Recursion in One Shot

-->> A function that called itself/
a function calling itself is RECURSION

What and Why?

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

$$f(n) = n \times f(n-1) \rightarrow \text{Recurrence Relation}$$

Loops → Factorial of a number

$$n! = n \times n-1 \times n-2 \times n-3 \cdots 3 \times 2 \times 1$$

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

$$5! = 5 \times 4!$$

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

$$4! = 4 \times 3!$$

Function calling itself

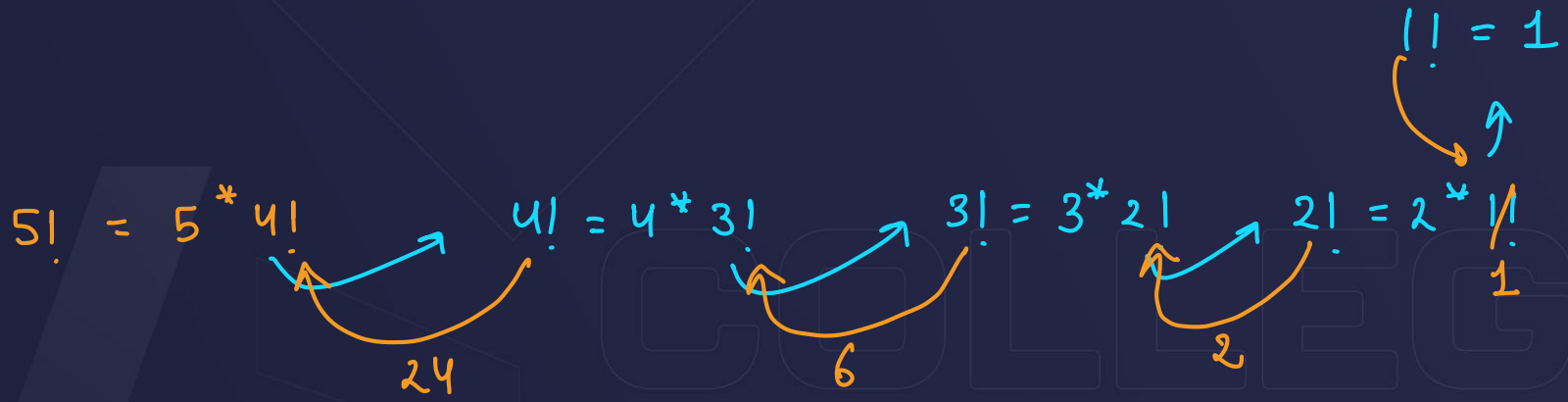
```
int factorial (int n){

    return n* factorial(n-1);

}
```

factorial(n) = n* factorial(n-1);

n! = n* (n-1)!



Ques : Make a function which calculates the factorial of n using recursion.

```c
int factorial(int n){
    return n*factorial(n-1);
}
```

4

4 x factorial (3);

```c
int main(){
    int n;
    printf("Enter a number : ");
    scanf("%d",&n);
    int fact = factorial(4);
    printf("%d", fact);
    return 0;
}
```

n = 4

```c
int factorial(int n){
    return n*factorial(n-1);
}
```

3

3 * factorial (2) ;

```c
int factorial(int n){
    return n*factorial(n-1);
}
```

2

2* factorial (1)

```c
int factorial(int n){
    return n*factorial(n-1);
}
```

1

1 * factorial (0)

```c
int factorial(int n){
    return n*factorial(n-1);
}
```

0

0* factorial (-1)

6-1

6-2

5

```
int factorial(int n){
    if(n==1) return 1;
    return n*factorial(n-1);
}
```

5 * factorial(4);

24 = 120

4

```
int factorial(int n){
    if(n==1) return 1;
    return n*factorial(n-1);
}
```

4* factorial(3);

6 = 24

1

```
int factorial(int n){
    if(n==1) return 1;
    return n*factorial(n-1);
}
```

3

```
int factorial(int n){
    if(n==1) return 1;
    return n*factorial(n-1);
}
```

3 * factorial(2);

2 = 6

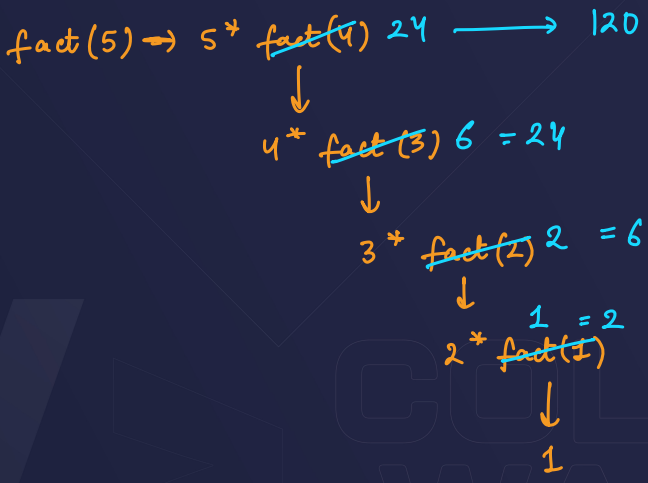
2

```
int factorial(int n){
    if(n==1) return 1;
    return n*factorial(n-1);
}
```

2 * factorial(1);

1 = 2

Tree Diagram



Ques : Print n to 1 using recursion

[6-4]

```
int decreasing (int n){
```

3

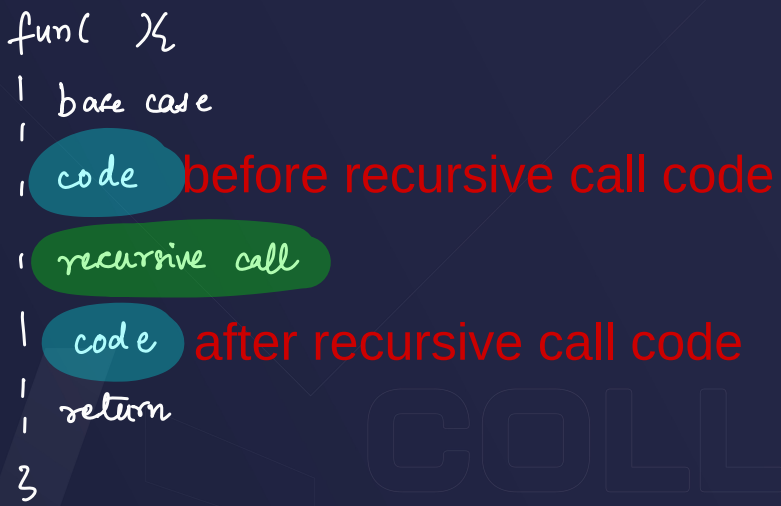
Output

7
6
5
4
3
2
1

'n' times good morning

[6-3]


void greeting(int n){
if(n==0) return;
printf("Good Morning\n");
greeting(n-1);
return;
}
Output
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
[6-3]


fun( ){
base case
code
before recursive call code
recursive call
code
after recursive call code
return
}

Ques : Print 1 to n

Output:

1
2
3
4
5

n = 5

```
1
void increasing(int n){
    if(n==0) return;
    printf("%d\n",n);
    increasing(n+1);
    return;
}
```

DRY run of wrong way

Output

- 1
- 2
- 3
- 4
- 5
- 6
- .
- .
- .

Using extra parameter

Ques : Print 1 to n (parameterised)

[6-5]

```c
#include<stdio.h>
void increasing(int x, int n){
    if(x>n) return;
    printf("%d\n",x);
    increasing(x+1,n);
    return;
}
int main(){
    int n;
    printf("Enter a number : ");
    scanf("%d",&n);
    increasing(1,5);
    return 0;
}
```

1 5

n = 5

5

Output

- 1
- 2
- 3
- 4
- 5
-

Ques : Print 1 to n (after recursive call) [6-6]

```
void increasing(int n){
    if(n==0) return; // base case
    increasing(n-1); // call
    printf("%d\n",n); // code
    return;
}
```

```
void increasing(int n){
    if(n==0) return; // base case
    increasing(n-1); // call
    printf("%d\n",n); // code
    return;
}
```

```
void increasing(int n){
    if(n==0) return; // base case
    increasing(n-1); // call
    printf("%d\n",n); // code
    return;
}
```

```
void increasing(int n){
    if(n==0) return; // base case
    increasing(n-1); // call
    printf("%d\n",n); // code
    return;
}
```

Output

- 1
- 2
- 3
- 4

Homework : Print Decreasing - Increasing

n = 4 →

4
3
2
1
1
2
3
4

Hint : Call se pehle , Call ke baad

H.W. n = 3 → DRY RUN

n = 4 , s = 0

Ques : Print sum from 1 to n (Parameterised)

```
void sum(int n, int s){
   if(n==0){
       printf("%d",s);
       return;
   }
   sum(n-1,s+n);
   return;
}
```

4 0 — 3 4 — α

```
void sum(int n, int s){
   if(n==0){
       printf("%d",s);
       return;
   }
   sum(n-1,s+n);
   return;
}
```

1 9 — 0 10 — α

```
void sum(int n, int s){
   if(n==0){
       printf("%d",s);
       return;
   }
   sum(n-1,s+n);
   return;
}
```

0 10 — α

Output

10

```
void sum(int n, int s){
   if(n==0){
       printf("%d",s);
       return;
   }
   sum(n-1,s+n);
   return;
}
```

3 4 — 2 7 — α

```
void sum(int n, int s){
   if(n==0){
       printf("%d",s);
       return;
   }
   sum(n-1,s+n);
   return;
}
```

2 4+3 — 1 9 — α

Ques : Print sum from 1 to n (Return type)

factorial (n) = n * factorial (n-1) ;

sum(n) = n + sum(n-1) ;

sum(5) = 1 + 2 + 3 + 4 + 5 = 5 + sum(4)

↓

4 + sum(3)

↓

3 + sum(2)

↓

2 + sum(1)

↓

1

Ques : Make a function which calculates 'a' raised to the power 'b' using recursion.

```
int power = 1;
int a, b;

for(int i=1; i<=b; i++){
    power = power * a;
```

$a^b = \underbrace{a \times a \times a \times a \ldots}_{b \text{ times}}$

```
if (b==0) return 1;
power(a,b) = a * power(a,b-1);
```

3

$$a^b = a * a^{b-1}$$

$2^4 = 2 * 2^3$ → $2^3 = 2 * 2^2$ → $2^2 = 2 * 2^1$ → $2^1 = 2 * 2^0$

$2^0 = 1$

*Multiple Calls

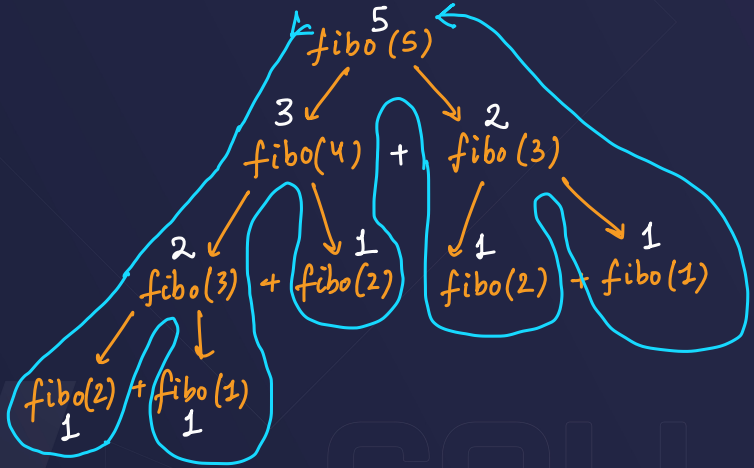
Ques : Write a function to calculate the nth fibonacci number using recursion.

1 1 2 3 5 8 13 21 34 55 89 ...

1 2 3 4 5 6 7 8 9 10 11

```
fibo(n) = fibo(n-1) + fibo(n-2);

if (n==1 or n==2) return 1
```



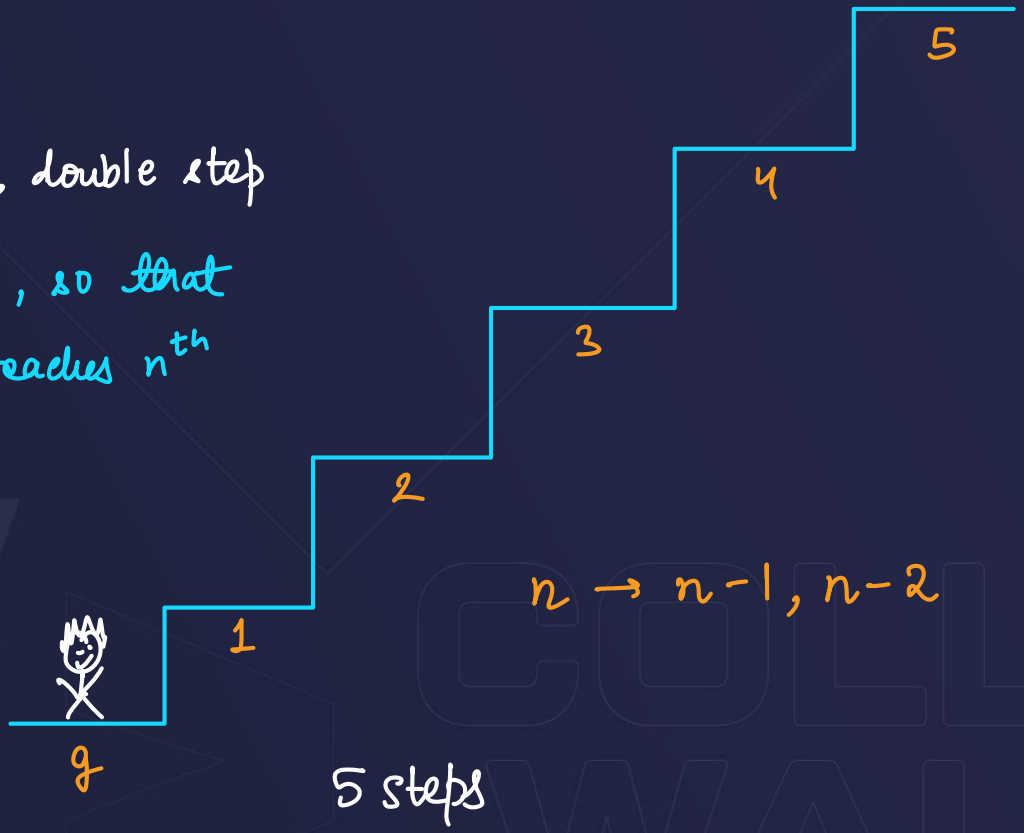
(n==1 || n==2)
return 1

Ques : Stair Path - 1

n^{th} stair

single step, double step

no. of ways, so that the person reaches n^{th} stair.



$n \rightarrow n-1, n-2$

1 1 1 1 1
1 1 1 2
1 2 1 1
1 1 2 1
2 1 1 1
1 2 2
2 1 2
2 2 1

total 8

Ques : Stair Path - 2

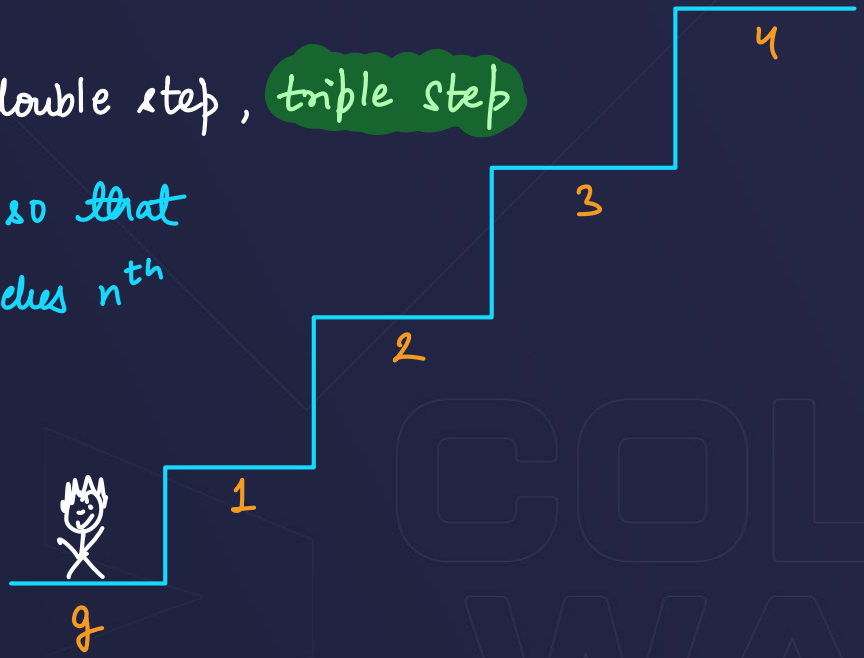
H.W.

[6-12]

n^{th} stair

single step, double step, triple step

no. of ways, so that the person reaches n^{th} stair.



1 1 1 1

1 1 2

1 2 1

2 1 1

2 2

1 3

3 1

Ques : Power function (logarithmic)

$a^b = a * a^{b-1}$;

$2^{64} = 2 * 2^{63}$

$2^{63} = 2 * 2^{62}$

$\vdots$

$2^1 = 2 * 2^0$

64 calls

if (b == 0) return 1;

if (b == 1) return a ;

pow(a,b) = pow(a, b/2) * pow(a, b/2)

$a^b = a^{b/2} * a^{b/2}$;

$2^{64} = 2^{32} * 2^{32}$

$2^{32} = 2^{16} * 2^{16}$

$2^{16} = 2^8 * 2^8$

$2^8 = 2^4 * 2^4$

$2^4 = 2^2 * 2^2$

$2^2 = 2^1 * 2^1$

6 calls

Ques : Power function (logarithmic)

$$a^b = a^{b/2} * a^{b/2}$$

Problem :

$$2^7 = 2^{7/2} * 2^{7/2}$$

$$2^7 = 2^3 * 2^3 = 16$$

($2^3 = 2^4 * 2^4$)

$$2^3 = 2^{3/2} * 2^{3/2}$$

$$2^3 = 2^1 * 2^1$$

$$2^3 = 2 \times 2 = 4$$

Solution :

if b is even

$$a^b = a^{b/2} * a^{b/2}$$

if b is odd

$$a^b = a^{b/2} * a^{b/2} * a$$

$b = 5$

$$a^5 = a^{5/2} \times a^{5/2} \times a$$

$$= a^2 \times a^2 \times a$$

```
int powerlog(int a, int b){
    if(b==0) return 1;
    int x = powerlog(a,b/2);
    if(b%2==0)
        return x*x;
    else
        return x*x*a;
}
```

$2^9 = 2^4 * 2^4 * 2;$

$2^4 = 2^2 * 2^2;$

$2^2 = 2^1 * 2^1;$

$2^1 = 2^0 * 2^0 * 2;$

$2^0 = 1$

$a = 2$

$b = 9$

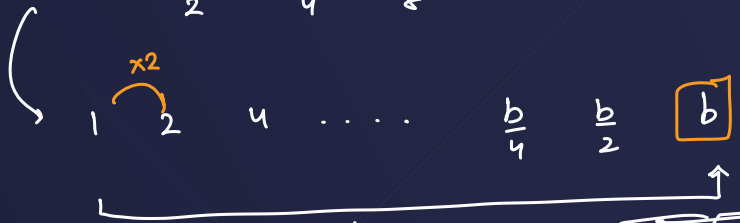
$b \to b/2 \to b/4 \to b/8 \cdots \to 1$

$a^b \Rightarrow$ b calls

$2^{100} =$ 100 calls

$a^b \rightarrow$ b calls $\log_2(b)$ calls

M-2 : $b + \frac{b}{2} + \frac{b}{4} + \frac{b}{8} + \cdots 2\ 1$ G.P.



$b = a_n$

$a_1 = 1$

$r = 2$

$\rightarrow$

$a_n = a_1 \times r^{n-1}$

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

$\rightarrow b = \frac{2^n}{2} \Rightarrow 2^n = 2b$

$\Rightarrow n = \log_2 2b$

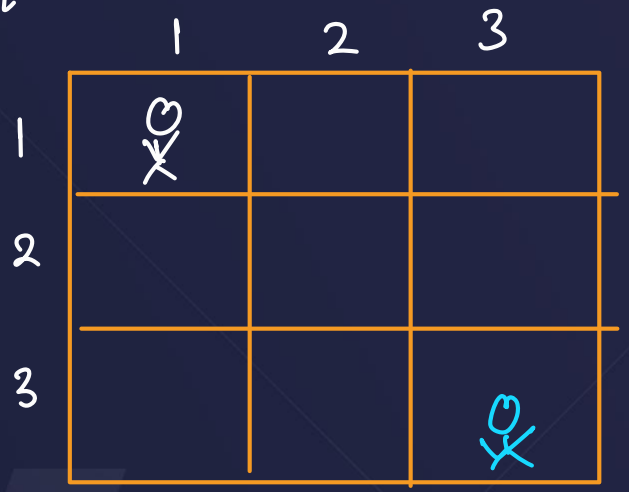
$n = \log(b) + \log(2)$

Ques : Maze path

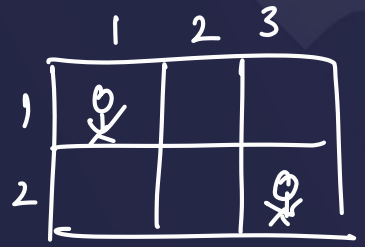
no. of ways ⟶ 'Down, Right'

'1 step at a time'

n, m

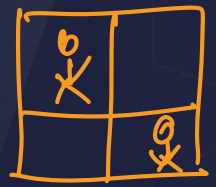


DDRR
DRDR
DRRD
RRDD
RDRD
RDDR



DRR
RRD
RDR

way - 1



DR
RD

Ques : Maze path

n = 3, m = 3

way - 2

Down → (n-1, m)

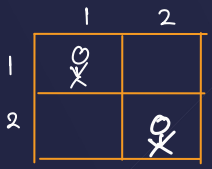
Right → (n, m-1)

```
if (n == 1 && m == 1) return 1
if (n==1){ // Can't go down
   rightways += 
3
```

SKILLS

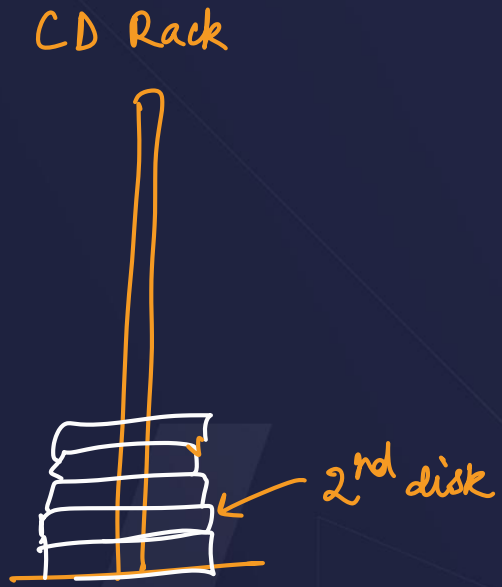
Homework

```
int maze(int cr, int cc, int er, int ec){
    int rightWays = 0;
    int downWays = 0;
    if(cr==er && cc==ec) return 1;
    if(cr==er){ // only rightWays call
        rightWays += maze(cr,cc+1,er,ec);
    }
    if(cc==ec){ // only downwards call
        downWays += maze(cr+1,cc,er,ec);
    }
    if(cr<er && cc<ec){
        rightWays += maze(cr,cc+1,er,ec);
        downWays += maze(cr+1,cc,er,ec);
    }
    int totalWays = rightWays + downWays;
    return totalWays;
}
```




```
int maze(int cr, int cc, int er, int ec){
    int rightWays = 0;
    int downWays = 0;
    if(cr==er && cc==ec) return 1;
    if(cr==er){ // only rightWays call
        rightWays += maze(cr,cc+1,er,ec);
    }
    if(cc==ec){ // only downwards call
        downWays += maze(cr+1,cc,er,ec);
    }
    if(cr<er && cc<ec){
        rightWays += maze(cr,cc+1,er,ec);
        downWays += maze(cr+1,cc,er,ec);
    }
    int totalWays = rightWays + downWays;
    return totalWays;
}
```

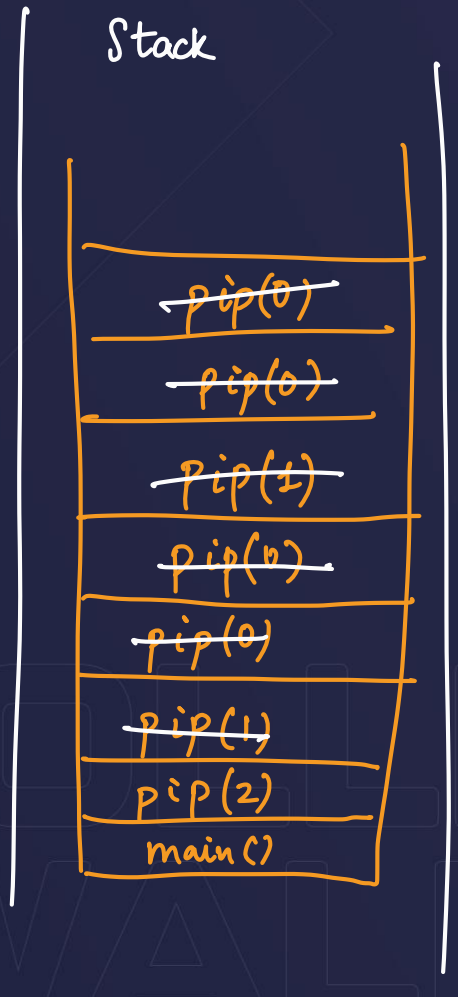
Call Stack



Pre In Post

n = 2

```
void preInPost(int n){
    if(n==0) return;
    printf("Pre %d\n",n);
    preInPost(n-1);
    printf("In %d\n",n);
    preInPost(n-1);
    printf("Post %d\n",n);
    return;
}
```



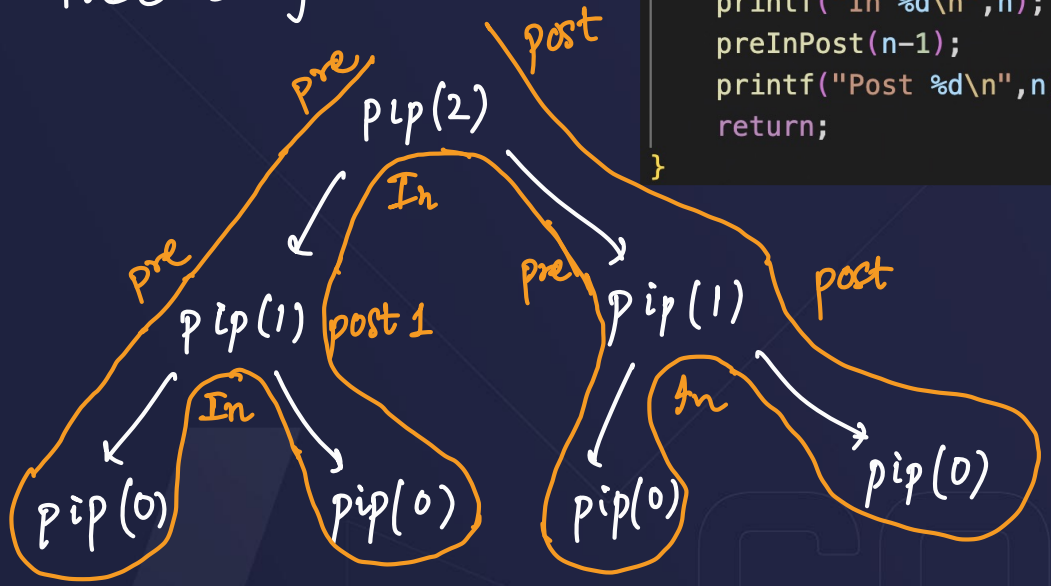
Output

- Pre 2
- Pre 1
- In 1
- Post 1
- In 2
- Pre 1
- In 1
- Post 1
- Post 2

Pre In Post

```
void preInPost(int n){
    if(n==0) return;
    printf("Pre %d\n",n);
    preInPost(n-1);
    printf("In %d\n",n);
    preInPost(n-1);
    printf("Post %d\n",n);
    return;
}
```

Tree Diagram



Output

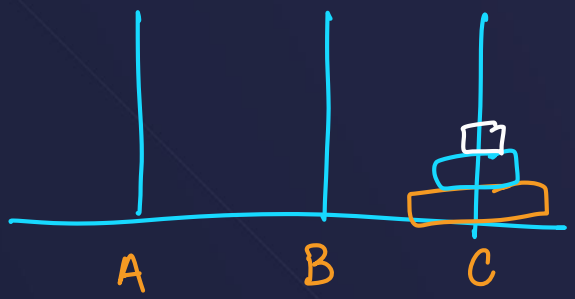
- Pre 2
- Pre 1
- In 1
- Post 1
- In 2
- Pre 1
- In 1
- Post 1
- Post 2

Ques : Print zig-zag

Input	Output
1	111
2	211121112
3	32111211123211121112 3
4	4321112111232111211123 4 32111211123211121112 34

Ques : Tower of HANOI

"Last"



Biggest disk to the 3rd rod

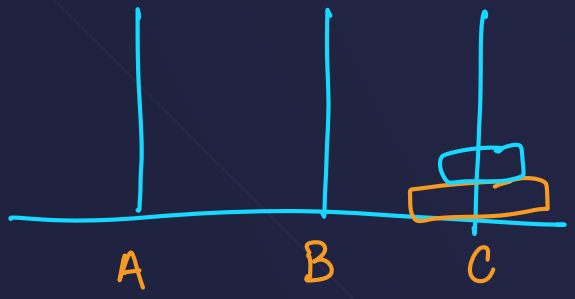
Steps

A → C
A → B
C → B
A → C
B → A
B → C
A → C

Input → n → n. of disks

disks	min moves
3 →	$2^3 - 1 = 7$
4 →	$2^4 - 1 = 15$
5 →	$2^5 - 1 = 31$

Ques : Tower of HANOI



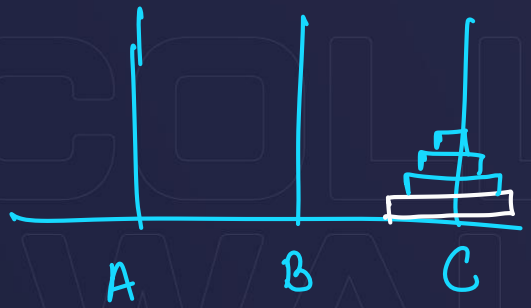
A → B

A → C

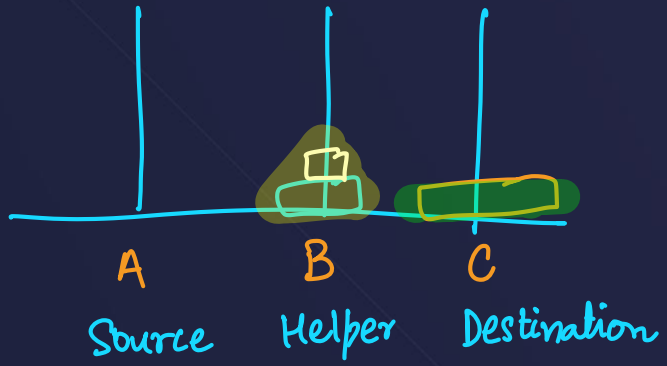
B → C

n = 2

m. moves → $2^2 - 1 = 3$



Ques : Tower of HANOI



n-1 disks

S → H Small pyramid // call

S → D largest disk

H → D Small pyramid // call

Ques : Tower of HANOI