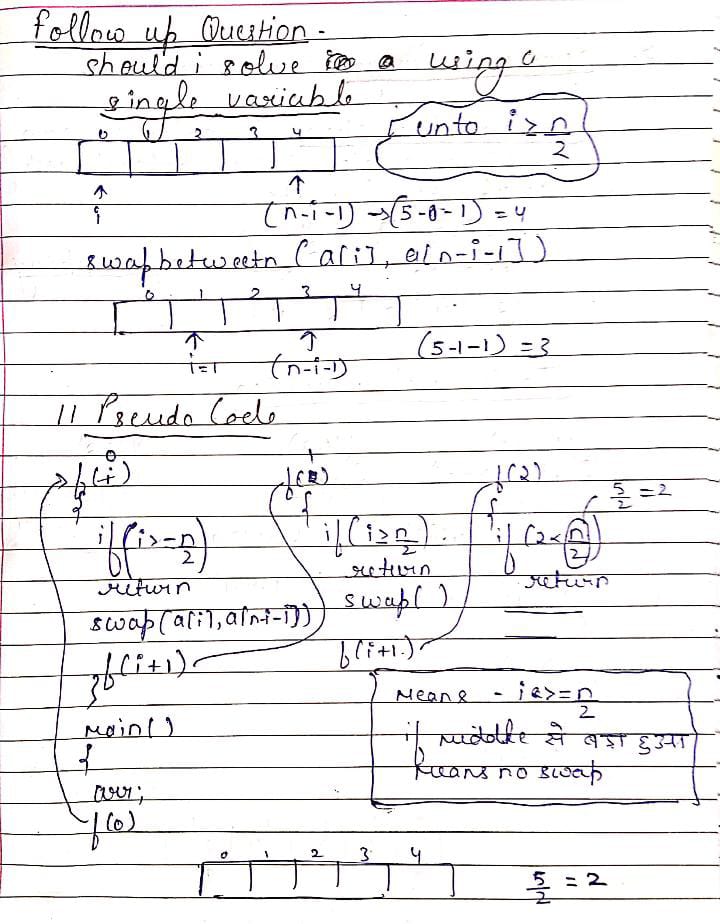
    return 0;

}

**Important Question –**

**Q1 – Reverse An Array**

****



#include <bits/stdc++.h>

using namespace std;

void f(int i,int arr[],int n)

{

    if(i>=n/2)return ;

    swap(arr[i],arr[n-i-1]);

    f(i+1,arr,n);

}

int main()

{

    int arr[5];

    for(int i=0;i<5;i++) cin>>arr[i];

    f(0,arr,5);

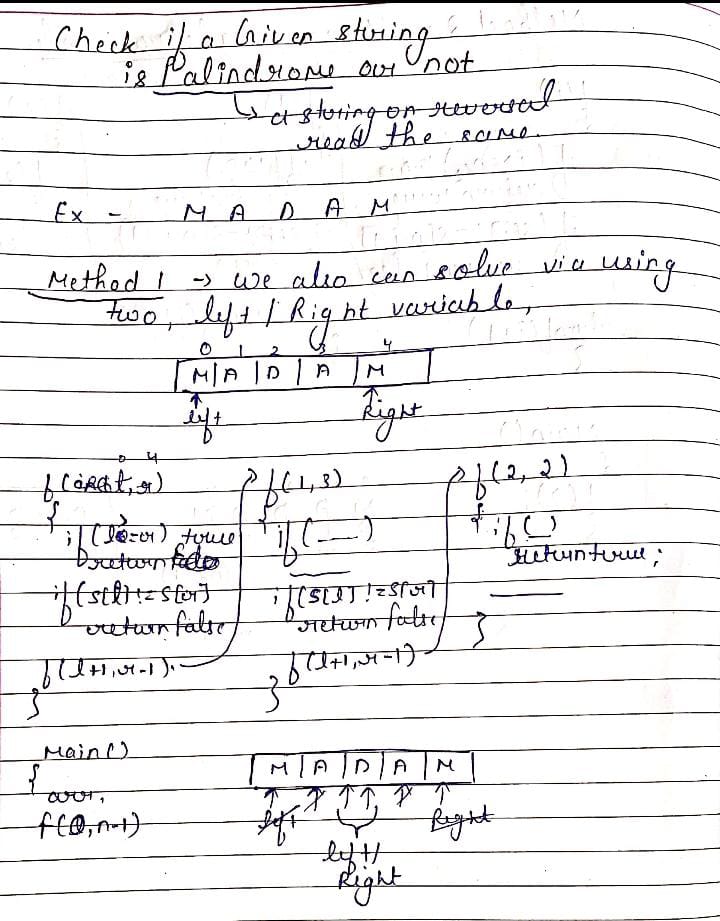
    for(int i=0;i<5;i++)

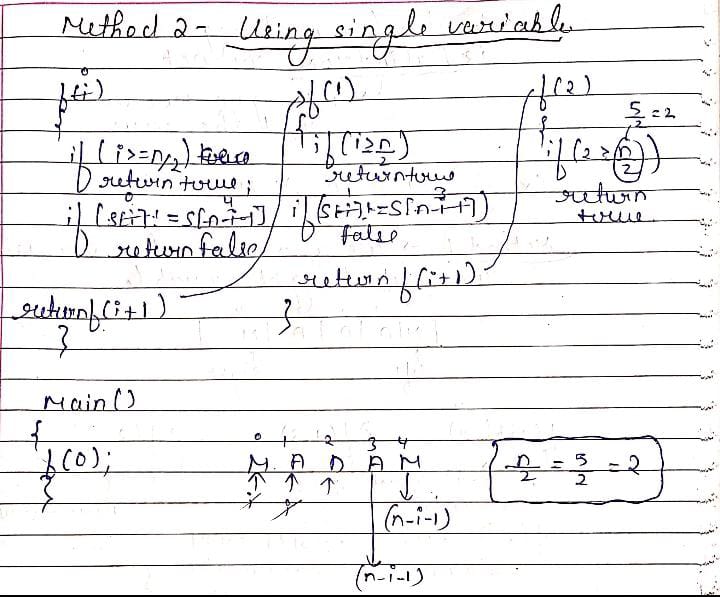
    cout<<arr[i]<<" ";

    return 0;

}

**Q2 – Check If A Given String I s Palindome or Not**

****



**Source Code**

#include<bits/stdc++.h>

using namespace std;

bool f(int i,string &s)

{

    if(i>=s.size()/2)return true;

    if(s[i]!=s[s.size()-i-1])

    return false;

    return f(i+1,s);

}

int main()

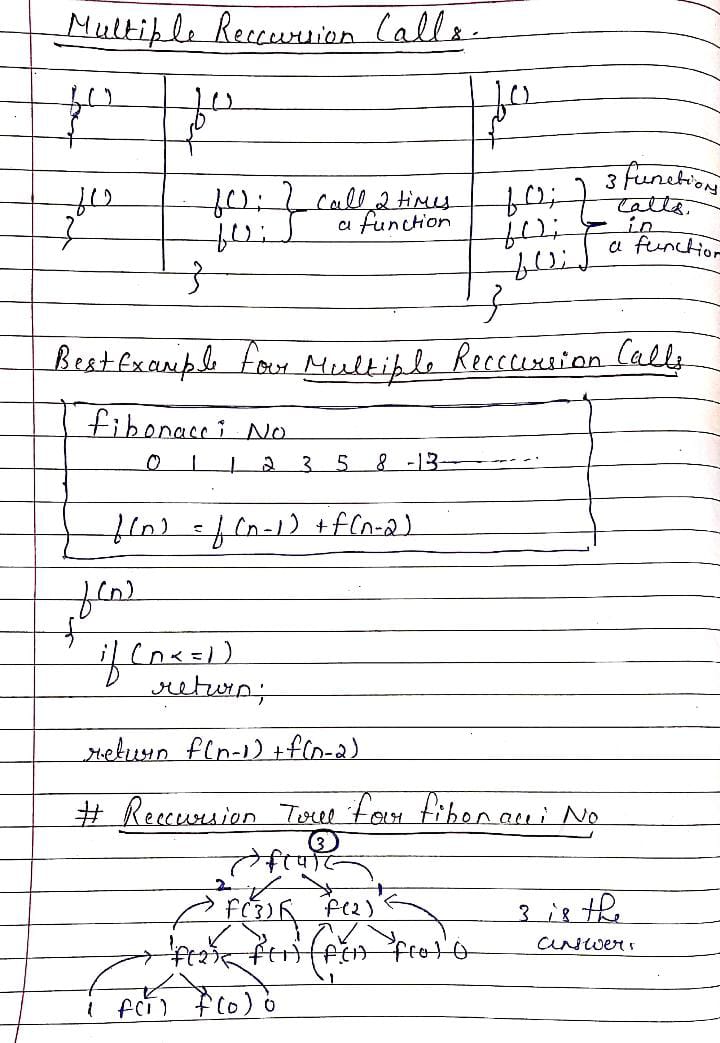
{

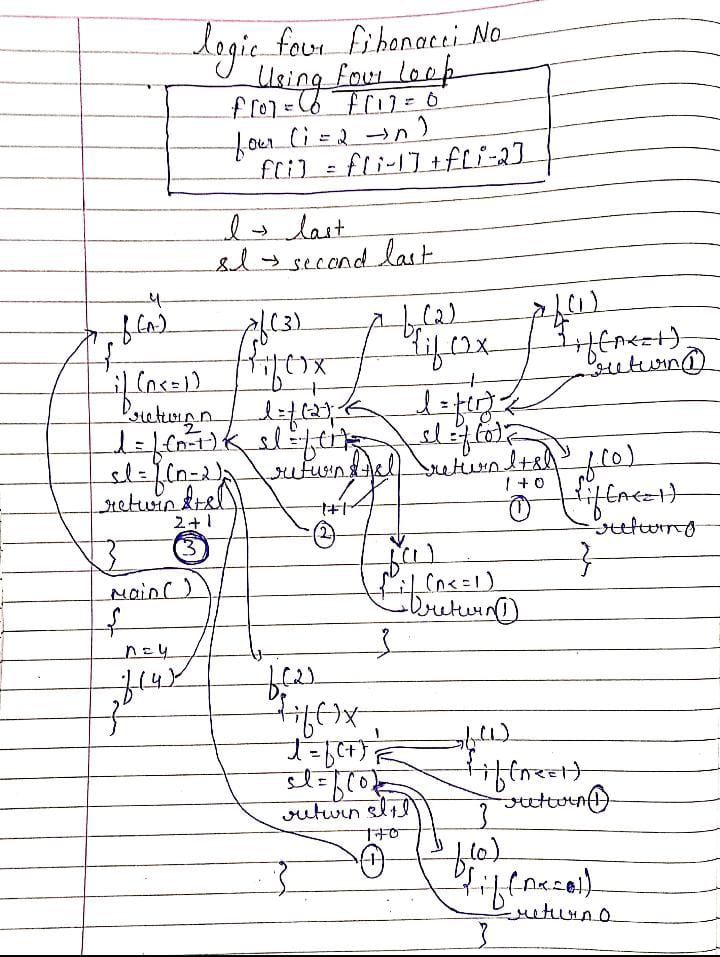
    string s = "MADAM";

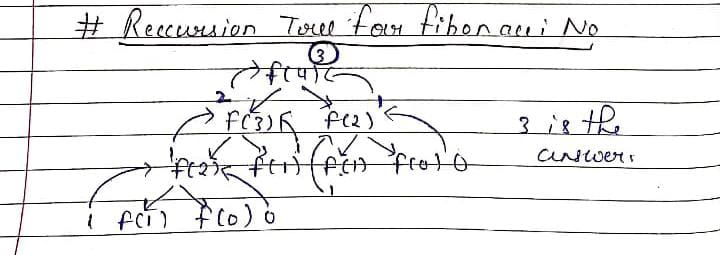
    cout<<f(0,s);

    return 0;

}

**Multiple Reccursion Calls**

****

**Here Every N Calls two recursion Which is (N-1) ,(N-2)**

**That’s Why it take O(2^N) Time Complexity**

#include <bits/stdc++.h>

using namespace std;

int factorial(int n)

{

    if(n<=1)return n;

    int last = factorial(n-1);

    int secondlast = factorial(n-2);

    int fib = last+secondlast;

    return fib;

}

int main(){

    int n;

    cout <<"ENTER VALUE"<<endl;

    cin>>n;

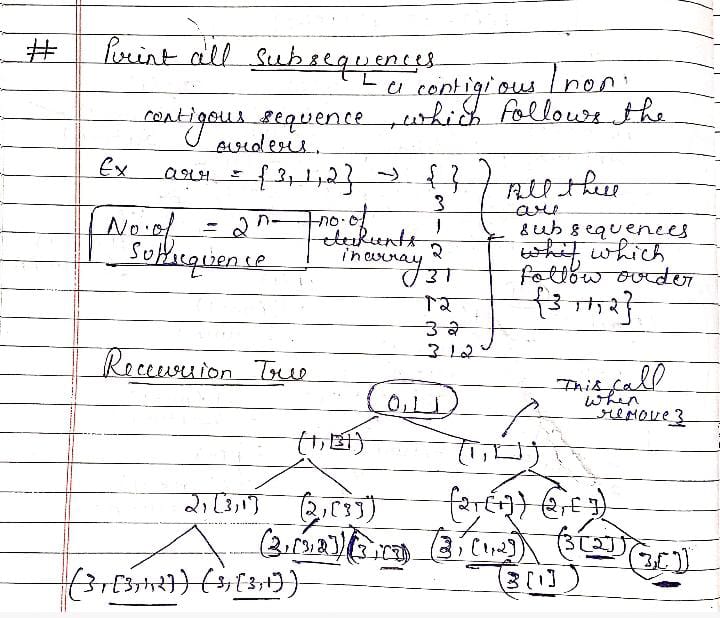
    int fibo = factorial(n);

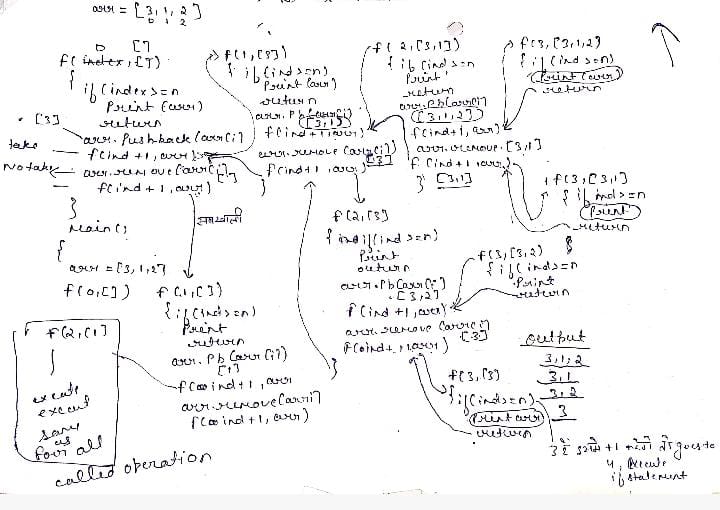
    cout<<"Your result is "<<fibo<<endl;

    return 0;

}

**Print All Subsequences**

****

****



#include <bits/stdc++.h>

using namespace std;

void printSubsequences(int index,vector<int> &ds,int arr[],int n)

{

    if(index==n)

    {

for(auto it:ds)

{

cout <<it;

}

if(ds.size()==0)cout<<"{}";

cout<<endl;

return ;    }

    ds.push\_back(arr[index]);

    printSubsequences(index+1,ds,arr,n);

    ds.pop\_back();

    printSubsequences(index+1,ds,arr,n);

}

int main()

{

    int arr[] = {3,2,1};

    int n = 3;

    vector<int> ds;

    printSubsequences(0,ds,arr,n);

    return 0;

}

**TC - Every Index Having Couple Of Options O(2^N \*N)**

**SC – O(N)**

**OutPut**

**321**

**32**

**31**

**3**

**21**

**2**

**1**

**{}**

**\*\*\* Print All Subsets ( Another Methdod) - In Case Of Array**

class Solution {

public:

    void solve(vector<int>& nums ,vector<int>ans,int index, vector<vector<int>> &result)

    {

        // Agr index hamara traverse krte krte array size se bahar so return

        if(index>=nums.size())

        {

            // but return krne se phle saare , jo bhi ans me daale hai unko print krna pdega

            result.push\_back(ans);

            return ;

        }

        // excluding the element

        solve(nums,ans,index+1,result);

        // including the Element

        int element = nums[index];

        ans.push\_back(element);

        solve(nums,ans,index+1,result);

    }

    vector<vector<int>> subsets(vector<int>& nums)

    {

        vector<vector<int>>result;

        vector<int>ans;

        int index = 0;  // indicates elements of nums array

        solve(nums,ans,index,result);

        return result ;

    }

};

**Input: nums = [1,2,3]**

**Output:([],[1],[1,2],[3],[1,3],[2,3],[1,2,3]]**

**\*\*\* Print All Subsets ( Another Methdod) - In Case Of String**

#include <bits/stdc++.h>

// During Function Calling yaha hume original wo wala vector bhejna pdega jisme final answer store hoga

void solve(string str ,string ans , int index,vector<string>& result)

{

    // Agr index hamara traverse krte krte array size se bahar so return

    if(index>=str.length())

    {

        // means is Question me empty size ka sub array output me nhi dena hai

        if(ans.length()>0)

        {

            result.push\_back(ans);

        }

        return;

    }

    // excluding the element

    solve(str,ans,index+1,result);

    // including the Element

    // string ke case me char lekr kaam krna hai

    char element = str[index];

    ans.push\_back(element);

    solve(str,ans,index+1,result);

}

vector<string> subsequences(string str)

{

    vector<string>result;

    // string ke case me empty string lena hai , or array ke case me ek vector lena hai

    string ans = "";

    int index = 0;

    solve(str,ans,index,result);

    return result ;

}

**\*\*\* Print Sub Array Via USING BIT MANIPULATION**

 // Using Bit Manipulation

       vector<vector<int>>result;

       vector<vector<int>> subsets(vector<int>& nums)

    {

        int n = nums.size();  // ex  - n = 3 -> 2^3  = 8

        // here we can also use 1<<n instead Of pow funtion , Because 1<<n it generally means

        /\*

        1 << n = 1 << 3 = ( multiply 3 times two )  2 \* 2 \* 2

        \*/

        for(int i=0;i<pow(2,n);i++)

        {

            // i = 0

            // i = 1

            // i = 2

            /\* i = 3    suppose 3 - 011  -  means take elemets at first and second index

                        011 in this index starting from right to left

                        But [1,2,3] in this case index starts from left to right

                        means Take

                        So For 3 ( 011 -> [1,2] )

            \*/

            // ..... till i = 7

            vector<int>ans;

            for(int j = 0;j<n;j++)

            {

                 // j loop iterate over its bits

            /\*

            i = 5 -> 1 0 1

                     2 1 0 - index

                                 j

            j = 1 0 1 - then 1 0 1

                        And  1 0 1  - [1,3]

                        First check for j = 0 , so first bit is set so push [1] at vector

                        then j left shift

                                 j

            j = 1 0 1 - then 0 1 0      <<<

                        And  1 0 1  - [1,3]

                        First check for j = 1 , so 2nd bit is not set so not push [2] at vector

                        then j left shift

                             j

            j = 1 0 1 - then 1 0 0

                        And  1 0 1  - [1,3]

                        First check for j = 2 , so 3rd is bit is set so push [3] at vector

            \*/

            // i = 5 -> 101  -> [1,3] -- Her we Goona Find which bit is set or whis is not set , because those bit is set we take it

                if((1<<j)&i)

                  ans.push\_back(nums[j]);

            }

            result.push\_back(ans);

        }

        return result;

    }