

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 \rightarrow Factorial of a number

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

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

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

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

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

Function calling itself

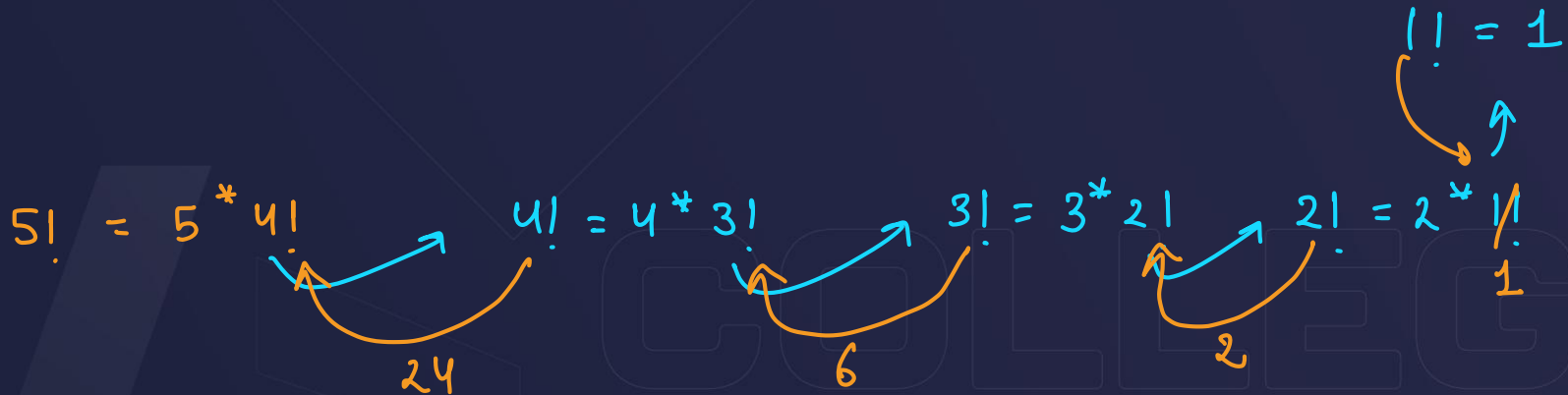
```
int factorial (int n){
```

```
    return n * factorial(n-1);
```

```
}
```

$$\text{factorial}(n) = n * \text{factorial}(n-1);$$

$$n! = n * (n-1)!$$



The diagram illustrates the recursive calculation of 5! using the formula $n! = n * (n-1)!$. It shows a sequence of equations connected by arrows that indicate the flow of recursive calls and returns:

- $5! = 5 * 4!$ (An arrow points from $4!$ to the next equation, and a return arrow points back to 24.)
- $4! = 4 * 3!$ (An arrow points from $3!$ to the next equation, and a return arrow points back to 6.)
- $3! = 3 * 2!$ (An arrow points from $2!$ to the next equation, and a return arrow points back to 2.)
- $2! = 2 * 1!$ (An arrow points from $1!$ to the final result, and a return arrow points back to 1.)
- $1! = 1$ (The base case, with a return arrow pointing back to 1.)

The final results of the recursive calls are shown below each equation: 24, 6, 2, and 1.

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

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

4 × factorial(3);

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

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

*3 * factorial(2);*

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

*2 * factorial(1)*

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

*1 * factorial(0)*

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

*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

3

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

 3 * factorial(2);
 2 = 6

1

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

2

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

 2 * factorial(1);
 1 = 2

Tree Diagram

$$\text{fact}(5) \Rightarrow 5 * \cancel{\text{fact}(4)} \quad 24 \longrightarrow 120$$

$$\downarrow$$

$$4 * \cancel{\text{fact}(3)} \quad 6 = 24$$

$$\downarrow$$

$$3 * \cancel{\text{fact}(2)} \quad 2 = 6$$

$$\downarrow$$

$$2 * \cancel{\text{fact}(1)} \quad 1 = 2$$

$$\downarrow$$

$$1$$

Ques : Print n to 1 using recursion

[6-4]
int decreasing(int n){

}

Output

7
6
5
4
3
2
1

'n' times good morning
[6-3]

```
void greeting(int5 n){
    ✓if(n==0) return;
    ✓printf("Good Morning\n");
    ✓greeting(n-1);
    ✓return;
}
```

✗

```
void greeting(int0 n){
    ✓if(n==0) return;
    printf("Good Morning\n");
    greeting(n-1);
    return;
}
```

✗

```
void greeting(int4 n){
    ✓if(n==0) return;
    ✓printf("Good Morning\n");
    ✓greeting(n-1);
    ✓return;
}
```

✗

```
void greeting(int1 n){
    ✓if(n==0) return;
    ✓printf("Good Morning\n");
    ✓greeting(n-1);
    ✓return;
}
```

✗

```
void greeting(int3 n){
    ✓if(n==0) return;
    ✓printf("Good Morning\n");
    ✓greeting(n-1);
    ✓return;
}
```

✗

```
void greeting(int2 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]


```
func {  
  | base case  
  |  
  | code before recursive call code  
  |  
  | recursive call  
  |  
  | code after recursive call code  
  |  
  | return  
}
```

Ques : Print 1 to n

Output:

n=5

1
2
3
4
5

```
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

COLLEGE
WALLAH

Ques : Print 1 to n (parameterised)

[6-5]

```
#include<stdio.h>
void increasing(1int x, 5int 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);    n = 5
    increasing(1,gn);
    return 0;
}
```

Output

- 1
- 2
- 3
- 4
- 5

COLLEGE
WALLAH

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

```
void increasing(int4 n){
    ✓if(n==0) return; // base case
    ✓increasing(n3); // call
    ✓printf("%d\n",n); // code
    ✓return;
}
```

```
void increasing(int3 n){
    ✓if(n==0) return; // base case
    ✓increasing(n2); // call
    ✓printf("%d\n",n); // code
    ✓return;
}
```

```
void increasing(int0 n){
    ✓if(n==0) return; // base case
    increasing(n-1); // call
    printf("%d\n",n); // code
    return;
}
```

```
void increasing(int1 n){
    ✓if(n==0) return; // base case
    ✓increasing(n0); // call
    ✓printf("%d\n",n); // code
    ✓return;
}
```

```
void increasing(int2 n){
    ✓if(n==0) return; // base case
    ✓increasing(n1); // 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(int4 n, int0 s){
    ✓ if(n==0){
    ✓     printf("%d",s);
    ✓     return;
    ✓ }
    ✓ sum(n3-1, s4+n);
    ✓ return;
}
```

✗

```
void sum(int3 n, int4 s){
    ✓ if(n==0){
    ✓     printf("%d",s);
    ✓     return;
    ✓ }
    ✓ sum(n2-1, s4+3+n);
    ✓ return;
}
```

✗

```
void sum(int1 n, int9 s){
    ✓ if(n==0){
    ✓     printf("%d",s);
    ✓     return;
    ✓ }
    ✓ sum(n0-1, s10+n);
    ✓ return;
}
```

✗

```
void sum(int2 n, int7 s){
    ✓ if(n==0){
    ✓     printf("%d",s);
    ✓     return;
    ✓ }
    ✓ sum(n1-1, s9+n);
    ✓ return;
}
```

✗

```
void sum(int0 n, int10 s){
    ✓ if(n==0){
    ✓     printf("%d",s);
    ✓     return;
    ✓ }
    sum(n-1, s+n);
    return;
}
```

✗

Output
• 10

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

$$\text{factorial}(n) = n * \text{factorial}(n-1);$$

$$\text{sum}(n) = n + \text{sum}(n-1);$$

$$\begin{aligned} \text{sum}(5) &= 1 + 2 + 3 + 4 + 5 = 5 + \text{sum}(4) \\ &\quad \downarrow \\ &\quad 4 + \text{sum}(3) \\ &\quad \quad \downarrow \\ &\quad \quad 3 + \text{sum}(2) \\ &\quad \quad \quad \downarrow \\ &\quad \quad \quad 2 + \text{sum}(1) \\ &\quad \quad \quad \quad \downarrow \\ &\quad \quad \quad \quad 1 \end{aligned}$$

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;
```

```
}
```

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

if (b==0) return 1;

power(a,b) = a * power(a,b-1);

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

$$2^4 = 2 * 2^3 \rightarrow 2^3 = 2 * 2^2 \rightarrow 2^2 = 2 * 2^1 \rightarrow 2^1 = 2 * 2^0$$

1

*Multiple Calls

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

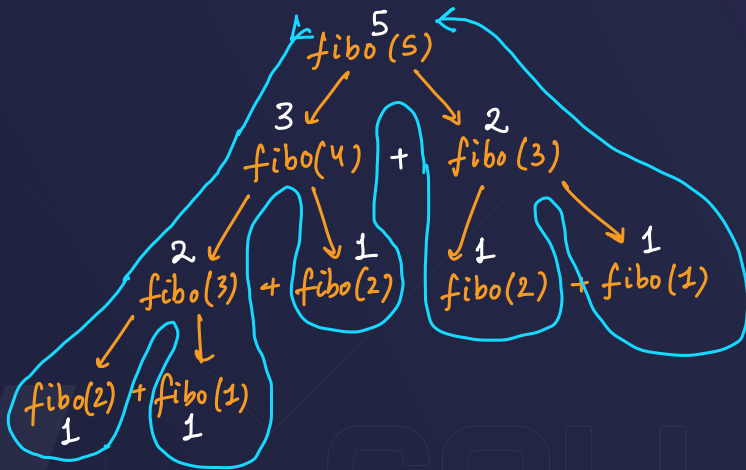
1	1	2	3	5	8	13	21	34	55	89	...
1	2	3	4	5	6	7	8	9	10	11	

$$\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2);$$

if $(n==1 \text{ or } n==2)$ return 1

COLLEGE
WALLAH

$(n==1 \parallel n==2)$
return 1

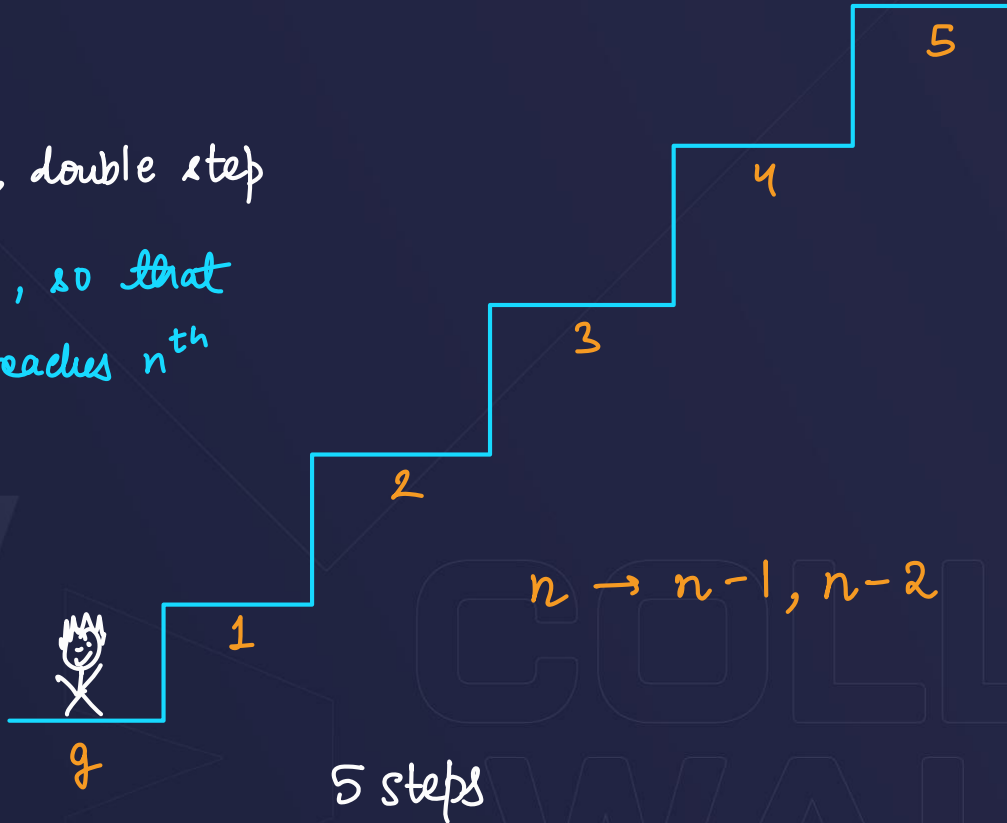


Ques : Stair Path - 1

n^{th} stair

single step, double step

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



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

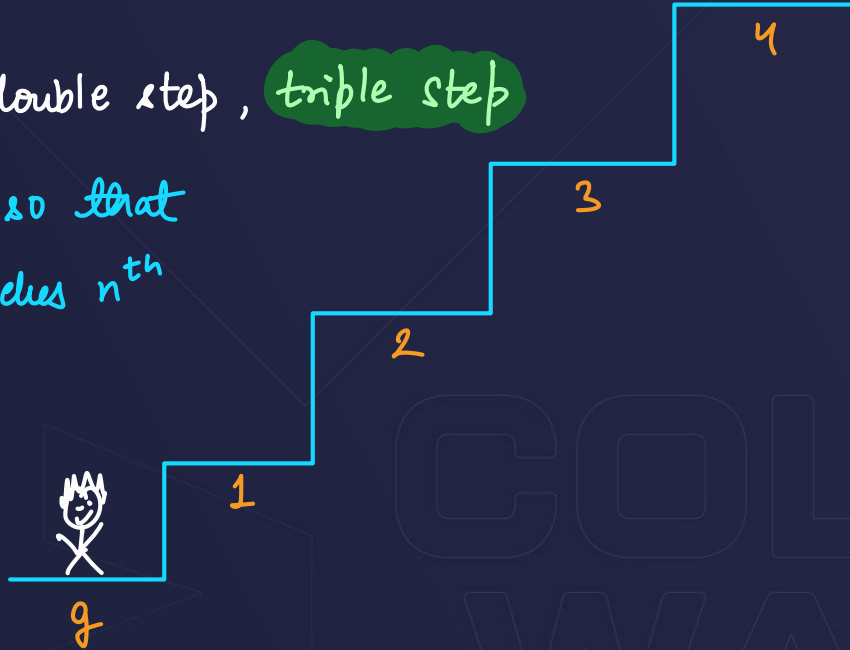
H.W.

n^{th} stair

[6-12]

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		

COLLEGE
WALLAH

Ques : Power function (logarithmic)

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

if (b == 0) return 1;

$$\text{pow}(a, b) = \text{pow}(a, b/2) * \text{pow}(a, b/2)$$

$$\begin{aligned} 2^{64} &= 2 * 2^{63} \\ 2^{63} &= 2 * 2^{62} \\ &\vdots \\ 2^1 &= 2 * 2^0 \end{aligned}$$

64
calls

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

$$\begin{aligned} 2^{64} &= 2^{32} * 2^{32} \\ 2^{32} &= 2^{16} * 2^{16} \\ 2^{16} &= 2^8 * 2^8 \\ 2^8 &= 2^4 * 2^4 \\ 2^4 &= 2^2 * 2^2 \\ 2^2 &= 2^1 * 2^1 \end{aligned}$$

6
calls

if (b == 1) return a;

Ques : Power function (logarithmic)

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

Solution:

Problem:

if b is even

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

if b is odd

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

$$\begin{aligned} 2^7 &= 2^{7/2} \times 2^{7/2} \\ \hookrightarrow 2^7 &= \underbrace{2^3}_{4} \times \underbrace{2^3}_{4} = 16 \end{aligned}$$

$$b=5$$

$$\begin{aligned} a^5 &= a^{5/2} \times a^{5/2} \times a \\ &= \underbrace{a^2 \times a^2 \times a} \end{aligned}$$

$$\begin{aligned} 2^3 &= 2^{3/2} \times 2^{3/2} \\ \hookrightarrow 2^3 &= 2^1 \times 2^1 \\ 2^3 &= 2 \times 2 = 4 \end{aligned}$$

```
#include <stdio.h>

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;
}
```

$$a = 2$$

$$b = 9$$

$$a^b \Rightarrow \underbrace{b \text{ calls}}$$

$$2^{100} = 100 \text{ calls}$$

$$2^9 = \underline{2^4} * 2^4 * 2;$$

$$2^4 = \underline{2^2} * 2^2;$$

$$2^2 = \underline{2^1} * 2^1;$$

$$2^1 = \underline{2^0} * 2^0 * 2;$$

↓
1

$$b \rightarrow b/2 \rightarrow b/4 \rightarrow b/8 \rightarrow \dots \rightarrow 1$$

$$a^b \rightarrow b/\text{calls}$$

$$\log_2(b) \text{ calls}$$

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

Diagram illustrating the sequence of terms in the geometric progression:

$$1 \xrightarrow{\times 2} 2 \quad 4 \quad \dots \quad \frac{b}{4} \quad \frac{b}{2} \quad \boxed{b}$$

n terms

$$n = \boxed{\log(b) + \log(2)}$$

$$\boxed{b = a_n} \rightarrow$$

$$a_1 = 1$$

$$r = 2$$

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

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

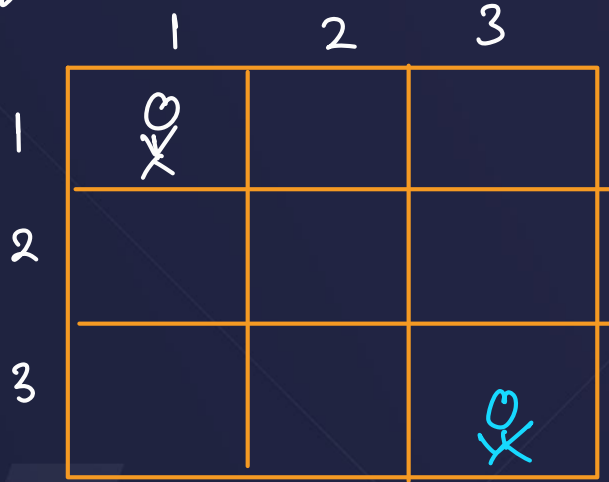
$$\rightarrow b = \frac{2^n}{2} \Rightarrow 2^n = 2b$$

$$\Rightarrow \boxed{n = \log_2 2b}$$

Ques : Maze path $\xrightarrow{\text{no. of ways}}$ 'Down, Right'

'1 step at a time'

n, m



DDRR

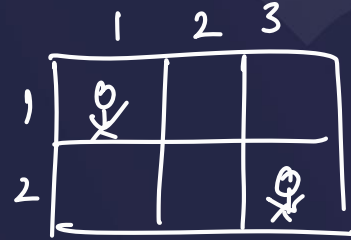
DRDR

DRRD

RRDD

RDRD

RDDR

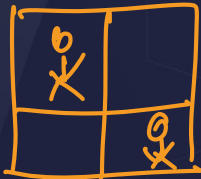


DRR

RRD

RDR

way - 1



DR

RD

Ques : Maze path

$n=3, m=3$



way - 2

Down $\rightarrow (n-1, m)$

Right $\rightarrow (n, m-1)$

if $(n==1 \ \& \ m==1)$ return 1

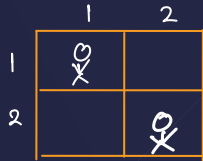
if $(n==1) \& \ // \text{ Can't go down}$

rightwayl +=

3

Homework

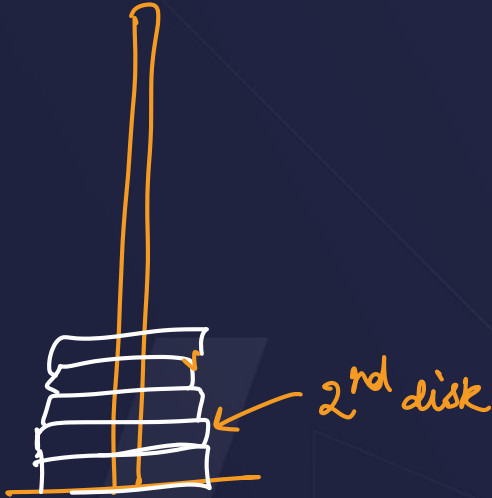
```
int maze(int1 cr, int1 cc, int2 er, int2 ec){
    int rightWays = 0;
    int downWays = 0;
    if(cr==er && cc==ec) return 1;
    if(cr==er){ // only rightWays call
        rightWays += maze(cr,cc+1,er,ec);
    }
    if(cc==ec){ // only downwards call
        downWays += maze(cr+1,cc,er,ec);
    }
    if(cr<er && cc<ec){
        1 ✓ rightWays += maze1 2 2 2(cr,cc+1,er,ec);
        downWays += maze(cr+1,cc,er,ec);
    }
    int totalWays = rightWays + downWays;
    return totalWays;
}
```



```
int maze(int1 cr, int2 cc, int2 er, int2 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){
        2 2 2 2
        rightWays += maze(cr,cc+1,er,ec);
        downWays += maze(cr+1,cc,er,ec);
    }
    0 1
    int totalWays = rightWays + downWays;
    return totalWays;
}
```

Call Stack

CD Rack



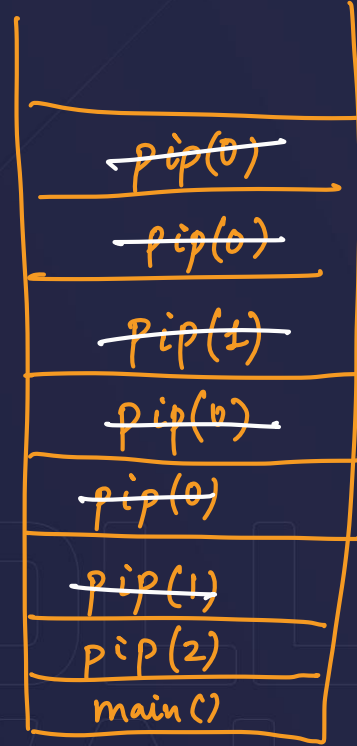
COLLEGE
WALLAH

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;
}
```

Stack



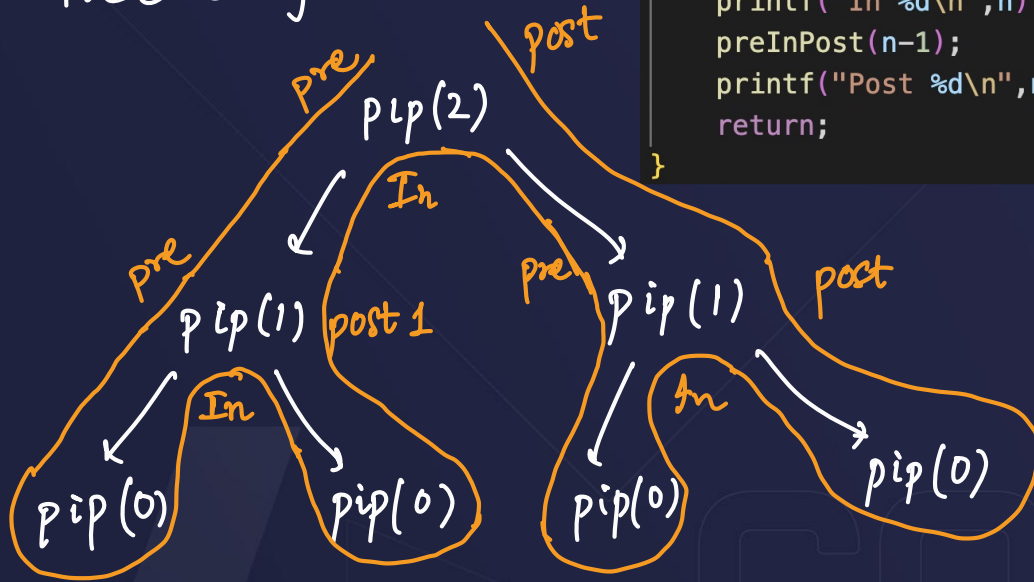
Output

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

Pre In Post

Tree Diagram

```
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

Ques : Print zig-zag

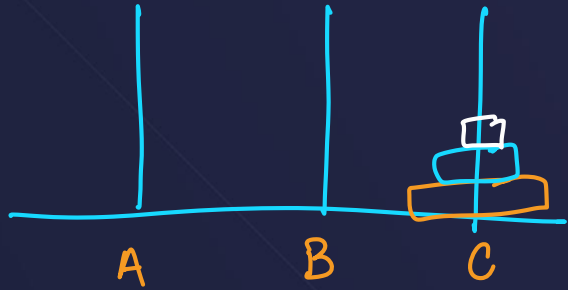
Input **Output**

1 111
 2 211121112
 3 321112111232111211123

4 432111211123211121112343211121112321112111234

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

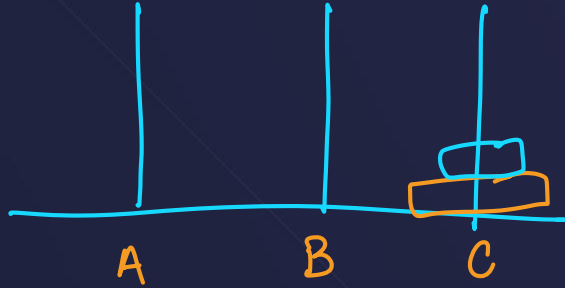
min moves

$$3 \rightarrow 2^3 - 1 = 7$$

$$4 \rightarrow 2^4 - 1 = 15$$

$$5 \rightarrow 2^5 - 1 = 31$$

Ques : Tower of HANOI



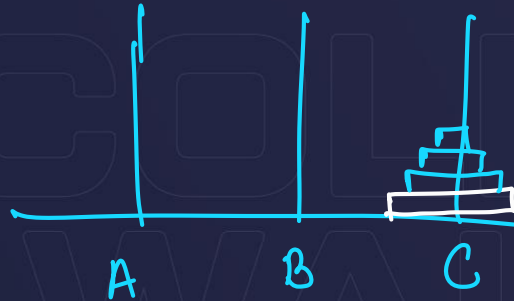
$A \rightarrow B$

$A \rightarrow C$

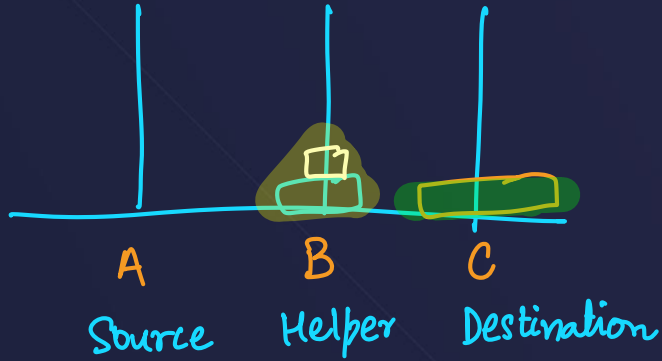
$B \rightarrow C$

$$n = 2$$

$$\text{m. moves} \rightarrow 2^2 - 1 = 3$$



Ques : Tower of HANOI



$S \rightarrow H$

$S \rightarrow D$

$H \rightarrow D$

small pyramid // call

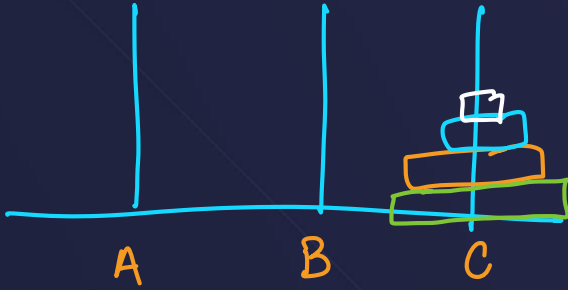
largest disk

small pyramid // call

$n-1$ disks
↑

COLLEGE
WALLAH

Ques : Tower of HANOI



COLLEGE
WALLAH