

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** .

Eg- $\text{int fun}(\text{int } n)$
 $\{$
 $\quad \text{fun}(n); \rightarrow \text{function call itself again \& again.}$
 $\}$

If the solution of a problem depends on a smaller problem (subproblem) of same type, then we will use recursion.

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.

Eg: Find 2^n .

We can say that $2^n = \underbrace{2 \times 2 \times 2 \times \dots \times 2}_{n \text{ times}}$

$$\Rightarrow 2^n = 2 \times 2^{n-1}$$

If we make a function that gives us 2^n when it's called like:

$\text{fun}(n);$

Then $\text{fun}(n) = 2 * \text{fun}(n-1); \leftarrow \text{Recurrence Relation}$

Eg: Find factorial ($n!$)

We know that $5! = 5 \times 4!$

$$\Rightarrow \text{fact}(5) = 5 \times \text{fact}(4)$$

$$\Rightarrow \text{fact}(n) = n * \text{fact}(n-1)$$

Given that we stop at a condition, example $0! = 1$, this is called base condition. Without this, our function will go bonkers!

$$\text{fact}(3) = 3 \times \text{fact}(2)$$

$$\text{fact}(2) = 2 \times \text{fact}(1)$$

$$\text{fact}(1) = 1 \times \text{fact}(0)$$

$$\begin{aligned}
 \text{fact}(0) &= 0 \times \text{fact}(-1) \\
 \text{fact}(-1) &= -1 \times \text{fact}(-2) \\
 &\vdots \\
 &\vdots \\
 &\infty
 \end{aligned}$$

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.

Recursion - When a function call itself until a specific condition is met.

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

Example -

Initialization of function	Function calling	void f()	void f()	void f()	void f()	keep calling again and again
		{ print(1); f(); }	{ print(1); f(); }	{ print(1); f(); }	{ print(1); f(); }	
starting of calling	main() { f(); return }					
			f(2), L2	f(2), L2	f(2), L2	so we just keep on calling then again & again, so these particular functions will be waiting in memory
			stack → f(2), L2			
						Output 1 1 1 1

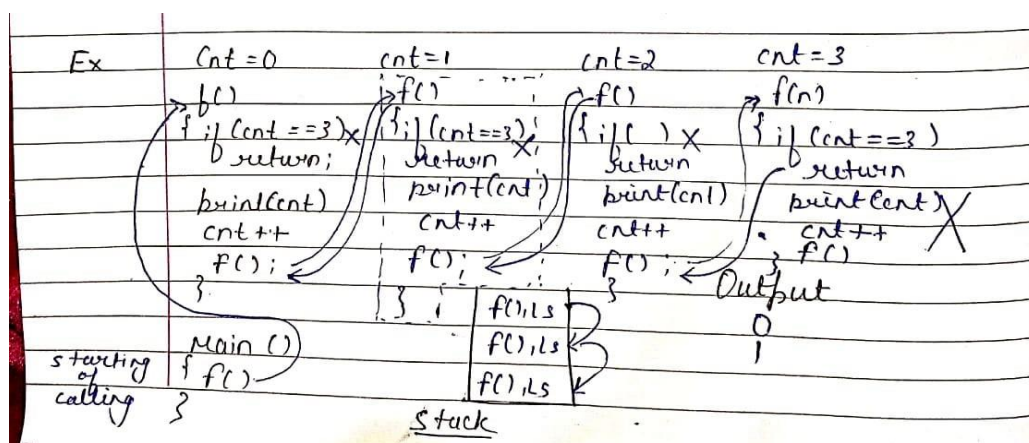
because they are yet not completed, this wait is stackoverflow.

(segmentation fault occurs happens - when lot of functions calls, and these functions calls not completed.)

```
#include <bits/stdc++.h>
using namespace std;
void print()
{
    cout<<'1'<<endl;
    print();
}
int main()
{
    print();
    return 0;
}
```

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

Example –



```
#include <bits/stdc++.h>
using namespace std;

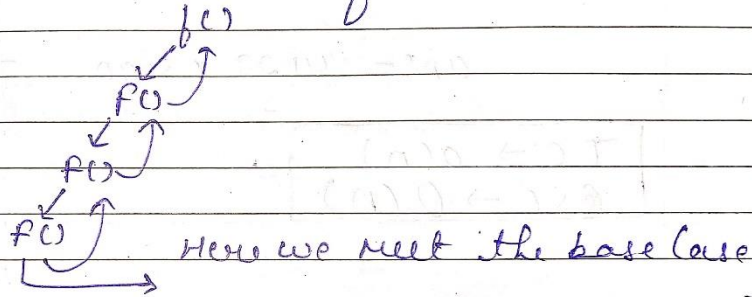
int cnt = 0; // Define Globally
void print()
{
    if(cnt==4) // Base Condition
        return ;
    cout<<cnt<<endl;
    cnt++;
    print(); // Function Calling untill Reach The Base Condition
}
int main()
{
    print();
    return 0;
}
```

OutPut - 0 1 2 3

Recursion Tree →

3

Simpler representation of Recursion



Recursion

Base Case

Stack Overflow / Stack Space

Recursion Tree

Question - 1

Date: / /
Page No.

Basic Recursion Problems -

① Print Name n times Using Recursion

```

void f(i, n)
{
    if (i > n)
        return;
    print("Raj");
    f(i+1, n);
}

main()
{
    int n;
    cin >> n;
    f(1, n);
}
    
```

$n=3$

Output
Raj
Raj
Raj

Recursion Tree

TC → $O(N)$

Calling n no. of function

SC → $O(N)$ →

This space complexity
be hypothetical,

because after executing all functions, it will
automatically remove from stack space.

$f(1, 3)$ Raj
 $f(2, 3)$ Raj
 $f(3, 3)$ Raj
 $f(4, 3)$ Base Condition

Question - 2

Print in terms of $N \rightarrow 1$ $N=3$, $N=4$

```

3,3
f(i,N)
{
  if (i < 1)
    return;
  print(i);
  f(i-1, N);
}

2,3
f(i,N)
{
  if (i < 1)
    return;
  print(i);
  f(i-1, N);
}

1,3
f(i,N)
{
  if (i < 1)
    return;
  print(i);
  f(i-1, N);
}

0,3
f(i,N)
{
  if (i < 1)
    return;
  // X
  // X
  // X
}
  
```

main()

```

{
  int n;
  cin >> n;
  f(n, n);
}
  
```

$n=3$

Recursion Tree

```

      f(3,3) - 3
      /  \
    f(2,3) - 2
    /  \
  f(1,3) - 1
  /  \
f(0,3) Base Condition
  
```

Output

```

3
2
1
  
```

Question - 3

3) Print from 1 to N (By use Backtrack)

Not allow to use (+), we use (-) insted of (+)

```

3,3
f(i,N)
{
  if (i < 1)
    return;
  f(i-1, N);
  print(i);
}

2,3
f(i,N)
{
  if (i < 1)
    return;
  f(i-1, N);
  print(i);
}

1,3
f(i,N)
{
  if (i < 1)
    return;
  f(i-1, N);
  print(i);
}

0,3
f(i,N)
{
  if (i < 1)
    return;
  // X
  // X
}
  
```

main()

```

{
  input(n);
  f(n, n);
}
  
```

$n=3$

Output

```

1
2
3
  
```

Question - 4

4) Print from N to 1 (By Backtrack)

Not to use (-), we use (+) instead of (-)

1, 3

```

    f(i, N)
    {
        if (i > N)
            return;
        f(i+1, N)
        print(i)
    }

    f(2, 3)
    {
        if (2 > 3) X
        return
        f(3, 3)
        {
            if (3 > 3) X
            return
            f(4, 3)
            {
                if (4 > 3) ✓
                return
            }
        }
        print(2)
    }
    print(1)
}
    
```

main

```

{
    input(n) n=3
    f(1, 3)
}
    
```

Output

3
2
1

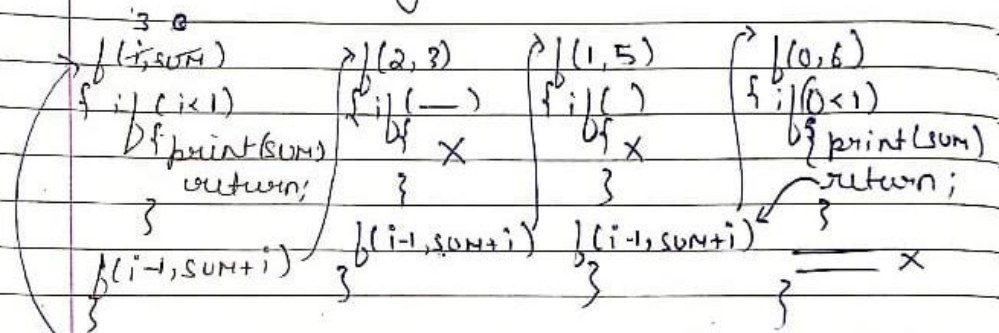
Recursion

1) Sum of first N numbers

Parameter Functional

Ex - $N=3$ sum of N numbers = $1+2+3=6$

Parametrised Way -

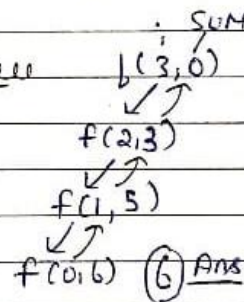


main()

```
int n;  
cin >> n;  
f(n, 0);  
}
```

Output = 6

Recursion Tree



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)

Functional Way $n = 3$
 $3 + f(2) \rightarrow 2 + f(1) \rightarrow 1 + f(0)$
 $f(n) \rightarrow$ sum of first N no.
 $f(3) = 3 + f(2)$
How we make a function, which is used to solve remaining sub problem.

$f(0) = 0$
 $f(1) = 1$
 $f(2) = 3$

Date: / /
Page No.

3
f(3)
if (n==0)
return 0;
return n + f(n-1);
Main()
n = 3
f(n);

f(2)
if (n==0)
return 0;
return 2 + f(1);
f(1)
if (n==0)
return 0;
return 1 + f(0);
f(0)
if (n==0)
return 0;

3
2+1
1+0
0

Output = 6 Ans

TC $\rightarrow O(N)$
SC $\rightarrow O(N) \rightarrow$ Auxiliary space.

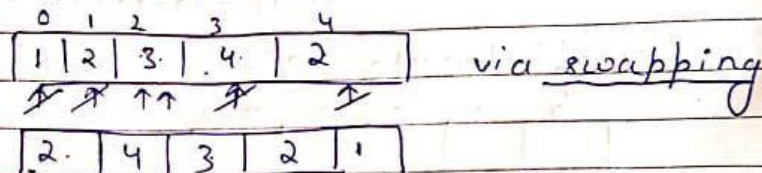
```
#include<bits/stdc++.h>
using namespace std;
int sum(int N)
{
    if(N==0)return 1;
    return N*sum(N-1);
}
int main() {
    int n;
    cout<<"Enter the no : "<<endl;
    cin>>n;
    cout<<sum(n)<<endl;
    return 0;
}
```


Important Question –

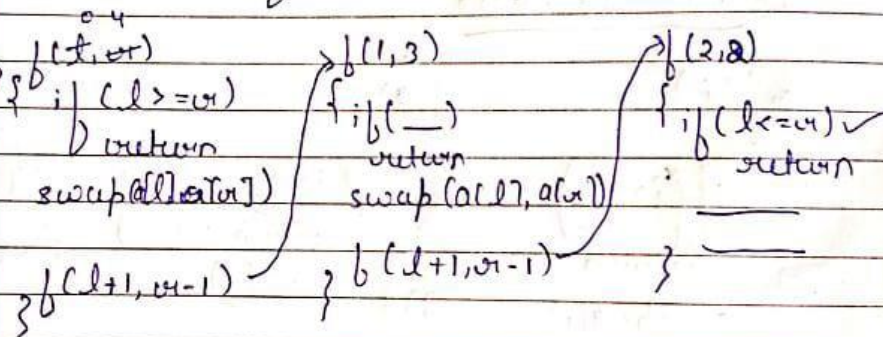
Q1 – Reverse An Array

Q- Reverse an Array (via Recursion)

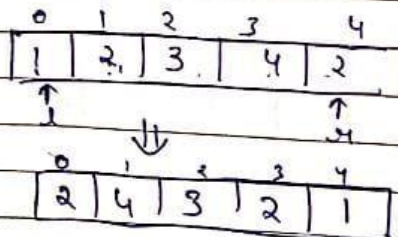
In terms of iteration, normally we can use two pointer Approach and simply swap the elements.



In Terms of Recursion

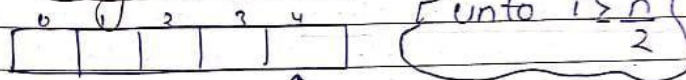


```
main()
{
    arr
    f(0, n-1)
}
```



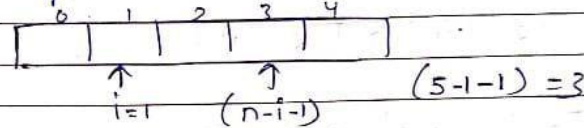
Follow up Question -

should i solve it using a single variable

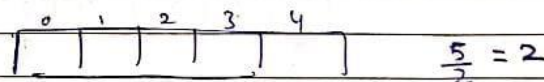
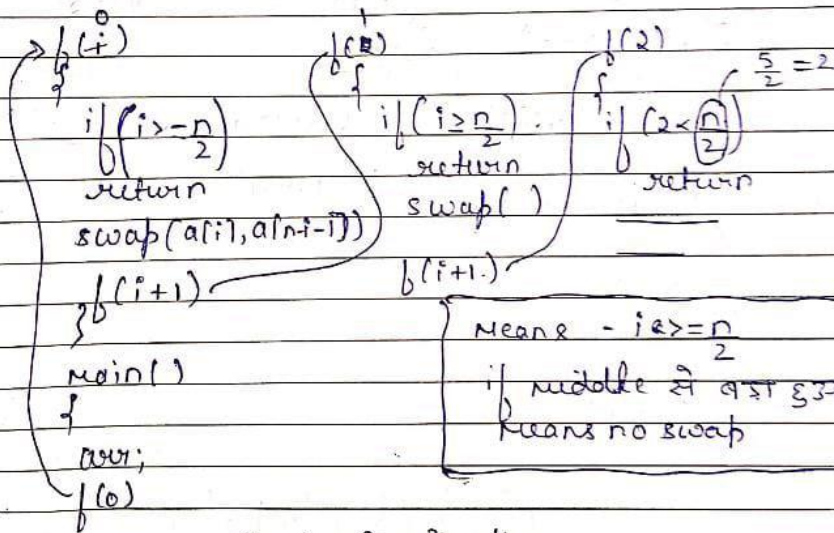


$(n-i-1) \rightarrow (5-0-1) = 4$

swap between $(a[i], a[n-i-1])$



11 Pseudo Code



```

#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 Is Palindrome or Not

Check if a given string
is palindrome or not

↳ a string on reversal
reads the same.

Ex - M A D A M

Method 1 → we also can solve via using
two, left / Right variable,

0	1	2	3	4
M	A	D	A	M

↑ ↑
left right

```

    b(left, n)
    {
        if (left == n) return true;
        if (s[left] != s[n-1]) return false;
        b(left+1, n-1);
    }

    b(1, 3)
    {
        if (1 == 3) return true;
        if (s[1] != s[3]) return false;
        b(2, 2);
    }

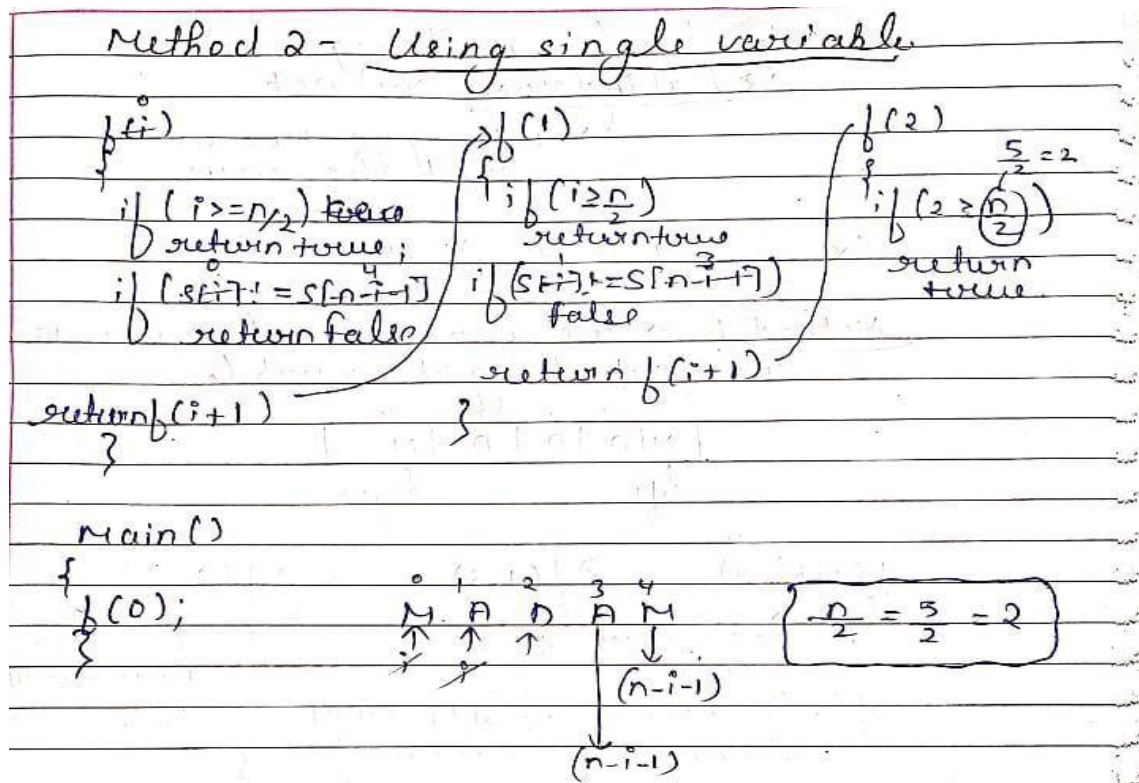
    b(2, 2)
    {
        if (2 == 2) return true;
    }
    
```

```

    main()
    {
        int n = s.length();
        f(0, n-1);
    }
    
```

0	1	2	3	4
M	A	D	A	M

↑ ↑ ↑ ↑ ↑
left left+1 left+2 left+3 right



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 Recursion Calls

Multiple Recursion Calls

<code>f()</code>	<code>f()</code>	<code>f()</code>
<code>{</code>	<code>{</code>	<code>{</code>
<code>f()</code>	<code>f();</code>	<code>f();</code>
<code>}</code>	<code>f();</code>	<code>f();</code>
	<code>}</code>	<code>}</code>

call 2 times a function

3 functions calls in a function

Best Example for Multiple Recursion Calls

Fibonacci No

0 1 1 2 3 5 8 13

$f(n) = f(n-1) + f(n-2)$

```

f(n)
{
  if (n <= 1)
    return;

  return f(n-1) + f(n-2);
}
    
```

Logic for Fibonacci No Using For Loop

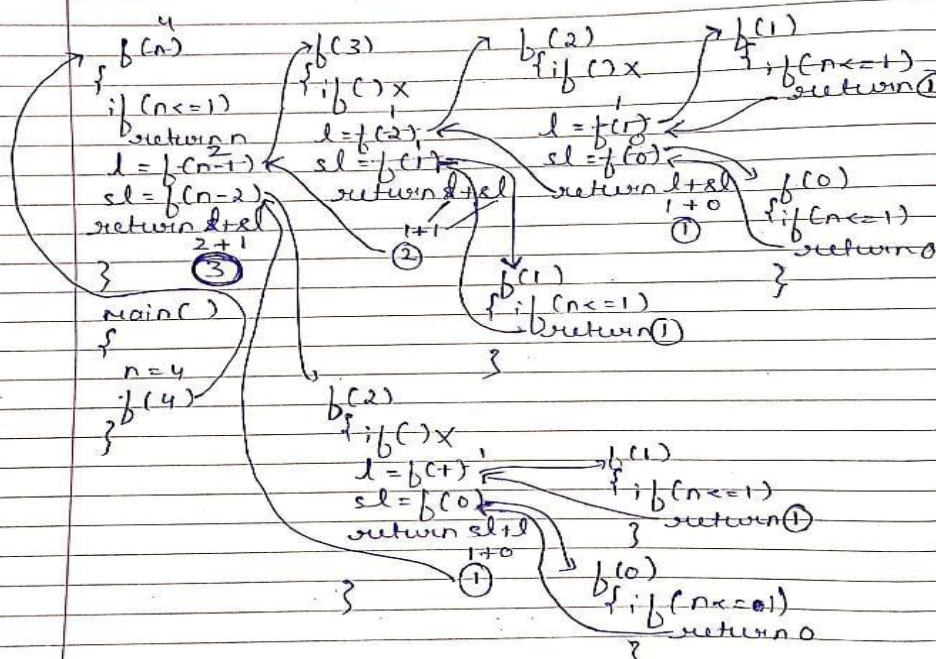
$f(0) = 0$ $f(1) = 0$

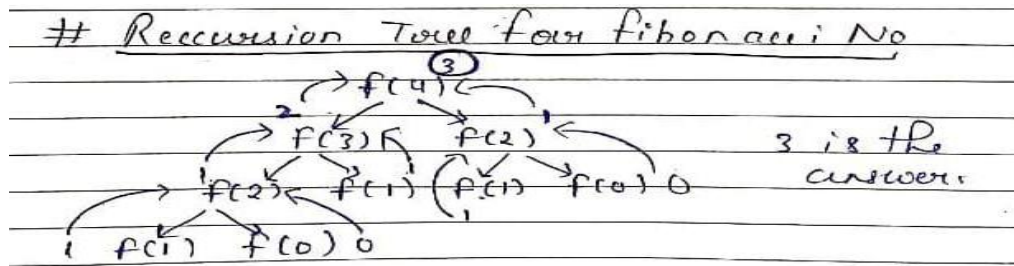
for $(i = 2 \rightarrow n)$

$f(i) = f(i-1) + f(i-2)$

$l \rightarrow$ last

$sl \rightarrow$ second last





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

Print all subsequences

contiguous / non-contiguous sequence, which follows the order.

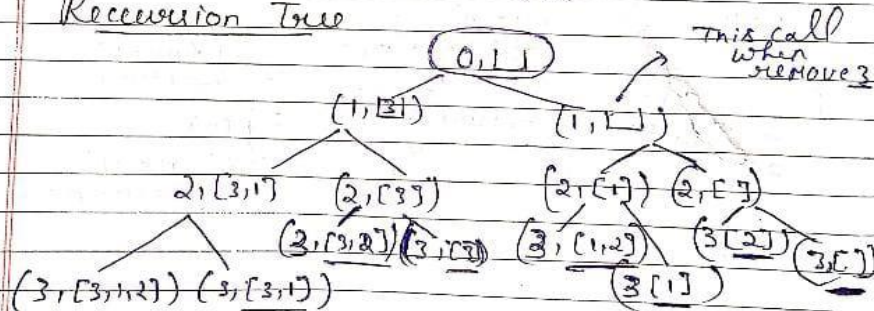
Ex array = {3, 1, 2} → { }

No. of Subsequence = 2^n

no. of elements in array

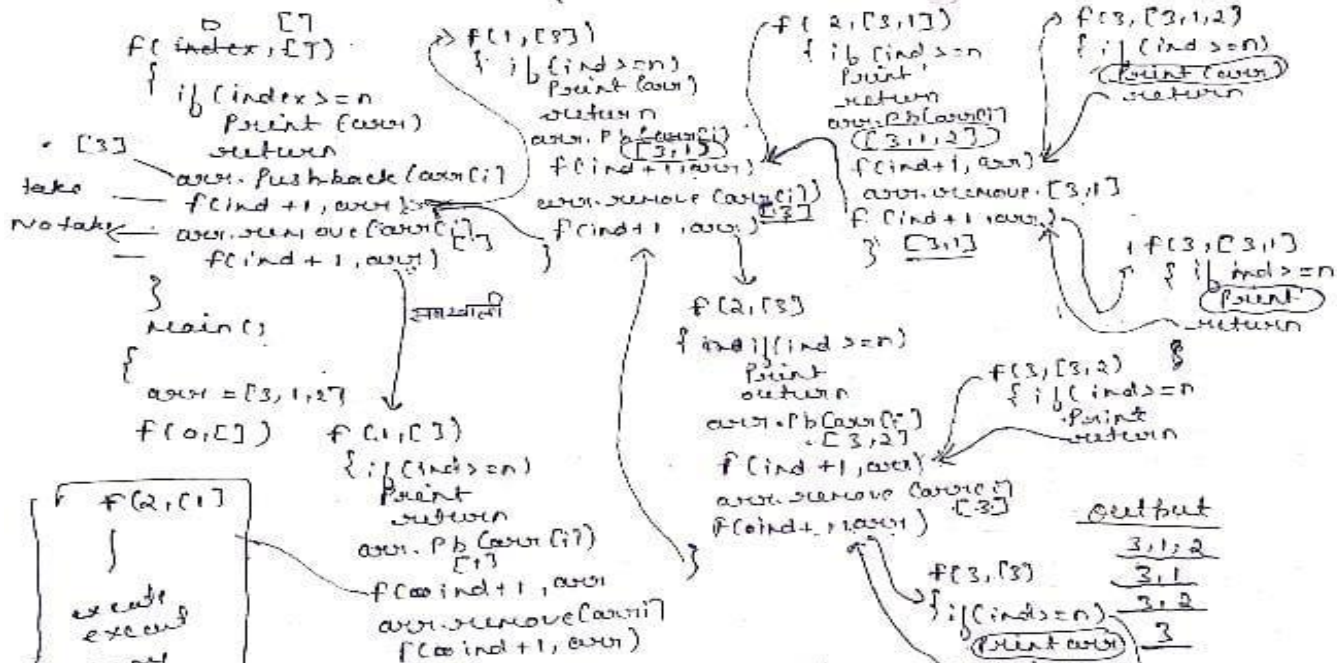
all these are subsequences which follow order {3, 1, 2}

Recursion Tree



this call when remove 3

array = [3, 1, 2]



output

3, 1, 2
3, 1
3, 2
3

3 * 2 * 2 + 1 + 2 * 2 * 2 goes to 4, return if statement

```

#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 Method) - 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 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;
}

```