RECURSION

Structure of Recursion

```
—— fun(——)
{
    Base Cases
    —— —— ——
    Recursive Call (i.e call to func)
    with atleast one change in parameter
    —— —— ——
}
```

Application of Recursion:

1. Many Algorithm techniques are based on Recursion
   - DP
   - Back tracking
   - Divide & Conquer (Binary Search, Quick Sort, Merge Sort,

2. Many problems inherently recursive → easy to write Recursive code than iterative co
   - Tower of Hanoi
   - DFS based traversals (DFS of Graph and Inorder/Pre/Postorder traversal of tree)

Disadvantage of Recursion

- Iterative code causes less overhead.
- Function call overhead.

→ Decimal to binary conversion

```
void fun(int n)
{
    if(n==0)
        return;
    fun(n/2);
    pf(n%2);
}
```

This prints binary of a no. in reverse order so first store the output then display it by doing reverse pf.

I→ Print n to 1 using Recursion

```
void printNo(int n)
{
    if(n==0)
        return;
    cout<<n<<" ";
    printNo(n-1);
}
```

If n=5
5 4 3 2 1

** See how to write simply we write return.

$$T(n) = \underbrace{T(n-1)}_{\text{call for n-1 times}} + \underbrace{\Theta(1)}_{\text{function does const. amount of work}}$$

Time Complexity = $\Theta(n)$
Auxillary space = $\Theta(n)$
↓
we will have (n+1) function call in function call stack (say if n=4 then)

4 → 3 → 2 → 1 → 0

n+1

Pre, In Order Traversal are Tail Recursive
Post is not

II Print ~~N~~ 1 to n

```
void print1toN(int n)
{
    if(n==0)
        return;
    print1toN(n-1);
    cout<<n<<" ";
}
```

If n=5
1 2 3 4 5

Time Complexity $\theta(n)$
$T(n) = T(n-1) + \theta(1)$
Auxillary space → ~~$\theta(n)$~~ $O(n)$

Tail Recursion

- A function is said to be tail recursive if it has nothing more to do when the child function is finished (say I eg)
- Example I take less time on modern compilers b/c it is tail recursive.

```
void fun(int n)
{
    if (n==0)
        return;
    print(n);
    fun(n-1);
}
```

⟹ modern compiler

```
void fun(int n)
{ start:
    if(n==0)
        return;
    print (n);
    fun(n-1);  → n=n-1
                 goto start;
}
```

replaced by this

- recursion is removed completly
- Auxillary space O(1)
- Tail call elimination: the changes which you are changing parameter, adding goto, start called.

Tail Recursion is the reason why Quick Sort is faster than Merge Sort

initially k=1

```
* void f(int n)                       void f(int n, int k)
  {                                   {
     if(n==0)                            if(n==0)
       return;        ==>                  return;
     f(n-1);       changing to           print(k);
     print(n);     Tail Recursive        f(n-1, k+1);
  }                                   }
```

```
fun(3)                                f(3,1)
 ├→ fun(2)                             ├ 1
 │   ├ fun(1)                          ├ f(2,2)
 │   │   ├ fun(0)                      │   ├ 2
 │   │   └→ 1                          │   ├ f(1,3)
 │   └→ 2                              │   │   ├ 3
 └→ 3                                  │   │   └ f(0,4)
```

```
* int f(int n)                        int f(int n, int k)
  {                                   {
     if(n==0 || n==1)      ==>           if(n==0 || n==1)
       return 1;        change to          return k;
     return n* f(n-1);  tail recursion   return f(n-1, k*n);
  }                                   }
```

- Here Parent call not finish immediatly after the child call, here multiplication is done afterward

```
f(3)                                  f(3,1)
 └→ 3 * f(2)                            └→ f(2,1*3)
        └→ 2 * f(1)          return 6      └→ fun(1, 1*3*2)
                ↓                                  return 6
      it is first calculated
      then returned
```

0, 1, 1, 2, 3, 5, 8, 13.

→ n^{th} fibonacci Number where n>=0 :

```
int fib (int n)
{
    if(n==0) return 0;        ), And if(n <= 1)
    if (n==1) return 1;                  return n;

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

→ Sum of natural no. using Recursion

```
int getSum(int n)
{
    if(n==0)
        return 0;
    return n+ getSum(n-1);
}
```

TC → O(n)

AS → O(n) (n+1 function call in the function call stack

→ Pallindrome check

passing by reference
- to avoid string copy
- to optimize

```
bool isPalin(string &s, int l, int r)
{
    if( l>=r)
        return true;
    return( s[l]== s[r] && isPalin(s, l+1, r-1));
}
```

if this wrong no further check

TC → O(n)

T(n) = T(n-2) + O(1)

AS → O(n)

Recursion On Subsequences :

(I) To print subsequence :

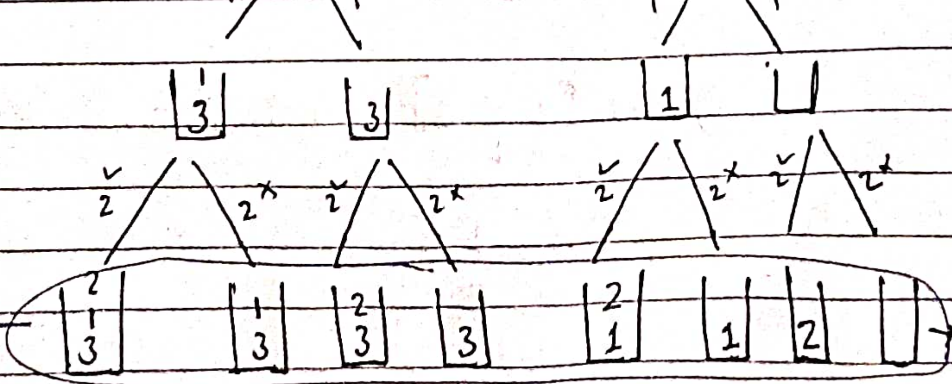
initially 0

```
void fun(vector<int> v1, vector<int> v2, i, n)
{
    if(i==n)
    {   for(auto it : v2)
            cout<<it<<" ";
    }
    cout<<endl;
    return;
    }

    // take particular index from v1 into the subsequence v2
    v2.push_back(v1[i]);      |
    fun(v1, v2, i+1, n);      | → pick condition
    v2.pop_back();            |
    fun(v1, v2, i+1, n);      | → not pick condition

}
```

eg: v1 [3 1 2]
0 1 2



this all is printed b/z i==3 in next step ∴ go to base case

in next step i==n; base case go, print v2.

→ empty v2 (empty sub sequence) ∅

(II) Printing Subsequences whose Sum is k :

```
          i      sum
f(v1, v2, 0, n, k, 0);  → main function call

void f(vector<int> v1, vector<int> v2, i, n, k, sum)
{
    if(i==n)
    {
        if(sum==k)
        {
            for(auto it: v2)
                cout<<it<<" ";
            cout<<endl;
        }
        return;  → function stop here no further calling
                   return to caller function.
    }

    v2.push_back(~~v1, v2, i~~ v1[i])
    sum += v1[i];
    f(v1, v2, i+1, n, k, sum);
    v2.pop_back();
    sum -= v1[i];
    f(v1, v2, i+1, n, k, sum);
}
```

2 index पर

eg: v1 [1 2 1] n=3, k=2
        0 1 2

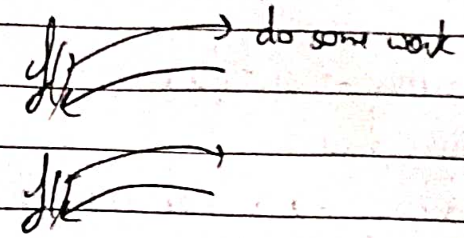
o/p 1 1
     2

Take / Not take Index

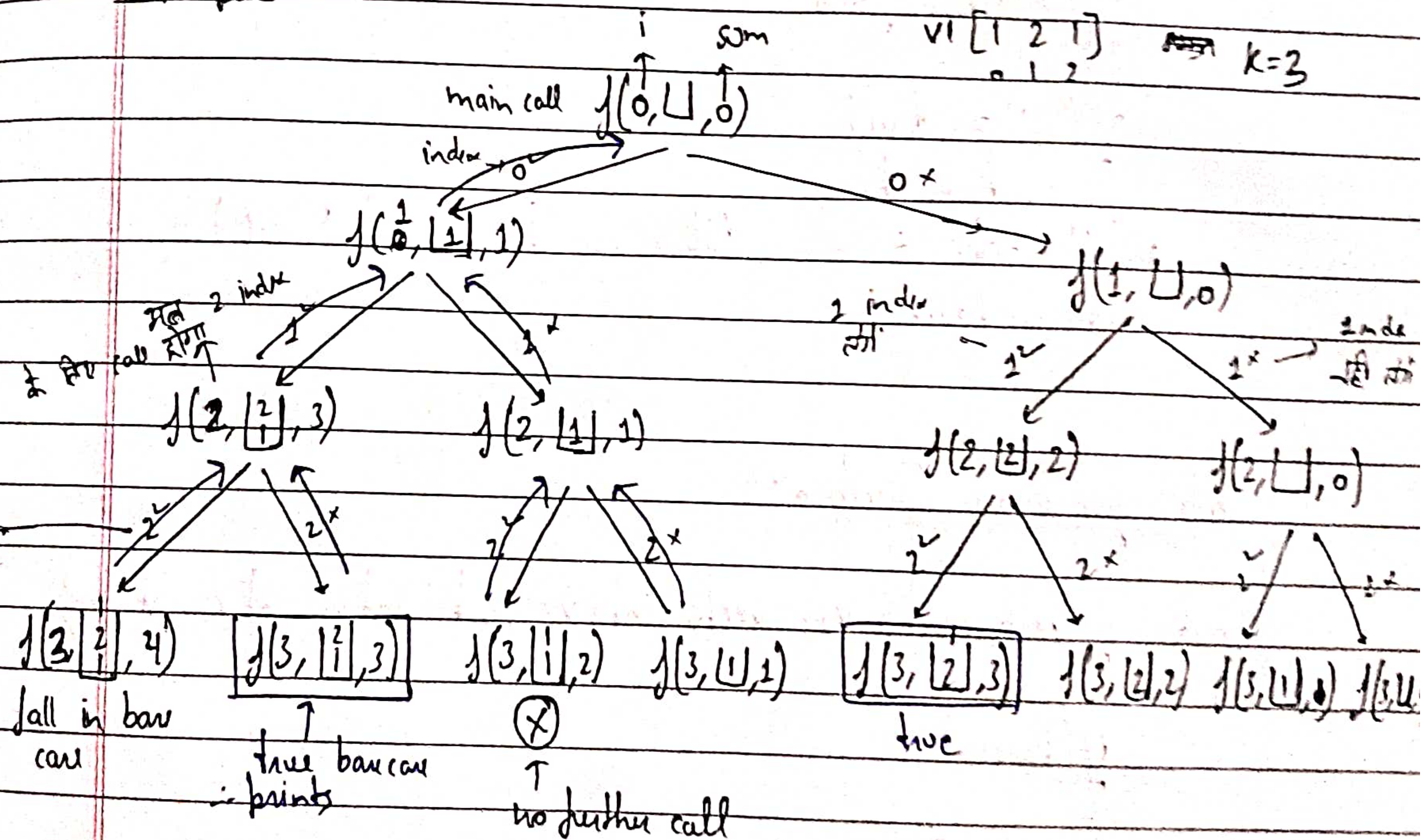
→ When there is multiple ~~recursion~~ function call inside the function then remember one function complete its execution then other one is called.

मतलब

एक function call हुआ अपना कुछ काम किया complete हुआ, return हुआ फिर उसके निचे वाले function ~~को~~ call होगा



Example



return; → back to caller function

लेकिन कुछ return नहीं कर रहा

void के function return type हमारा

(III) To print only one Subsequence whose Sum is k:

*** Technique to always print One Answer

```
bool f()
{
    base case
    {  condition → satisfied
       return true
       condition → x not satisfied
       return false.
    }
    if(f()==true)
        return true;
    ~~if(f()==t~~   ......   (there can be more fun' call)
    return false;
}
```

flag variable can also be used

eg! bool flag=false; declare globally
& if give in base case that
if (flag==false)
 flag = true;

Code for before Question:

```
bool f(vector<int> v1, vector<int> v2, int i, int n, int k, int sum)
{
    if(i==n)
    {
        if(sum==k)
        {
            for(auto it: v2)
                cout<<it<<" ";
            return true;
        }
        return false;
    }
```

→ जैसे ही पहला subsequence मिला print it & return true to caller.

and caller returns true to caller so on & finally goes to main caller and complete the recursion.

```
        v2.push_back(v1[i]);
        sum += v1[i];
        if( f(v1, v2, i+1, n, k, sum)==true)
            return true;
        v2.pop_back()
        sum -= v1[i]
        if( f(v1, v2, i+1, n, k, sum) == true)
            return true;
        return false;
}
```

अगर no subsequence present
return 0(false)

(IV) Count the Subsequence with sum==k : $O(2^n)$

For every index you have 2 choices either pick or not pick

$\underset{0}{2}\ \underset{1}{2}\ \underset{2}{2} \to 2^3\ (2^n)$

Technique : ★★★

```
int f()
{
        base case
            return 1 → condition satisfy
            return 0 →    "       not "

        l = f();
        r = f();

        return l+r;
}
```

0 initially

Code

```
int f(vector<int> v1, int i, int n, int k, int sum)
{
    if(i==n)
    {
        if(sum==k)
            return 1;
        return 0;
    }
    sum += v1[i];
    int l = f(v1, i+1, n, k, sum);
    sum -= v1[i];
    int r = f(v1, i+1, n, k, sum);
    return l+r;
}
```

eg: I/P v2 [1 2 1] k=3

O/P 2

12 21

★★

(VI) Combination Sum

I/P → n=4 k=7

a[] → 2,3,5,6

O/P → 2 2 3

2 5

find sum = 7 by taking no. any no. of times.

Complexity TC → $2^t \times k$

t → target variable

k → time

Remember: pass vector with address (&v1)


```
class Solution{
public:
    void f(vector<int> &v1, vector<vector<int>> v2, vector<int> ds,
                          int i, int t)
                                   ↳ target (k)
    {
        if( i==v1.size())
        {
            if(t==0)
            {     v2.push_back(ds);
            }
            return;
        }
        if(v1[i] <= t)
        {
            ds.push_back(v1[i]);
            f(v1, v2, ds, i, t-v1[i]);
            ds.pop_back();
        }
        f(v1, v2, ds, i+1, t);
    }
public:
    vector<vector<int>> CombinationSum(vector<int> &v1, int t)
    {
        vector<vector<int>> v2;
        vector<int> ds;
        f(v1, v2, ds, 0, t);
        return v2;
    }
};
```

Recursion Call for eg: v1 [2 3 5 6] t=7

→ left child is pick condition

→ right —— not ——