RECURSION - CLASS 1

Da / /.....

Recursion :-

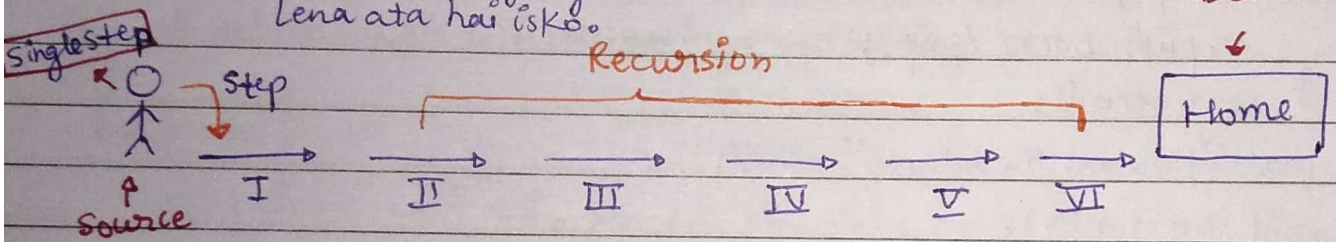
↳ Bookish Term → when a function calls itself directly/indirectly

↳ In-depth → Solution of Bigger Problem **depends** → Solution of chotti problem of same type.

∴ we can apply recursion in this case.

↳ Bhaiya ki Line → 1 case tum solve Karo, baki recursion sambhal lega

Problem statement - Isko ghr jaha hai but ek time pr single step Lena ata hai isko.



Bigger Problem → Step(6) = 1 + step(5)

∴ we can apply recursion in this case.

Single step — Recursion

Problem Statement - solve(n) **find** → 2^n

Bigger Problem → 2^n

$solve(n) = 2^n$

$Solve(n) = 2 \times 2^{n-1}$

∵ $Solve(n-1) = 2^{n-1}$

$Solve(n) = 2 \times solve(n-1)$

Big Problem — Choti Problem

∴ same type of problems so we can apply recursion here.

Spiral

Date ..../..../.....

PS → solve (n) ⟶ (n ⟶ 1) counting Print

i.e. solve (n-1) ⟶ (n-1 ⟶ 1) counting Print krega

Solve (n) → n, n-1, n-2, - - - - , 1

Solve (n) = n, (n-1, n-2, - - - - - 1)

Recursion

Big Problem ← Solve (n) = n, solve (n-1) → small problem of same type

∵ we can apply Recursion in this case

→ Factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

Solve (5) = 5!

solve(5) = 5 x 4 X 3 X 2 X 1

4!

solve(4) = 4! = 4X3X2X1

solve(5) = 5X4!

→ Ye Recursion Krdega

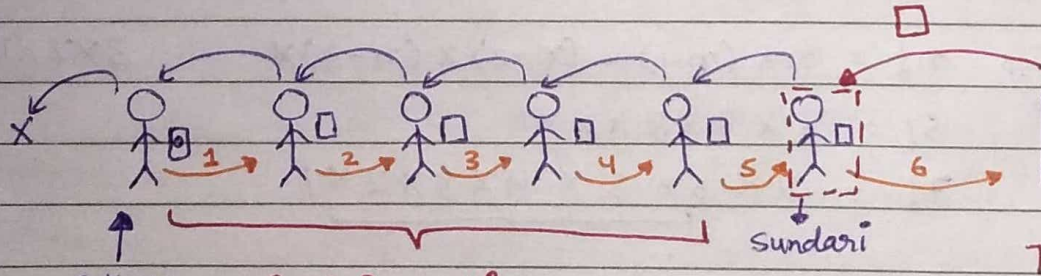
Solve (5) = 5 x solve(4)

Big Problem — Ye hum Krange — choti problem

Rule → if piche → Present → pass the paper

→ if → piche → absent → Rukjao

Ex:-



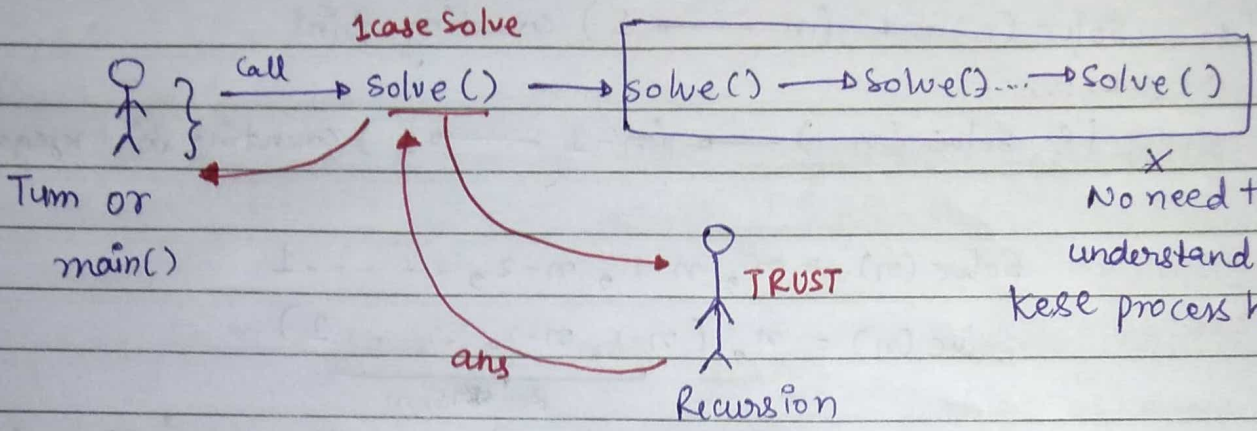
Rule → Aage wale Ko is blank paper pe +1 likh Kedo

Teacher ne sundari Ko paper diya tha to usse himapis chaiye !!

Piche K process se iskokoi Lena dena nhi !!

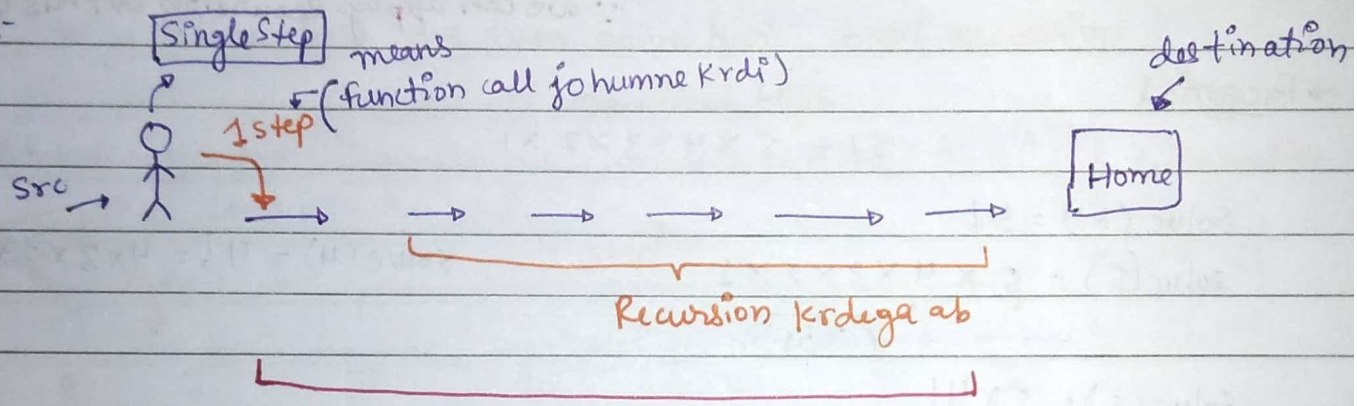
Spiral

Date ..../..../.....



∴ Recursion pe trust rakho ki ye ans laake dega bs.

Ex.-



Overall Ans	=	1	+	Recursion Ans
Big	=	1	+	choti

Q factorial → $n! = n \times (n-1) \times (n-2) \times (n-3) \times \cdots 3 \times 2 \times 1$

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$

$fact(7) = 7 \times 6!$

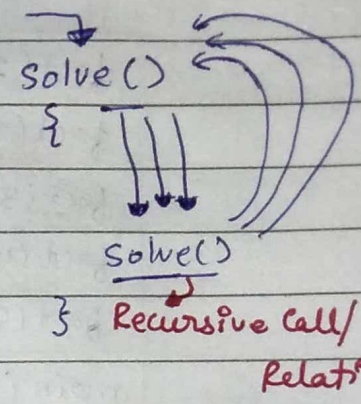
$fact(7) = 7 \times fact(6)$

$fact(n) = n \times fact(n-1)$

Date ..../..../.....

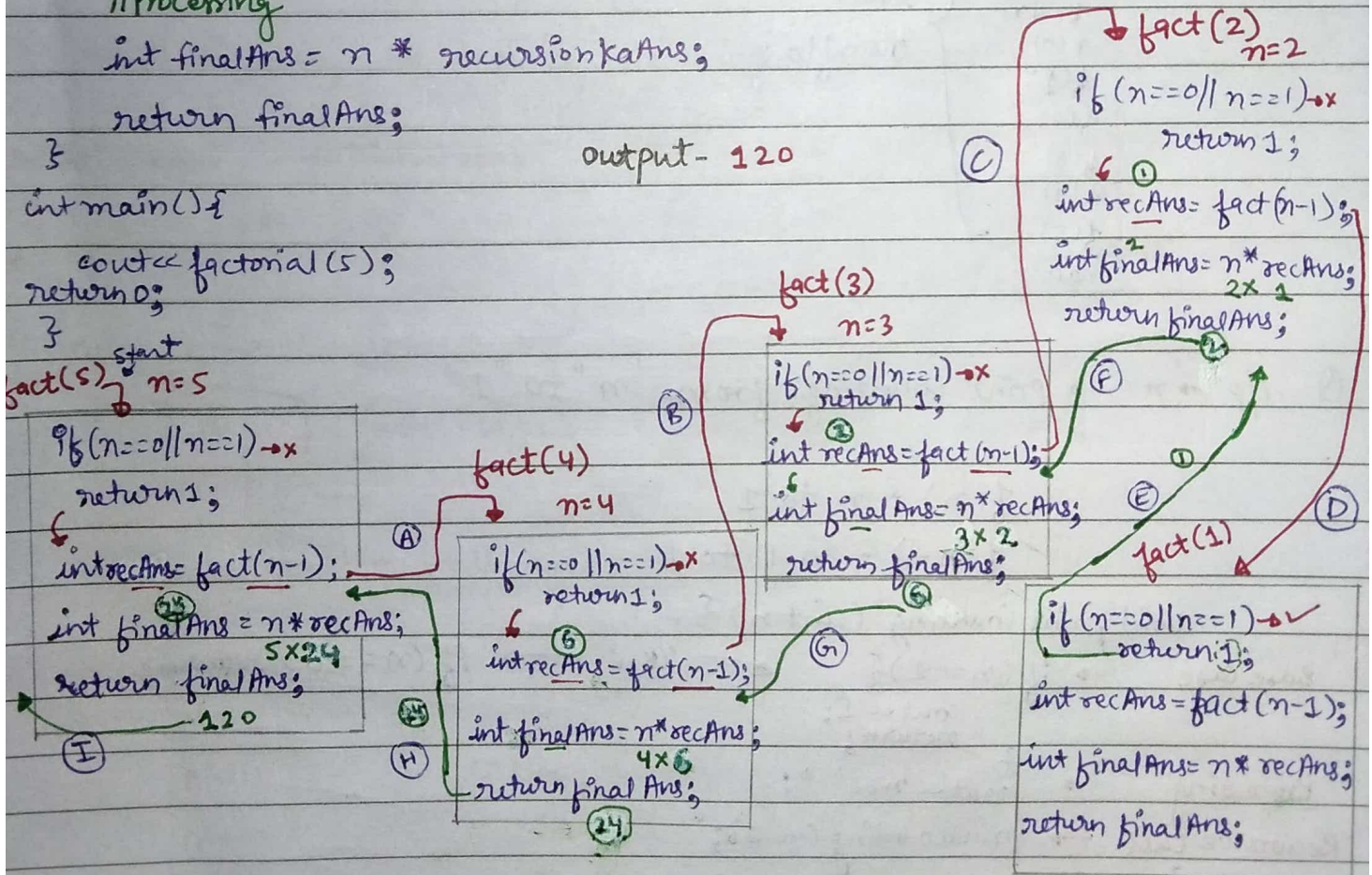
Parts of Recursive code :-

- Base Case → mandatory
  - ↳ To stop the recursion
- Recursive Call → mandatory
  - or
  - Recursive Relation
- Processing → optional




```
ode⇒ int factorial(int n){
    //baseCase
    if (n==1) return 1;
    if(n==0) return 1;
    //Processing
    //Recursive Relation
    int recursionKaAns = factorial(n-1);
    //Processing
    int finalAns = n * recursionKaAns;
    return finalAns;
}
int main(){
    cout<< factorial(5);
return 0;
}
```

output- 120



Spiral

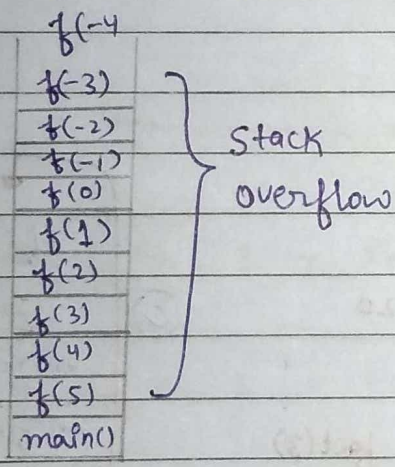
Date ..../..../.....

fact(1)
fact(2)
fact(3)
fact(4)
fact(5)
main()

~~fact(1)~~	Popped from stack and returned 1
~~fact(2)~~	Popped & returned 2
~~fact(3)~~	Popped & returned 6
~~fact(4)~~	Popped & returned 24
~~fact(5)~~	Popped & returned 120
main()	

∴ function call stack

→ If we don't apply Base case, then stack will filled with calls and it arise stack overflow condition and memory gets occupied by the program.



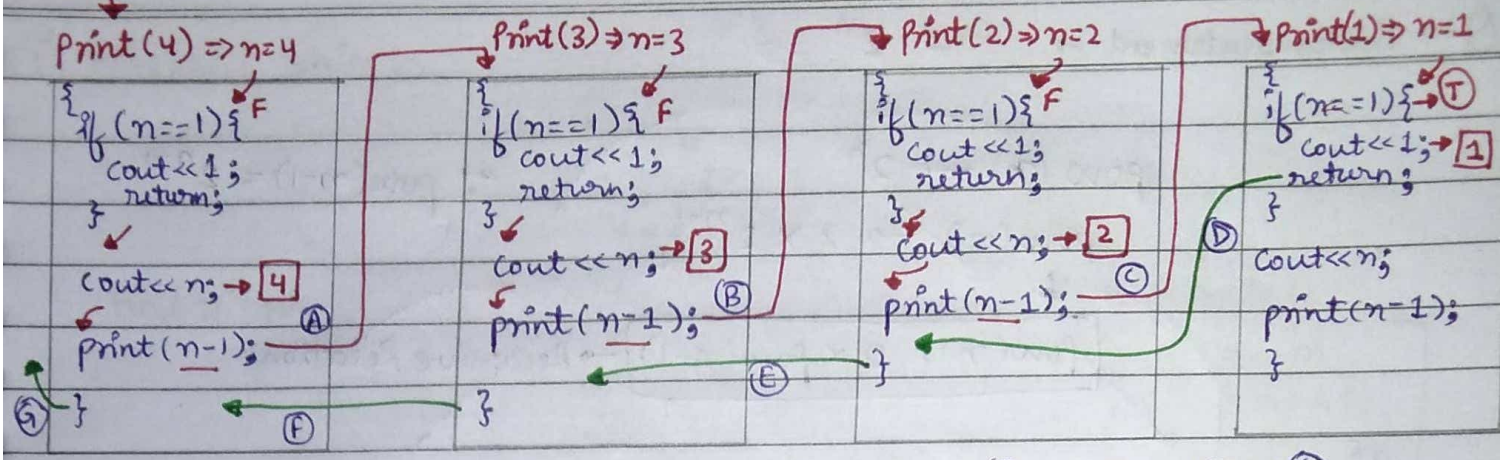
Q i/p → n → print counting from "n" to "1"

f(n) = n to 1

f(n-1) = n-1 to 1

```
Void printCounting (int n) {
Base Case  → if (n==1){        ← //B.C →  if (n==0) return;
               cout<< 1;
               return;
             }
Processing → cout<< n<<" ";
Recursive call → printCounting(n-1);
             }
```

Date ..../..../.....



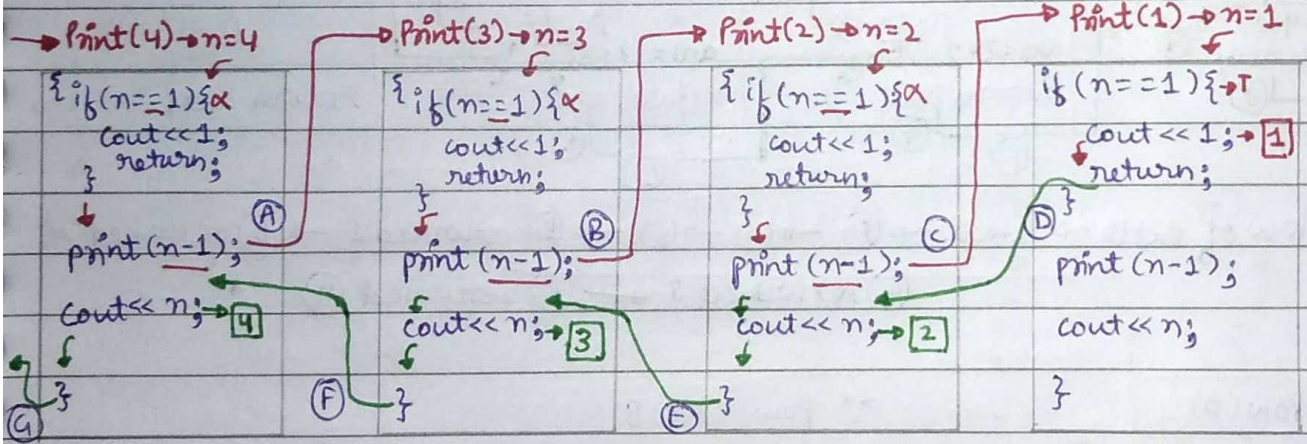
Flow of calls → (A) → (B) → (C) → (D) → (E) → (F) → (G)
↳ Processing BEFORE calling.

function call stack	Print(1)
	Print(2)
	Print(3)
	Print(4)
	main()

∴ It will print → 4 3 2 1 and popped out accordingly.

∵ Here, the function is ending with Recursive Relation. So, it's called Tail Recursion.

→ when we change the order of Processing and Recursive Relation.



Flow of calls → (A) → (B) → (C) → (D) → (E) → (F) → (G)
↳ Processing AFTER calling.

function call stacks	Print(1)
	Print(2)
	Print(3)
	Print(4)
	main()

∴ It will print → 1 2 3 4 & popped out

∵ Here, the Recursive Relation is before processing it's called Head Recursion.

Spiral

Date ..../..../.....

Q → Problem Statement → find 2^n

pow(n) → 2^n ∵ pow(n-1) = 2^{n-1}

pow(n) → $2 \times 2^{n-1}$

↓

pow(n) = 2 × pow(n-1) → Recursive Relation

2^5

↘ 2×2^4

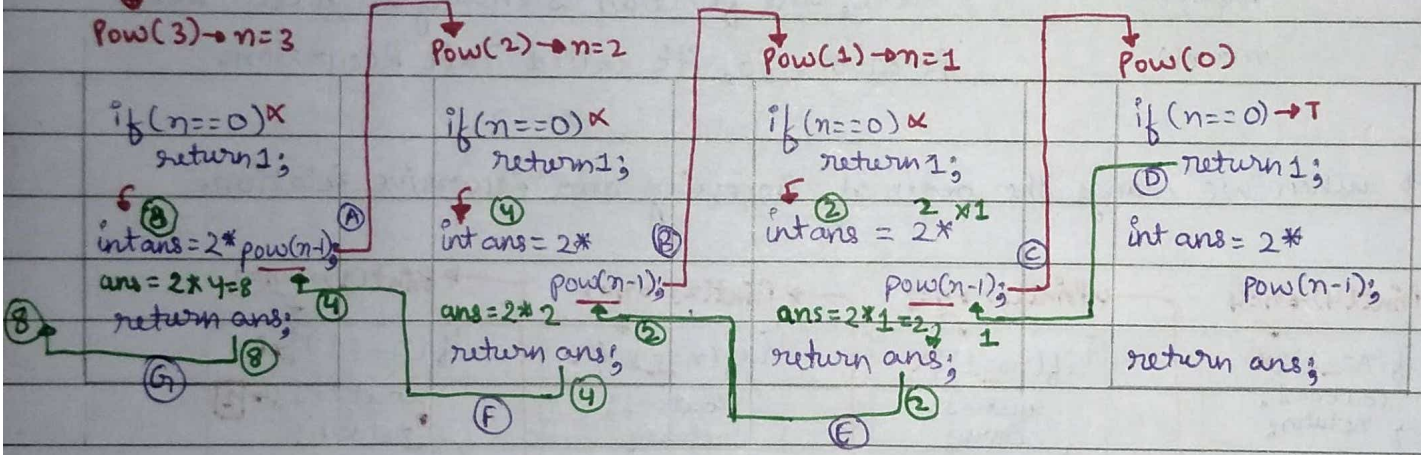
↘ 2×2^3

↘ 2×2^2

↘ 2×2^1

↘ 2×2^0 = 1 → Basecase → if(n==0) return 1;

Start



Flow of calls → start → (A) → (B) → (C) → (D) returned 1 → (E) returned 2 → (F) returned 4 → (G) returned 8

Pow(0)
Pow(1)
Pow(2)
Pow(3)
main()

∵ Pow(3) = 8

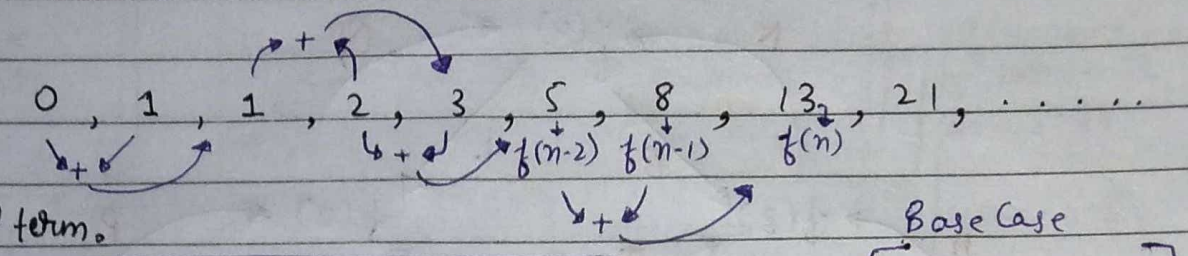
↳ Each call will pop out after returning particular ans.

Function call stack

Spiral

Date ..../..../.....

Q Fibonacci Series :-



find n^{th} term.

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

↳ Recursive Relation

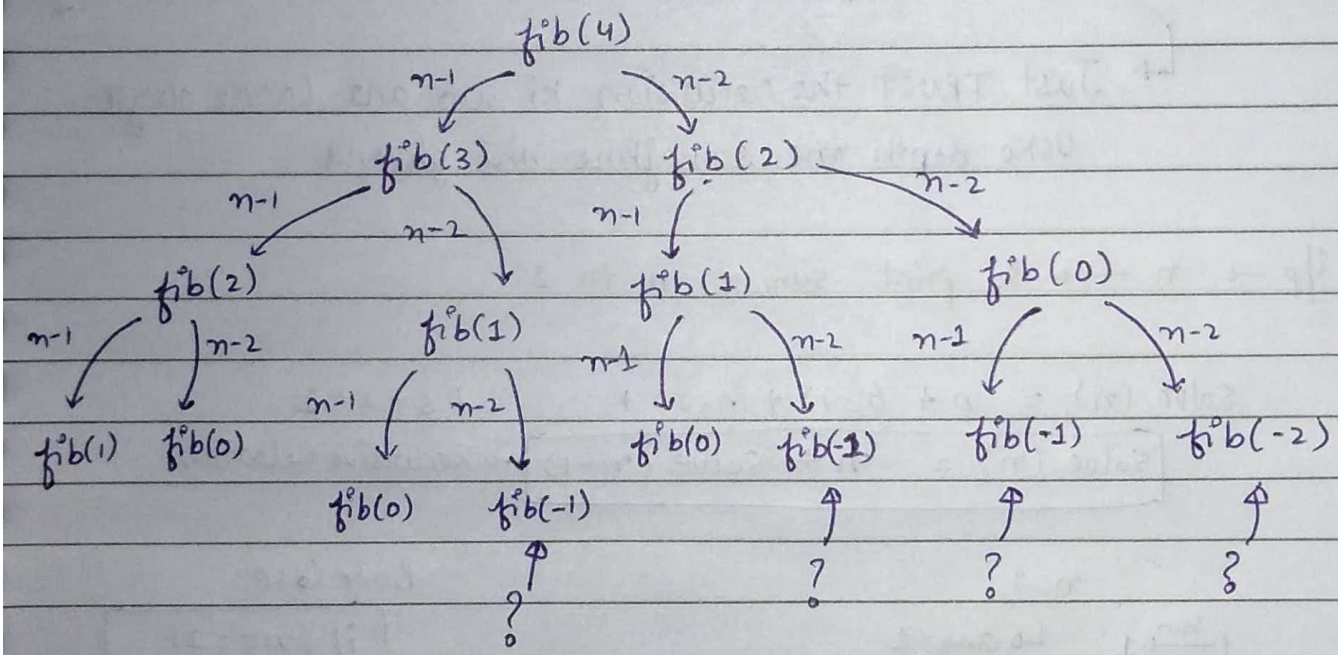
Base Case

```
if (n == 0)
    return 0;
if (n == 1)
    return 1;
```

Code :-

```
void fib (int n) {
//Base Case
if(n==0) return 0;
if(n==1) return 1;
//R.R
int ans = fib(n-1) + fib(n-2);
return ans;
```

Recursion Tree



fib(0) Ke niche ans kese nikalu nhi pta

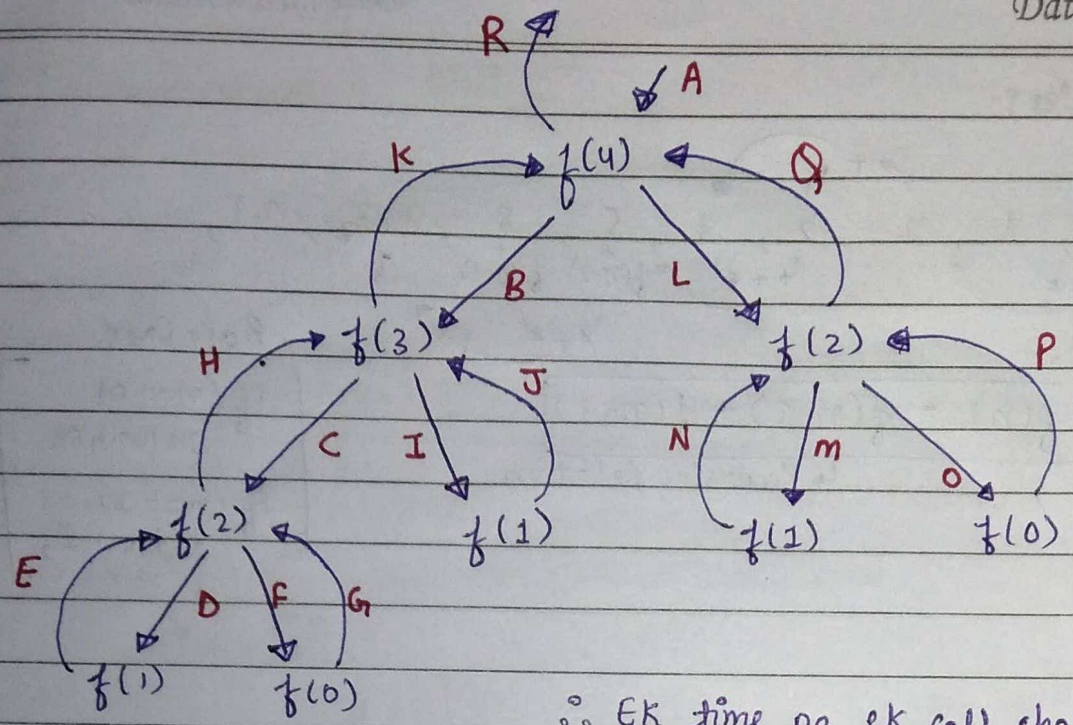
or

fib(1) Ke niche kese nikalu nhi pta so yahi hai Base Case, inki values pta honi chaiye.

```
if(n==0)
    return 0;
if(n==1)
    return 1;
```

Base Case

Date ..../..../.....



∴ EK time pe ek call chalegi

→ [To understand Recursion, you need to first understand Recursion]

↳ Just TRUST the recursion ki wo ans Laake dega. Usko depth me samajhne nhi ghusna

Q i/p → n → so print sum of n to 1

Solve (n) = n + (n-1) + (n-2) + ---- + 3 + 2 + 1

Solve (n) = n + solve (n-1) → Recursive Relation

n = 1

1 —Sum→ 1 Lo ans = 1

Base Case

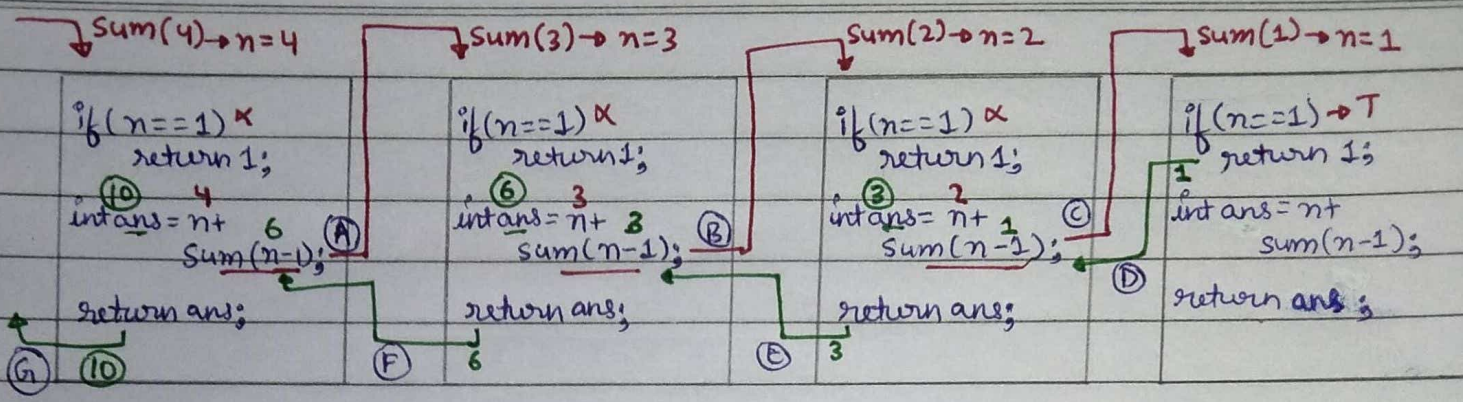
```
if (n==1)
    return 1;
```

Code :-

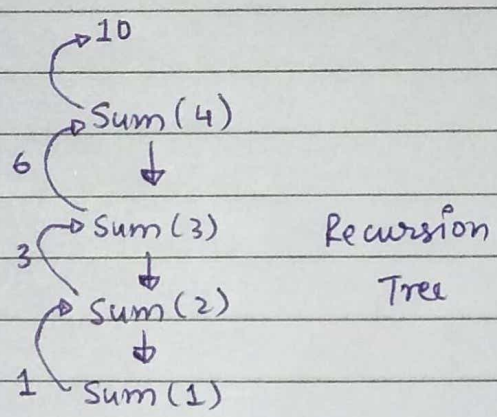
```
int sum (int n){
  //Base Case
  if(n==1)
    return 1;

  int ans= n+ sum(n-1);
  return ans;
}
```

Date ..../..../.....



		Sum(4) = 10 (1+2+3+4)
	Sum(1)	• All entries will be pop out after returning particular value.
function	Sum(2)	
call	Sum(3)	
stack	Sum(4)	
	main()	



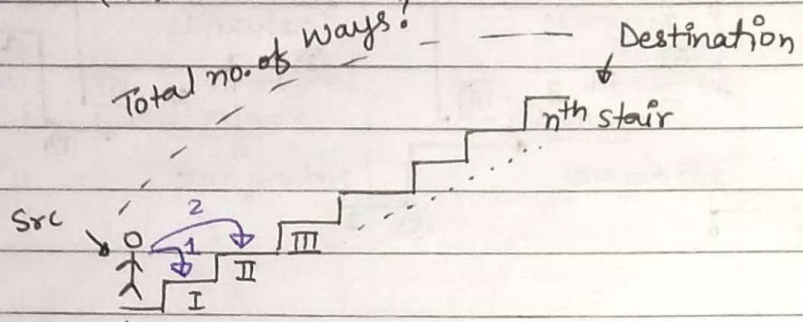
Recursion Tree

Spiral

RECURSION - CLASS 2

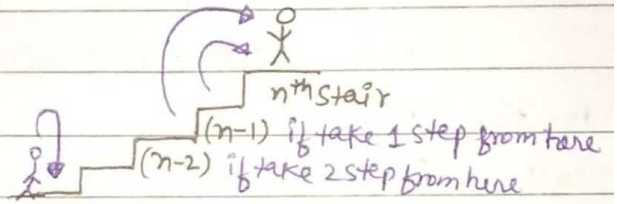
Date ..../..../.....

Q climbing Stairs (70) leetcode.



Can take either 1 step or 2 at once.

f(n) → no. of ways to reach n^{th} stair

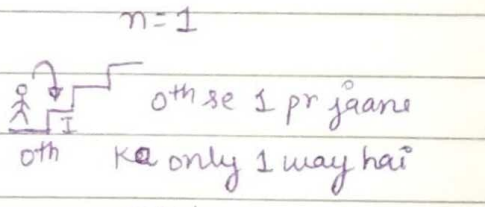


$$f(n) = f(n-1) + f(n-2)$$

R.R

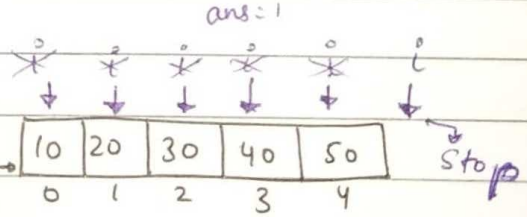
Code:-

```
int climbStairs (int n){
if(n==0) return 1;
if(n==1) return 1;
int ans = climbStairs(n-1) + climbStairs(n-2);
return ans;
```



Arrays & Recursion:-

Q Print array elements using recursion.



array size index → Pass them in a funcn.

f(arr, 5, 0)
↓ print[10]
index++;

f(arr, 5, 1)
↓ Print 20
index++;

f(arr, 5, 2)
Print 30
index++;

f(arr, 5, 3)
print 40
index++

f(arr, 5, 4)
print 50
index++

Base case
index ≥ size
Rukjao

f(arr, 5, 5)

Date ..../..../.....

Q Search in Array.

i/p → arr →

10	20	30	40	50
0	1	2	3	4

target = 50 return T/F.

↳ milne pr → return True

↳ entire array traverse hogya → return false

Code:-

```
bool searchArray(int arr[], int size, int target,
                 int index) {
    //Base Case
    if (index >= size) return false;
    //Processing
    if (arr[index] == target) return true;
    //Recursive Relation
    bool ans = searchAns(arr, size, target, index+1);
    return ans;
```

call from main()

searchArr(arr, 3, 30, 0)

if (index ≥ size) ✗ return false;
if (arr[Index] == target) 10 == 30 ✗
int ans = searchArr(arr, size, target, index+1); T
return ans;

searchArr(arr, 3, 30, 1)

if (index ≥ size) 1 ≥ 3 ✗ return false;
if (arr[Index] == target) 20 == 30 ✗
return true;
int ans = searchAns(arr, size, target, index+1); T
return ans;

searchArr(arr, 3, 30, 2)

if (index ≥ size) 2 ≥ 3 ✗ return false;
if (arr[Index] == target) 30 == 30
return true;
int ans = searchArr(arr, size, target, index+1);
return ans;

if (index ≥ size) return false;
if (arr[Index] == target)
return true;
int ans = searchArr(arr, size, target, index+1);
return ans;

T ← return ans; ← return ans; ← True

main → searchArr(arr, 3, 30, 0)

arr →

10	20	30
0	1	2

target → 30

Q find minimum no. in an array.

```
void minfinder(int arr[], int size, int index, int &mini){
    //Base Case
    if (index >= size) return;
    //Processing
    mini = min(mini, arr[index]);
    //Recursive Call
    minfinder(arr, size, index+1, mini);
}
```

KEEP IN MIND:-

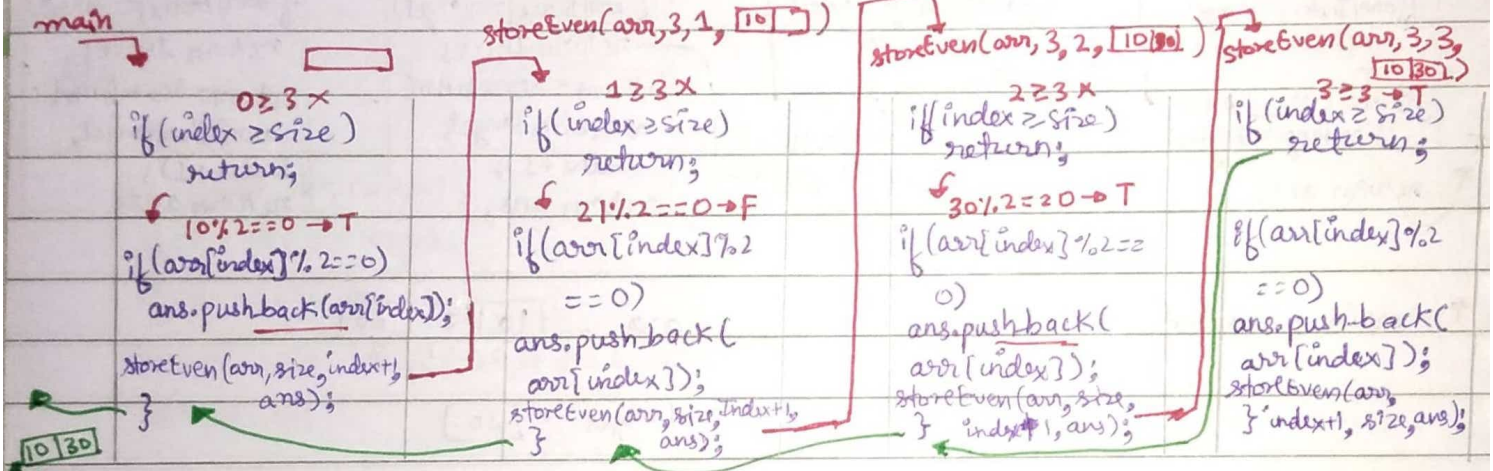
Tumhe agar kisi variable/DataStructure/Location ke andr answer store krna hai or tum usko kisi function me pass kar rahe ho or us function ke andr unme ans ka data store krna hai. So, make sure ki tumne us variable/DataStructure/Location ko BY REFERENCE pass kiya hai. Otherwise unki copy ban jygi or answer nhi store hoga usme.

Spiral

Date ..../..../.....

Q Store even numbers of an array in vector.

```
void storeEven ( int arr[], int size, int index, vector<int> &ans ) {
    // Base Case
    if ( index ≥ size ) return ;
    // Processing
    if ( arr[index] % 2 == 0 ) {
        ans.push_back ( arr[index] );
    }
    // Recursive Relation
    storeEven ( arr , size , index+1 , ans );
}
```



main → storeEven (arr, 3, 0, ▭)

array size Index vector<int> ans

10	21	30
0	1	2

Q You have an array in input. Double the values using recursion.

i/p → | 10 | 20 | 30 | 40 | 50 | → o/p | 20 | 40 | 60 | 80 | 100 |

```
void solve ( int arr[], int size, int index )
    // Base Case
    if ( index ≥ size ) return;
    // Processing
    arr[index] = 2 * arr[index] ;
    // Recursive Call
    solve ( arr, size, index + 1);
}
```

Spiral

Date ..../..../.....

Q Traverse the array using recursion and find maximum No.

```
void findMax (int arr, int size, int index, int &maxi ) {
    //Base Case
    if( index>= size) return;
    //Processing
    maxi = max ( maxi , arr[index ]);
    //Recursive Relation
    findMax ( arr, size , index+1, maxi);
}

int main(){
   arr[]= {10,20,30,40}; int size = 4; int index = 0;
   int maxi = INT_MIN;
   findMax ( arr , size, index , maxi);
}
```

Q find target in given array and return its index if its present and return -1 if it's not present.

```
int findTarget ( int arr[], int size, int index , int target) {
   //Base Case
   if( index >= size) return -1;
   // Processing
if(arr[index] == target)  return index;
   // Recursive Relation
   findTarget (arr , size, index+1, target );
}
```

Q Print index of all occurrence of target in Array.

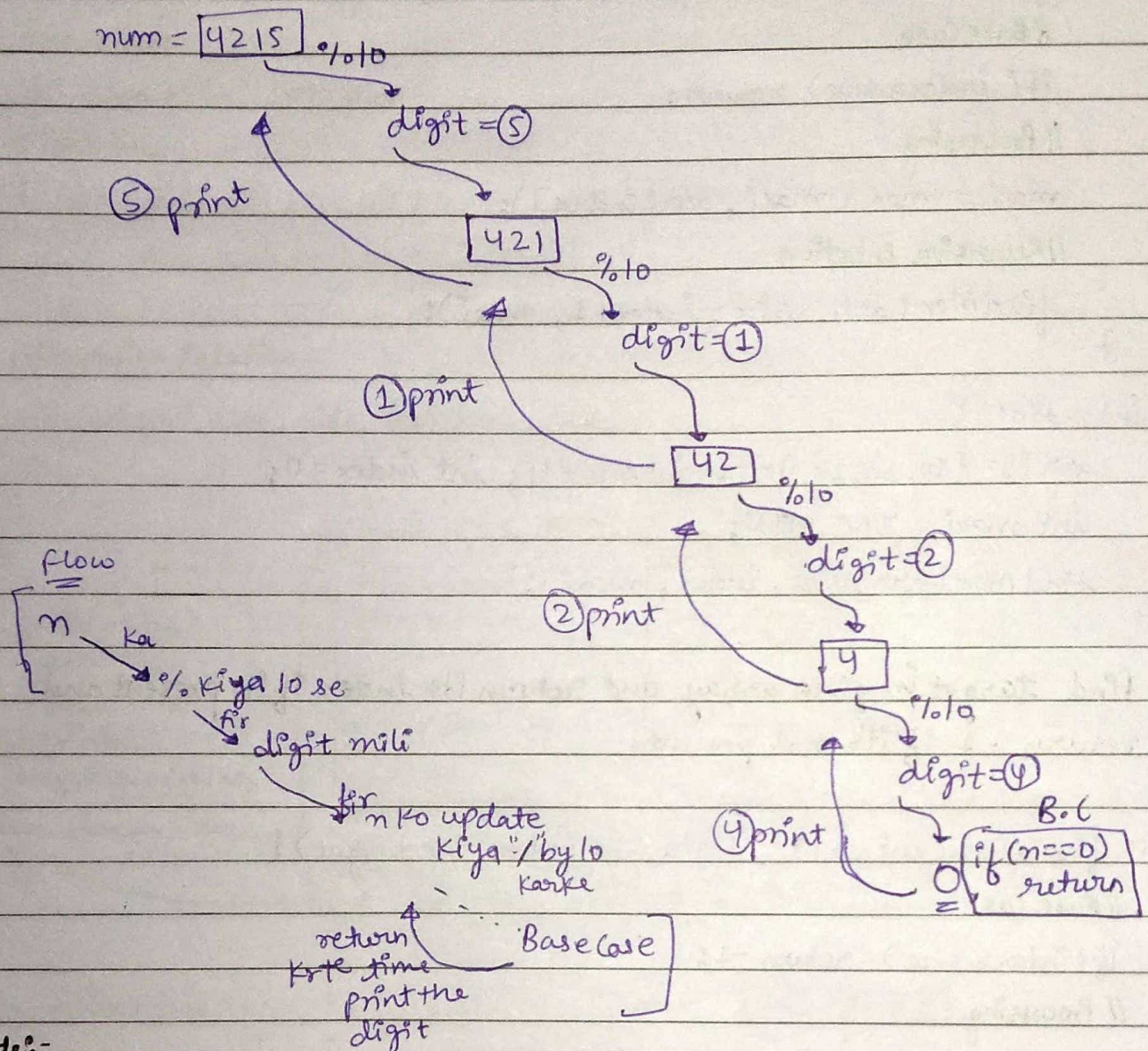
```
void printAllOccurrence( int arr[], int size, int index, int target){
   if ( index >= size) return ;   // Base Case
   if ( arr[index] == target) cout<< index<<" "; //Processing
   printAllOccurrence ( arr , size, index+1, target ); //R.R
}
```

Spiral

Date ..../..../.....

Q You have an integer in input and you have to print its digits.

Ex:-



Code:-

```
void printDigit(int num){
    if(num==0) return;      // Base Case
    int digit = num%10;     // Processing
    num = num/10;           // Updating num
    printDigit(num);        // Recursive Relation
    cout<<digit<<" "<<endl; // Processing
}
```

Date ..../..../.....

Q You have integer input. Store its digits into a vector.

```
void storeDigits ( int num, vector<int> & ans ){
    // Base Case
    if(num == 0) return;
    // Processing
int digit = num % 10;
    // Update num
    num = num / 10;
    // Recursive Relation
    storeDigits ( num, ans);
    // Processing
    ans.push_back(digit);
}
```

Q You have a vector in input. You have to return an integer using values of vector as digits for that integer.

i/p → vector → | 4 | 2 | 1 | 7 | o/p → | 4217 | → Integer

```
void createInteger ( vector<int> v, int index, int &ans){
    //Base Case
    if ( index ≥ size ) return ;
    // Processing
    ans = ans *10 + v[index];
    //Recursive Call
    createInteger ( v, index+1, ans);
}
```

Date ..../..../...

Q You have a string in input and target character. Print all occurrences of the target character into the string.

Ex:- string s = "Ditij" target = 'i' o/p → 1 3
0 1 2 3 4

```
void printStrTargetIndex (string str, int index, char target){
    // Base Case
    if (index == str.length()) return;
    // Processing
    if (str[index] == target) cout<< index <<" ";
    // Recursive Relation
    printStrTargetIndex (str, index+1, target);
}
```

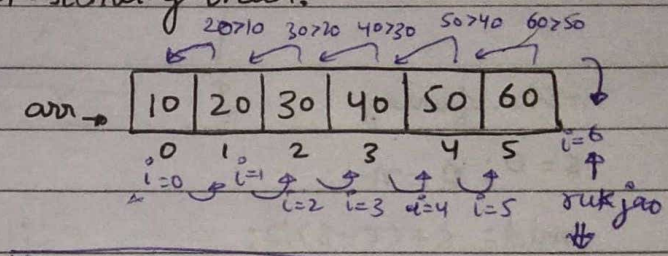
RECURSION CLASS-3

Date 13./.10./.23..

Q check the given array is sorted or not, in ascending order.

f(arr, size, index) — index: 1

(II) if arr[index] > arr[index-1]
- True → f(arr, size, index+1)
- false → return false



if (index ≥ size) → return true → Base Case

(I) main → f(array, size, index)
- array → arr
- size → 6
- (1) → for comparing it with previous ones.

Code:-

```
bool isSorted ( int arr[], int size, int index) {
    // Base Case
    if( index ≥ size ) return true;
    // Processing
    if ( arr[index] > arr[index-1]) {
        // Aage check krna padega ab
        // Ab recursion sambhalega
        bool aagekaAns = isSorted (arr, size, index+1);
        return aagekaAns;
    } else {
        return false; // Sorted nahi h
    }
}
```

arr → 10 20 30 (0 1 2)
size = 3
index = 1
main → f(arr, size, index)

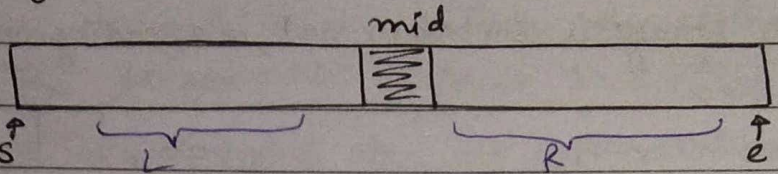
main

1 ≥ 3	2 ≥ 3 → F	3 ≥ 3 → T
if(index ≥ size) X return true;	if(index ≥ size) X return true;	if(index ≥ size) return true;
if(arr[index] > arr[index-1]) 20 > 10 → T	if(arr[index] > arr[index-1]) 30 > 20 → T	if(arr[index] > arr[index-1])
return f(arr, size, index+1);	return f(arr, size, index-1);	return f(arr, size, index-1);
else return false;	else return false;	else return false;
True	True	True

Spiral

Date ..../..../.....

Binary Search using Recursion :-



n = size of array

s = 0; e = n-1;

mid = s + (e-s)/2;

while (s > e) // Base case

if (arr[mid] == target) // found wala case
return mid;

if (arr[mid] < target) → Right me jao
return f(arr, mid+1 (s), e (e))

else → Left me jao
return f(arr, 0 (s), mid-1 (e))

Base case

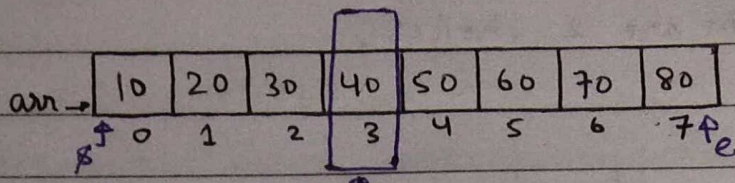
```
if (s>e)
  return -1;
```

Code :-

```
int binarySearch (int arr[], int s, int e, int target){
    // Base Case
    if ( s>e) return -1;
    // Processing → 1 case khud solve kro
    int mid = s + (e-s)/2;
    if (arr[mid] == target)
        return mid;
    // Baaki recursion sambhal lega
    if (arr[mid] < target) {
        // right me jao
        // s = mid+1 karke jyange so we can write
        return binarySearch (arr, mid+1, e, target);
    }
    else {
        // Left me jao
        // e = mid-1 karke so we can write
        return binarySearch (arr, s, mid-1, target);
    }
}
```

Date ..../..../.....

DryRun:-



target = 80

Call (arr, 0, 7, 80) → 6 returned

f(arr, s, e, target)

//B.C 0 > 7 → F x

//Processing mid = 0+7/2 = 3

40 == 80 → F

//R.R if(arr[mid] < target)

40 < 80 → True

return f(arr, mid+1, e, target)

return 6

f(arr, 4, 7, 80)

//B.C s > e → 4 > 7 F x

//Process mid = 4+7/2 = 5

60 == 70 → F

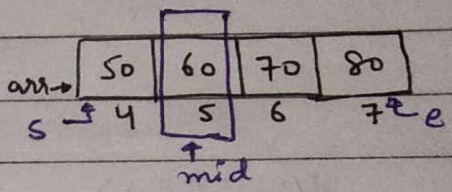
//RR if(arr[mid] < target).

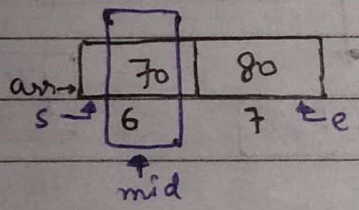
60 < 70 → T

// right me jao, s = mid+1

return f(arr, mid+1, e, target)

return 6





f(arr, 6, 7, 80)

//B.C → s > e → 6 > 7 → F x

//Processing mid = 6+7/2 = 6

if(arr[mid] == target)

70 == 70 → True

return (6) → mid ki value.

Subsequences of String :- [Include - Exclude Pattern]

From the given string, we can include or exclude any character in our output string. But ordering of characters must be same as it's in input string.

Ex:- i/p → "abc" ⇒ subsequences

✓ ✗ ✗ → a
✗ ✓ ✗ → b
✗ ✗ ✓ → c
✓ ✓ ✗ → ab
✗ ✓ ✓ → bc
✓ ✗ ✓ → ac
✓ ✓ ✓ → abc
✗ ✗ ✗ → ""

i/p → "xy" ⇒ subsequences

✓ ✗ → x
✗ ✓ → y
✓ ✓ → xy
✗ ✗ → ""

[∴ n = length of string → Total subsequences = 2^n]

Date ..../..../....

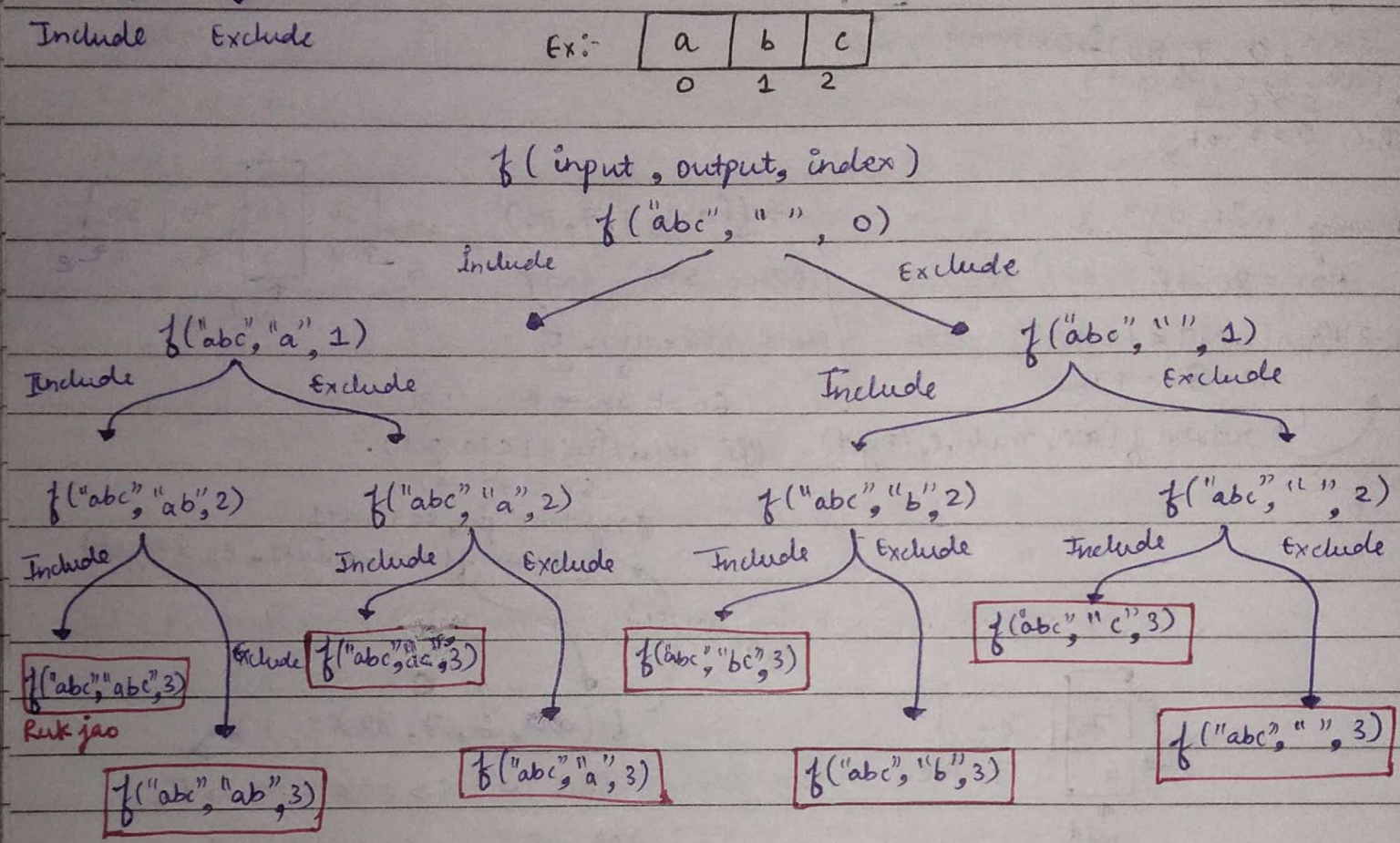
for every character, there are 2 choices.

'K'

Include Exclude

Ex:-

a	b	c
0	1	2



Code:-

```
void findSubSequences ( string input, string output, int index ){
    //Base Case
    if (index ≥ input.length()) {
        //ans jo hai, wo output string me built ho chuka hai
        //print krdo
        cout << output<<endl;
        return;
    }
    //Processing
    char ch = input[index];
```

Date ..../..../.....

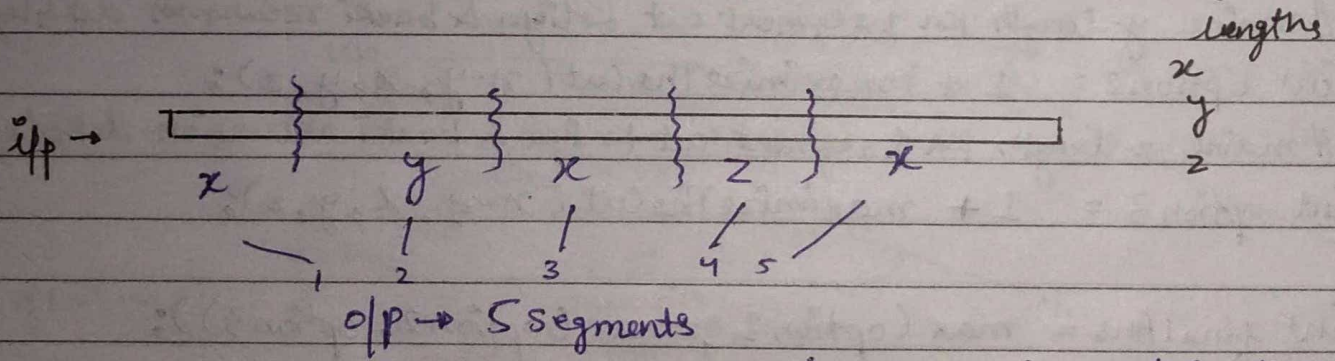
```
// include
output.push_back(ch);
findSubSequences(input, output, index+1);
// exclude
output.pop_back();
findSubSequences(input, output, index+1);
}
```

Exploring all Possible ways Pattern

Q Cut into segments / Maximize The Cut Segments (GFG)

i/p given a rod of n length. We can break it into 'x', 'y' & 'z' segments.
x, y, z are integers.

o/p → return the maximum no. of possible segments of x, y & z lengths.

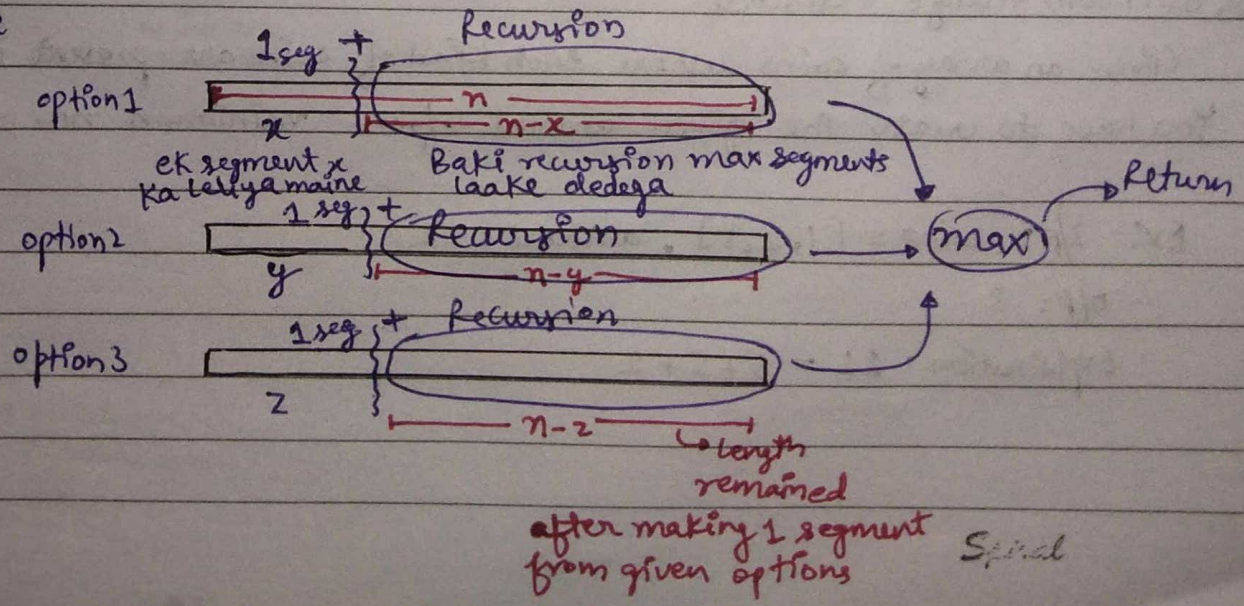


o/p → 5 segments

Keep in mind while applying this approach

→ Exploring All possible ways pattern (Ek case tum solve karo baki recursion sambhal lega)

options available
ya to x length
ya to y length
ya to z length
cut kr skte hai



Spiral

Date ..../..../.....

Code:-

```
int maximizeTheCuts(int n, int x, int y, int z){
    //Base Case
    if(n==0){
    // Jab length hi nahi hai rod ki to ans kya return krange go
    return 0;       // Length of rod = 0 ke liye no. of segments = 0
    }

    if(n<0){
    // Jab <0 wali condition aye to max me wo consider hina ho
     return INT_MIN;
    }

    // maine x length ka 1 segment cut krliya and baaki recursion dekhlega
    int option1 = 1 + maximizeTheCut(n-x, x, y, z);
    // maine y length ka 1 segment cut krliya & baaki recursion dekhlega
    int option2 = 1 + maximizeTheCut(n-y, x, y, z);
    // maine z length ka 1 segment cut krliya & baaki recursion dekhlega
    int option3 = 1 + maximizeTheCut(n-z, x, y, z);

    int finalAns = max(option1, max(option2, option3));
    return finalAns;
}
```

Q 322. Coin Change (Leetcode)

Given an array of coins where each kind of coin are present in infinite number. You have to create the given amount from minimum no. of coins.

Ex:- I/p: coins = [1,2,5], amount = 11

o/p: 3

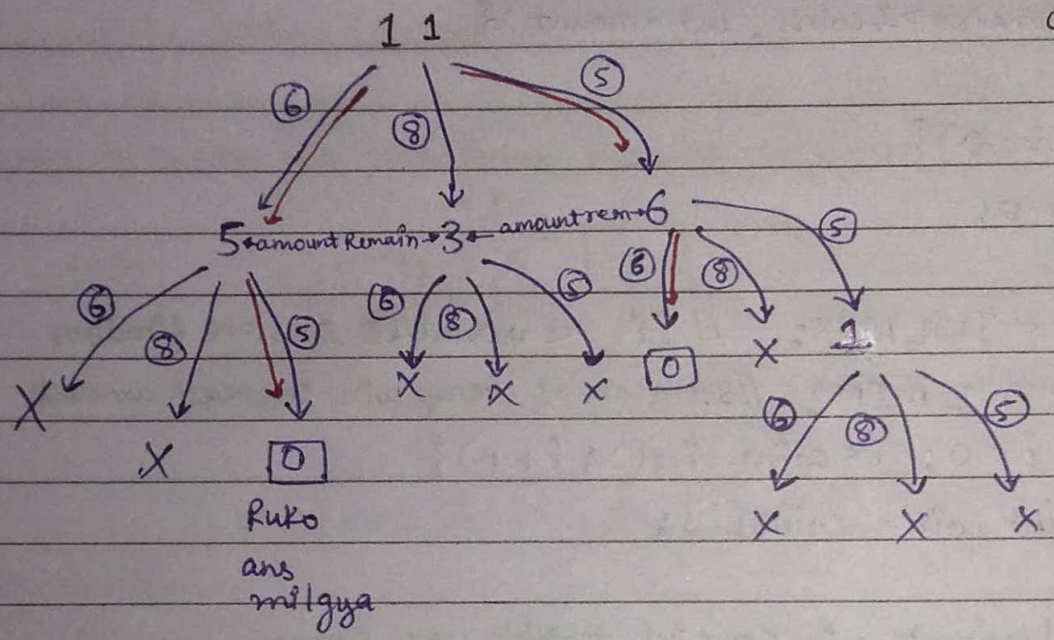
Explaination: 11 = 5+5+1

Date ..../..../.....

∴ Recursive Tree

Amount = 11

coins → [6, 8, 5]



Coin > amount value

↓

no recursive call

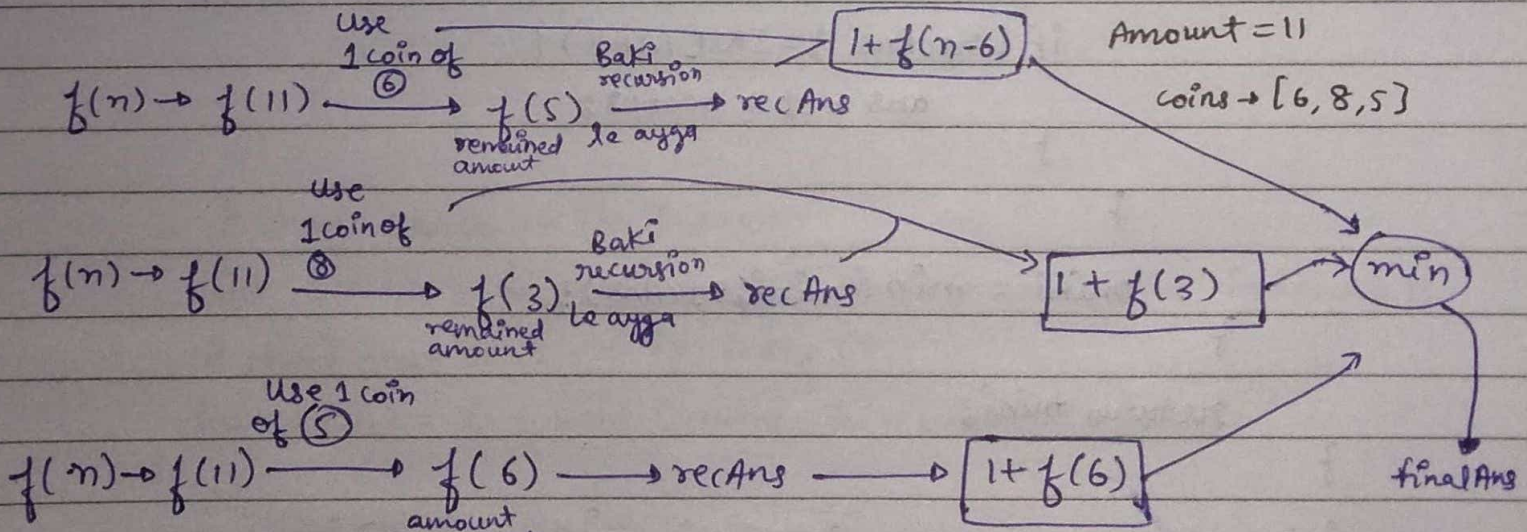
Base Case

```
if (n==0)
    return 0;
```

5 + 6 → 11 2 coins

6 + 5 → 11 2 coins

min → 2 coins



Dat : .../.....

Code:-

```
int solve (vector<int>& coins, int amount){
    // Base Case
    if (amount == 0){
        return 0;
    }
    int mini = INT_MAX;  // It's the variable to store finalAns
    int ans = INT_MAX;  // Store ans of every coin to create amount
    for (int i = 0; i < coins.size(); i++){
            int coin = coins[i];

            // current coin ko sirf tabhi use krange
            // Jab uski value <= amount hogi
            if (coin <= amount) {
                int recAns = solve (coins, amount - coin);
                if (recAns != INT_MAX){
                    ans = 1 + recAns;
                }
            }
            mini = min (mini, recAns);
        }
        return mini;
}
int coinChange (vector<int>& coins, int amount) {
    int ans = solve (coins, amount);

    if (ans == INT_MAX)
        return -1;
    else
        return ans;
}
```

Spiral

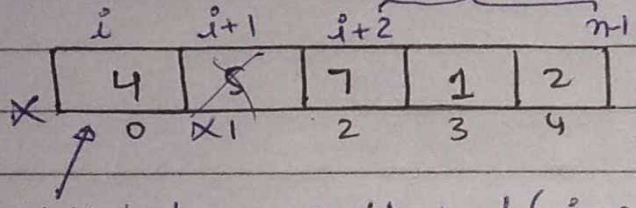
Date ../../....

Max sum of non-adjacent elements

Q 198. House Robber (leetcode)

Given an array of houses. we can only rob adjacent houses. We have to return the maximum amount which can be rob.

Ex:-



chori Krta hu → 4 + f(i+2, n-1)

chori nahi Kri → 0 + f(i+1, n-1)

max → final Ans

// B.C

i ≥ size → rukjao

Code:-

```
int solve (vector<int>& nums, int size, int index){
    // Base Case
    if (index >= size){
        return 0;
    }

    // chori Karlo → ith index pr
    int option1 = nums[index] + solve(nums, size, index+2);
    // chori mat karo → ith index pr
    int option2 = 0 + solve (nums, size, index+1);

    int finalAns = max (option1, option2);
    return finalAns;
}
int rob (vector<int>& nums) {
    int size = nums.size(); int index = 0;
    int ans = solve (nums, size, index);
    return ans;
}
```