Recursion - 2

Lecture-28

Raghav Garg

*Multiple Calls

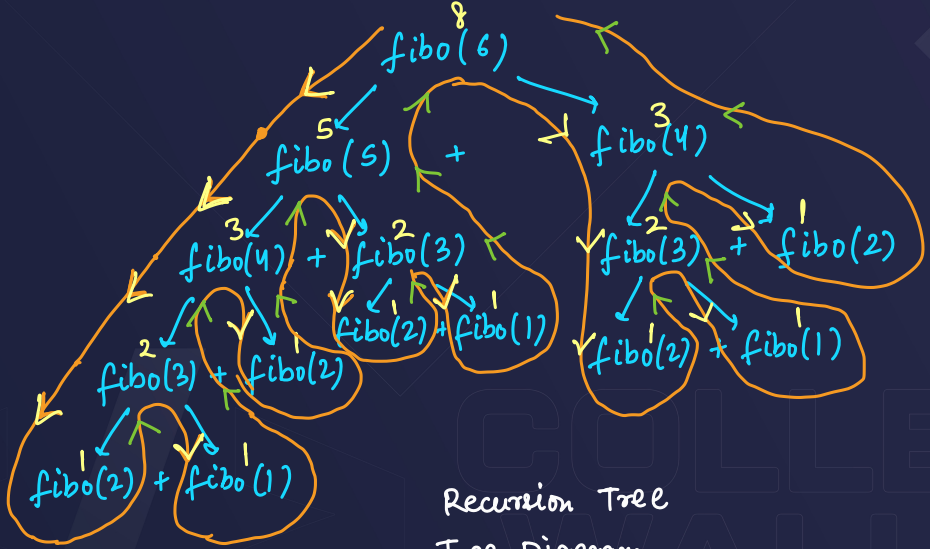
Ques : Write a function to calculate the nth fibonacci number using recursion.

recurrence relation / formula

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

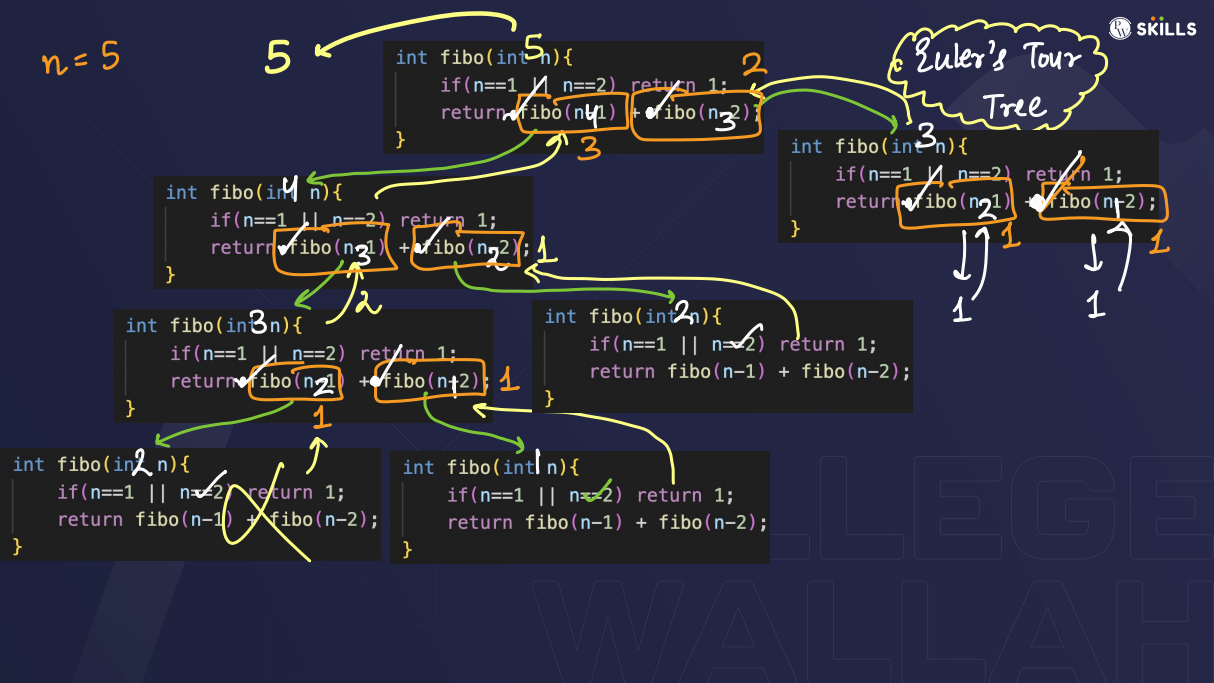
```
if(n==1 || n==2) return 1;
return fibo(n-1) + fibo(n-2);
```

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



Recursion Tree

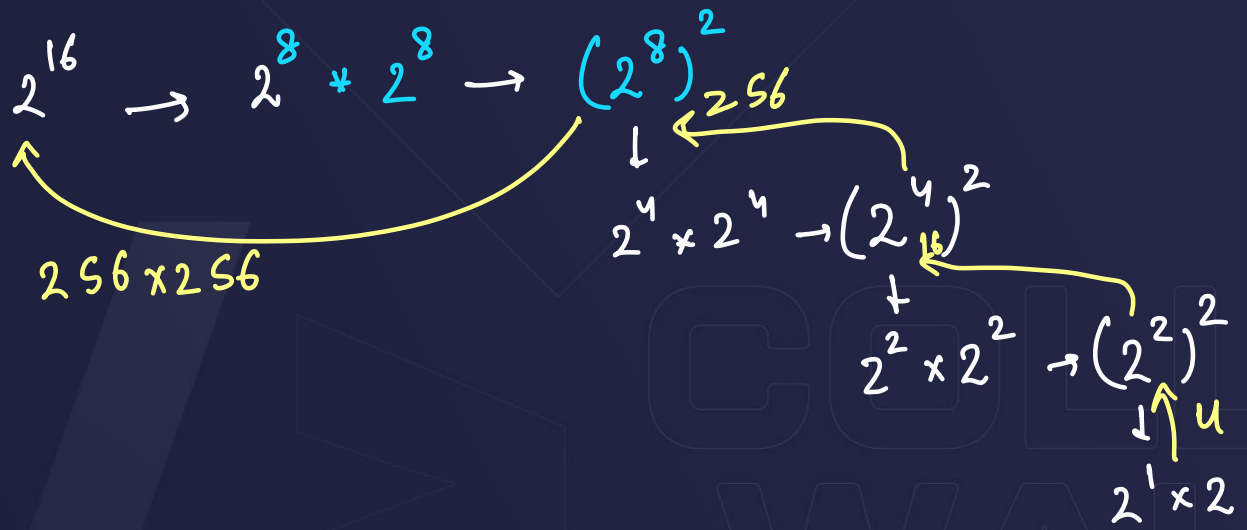
Tree Diagram


n = 5
5
int fibo(int n){
if(n==1 || n==2) return 1;
return fibo(n-1) + fibo(n-2);
}
Euler's Tour Tree
SKILLS

Ques : Power function (logarithmic)

$x^n = x * x^{n-1}$

pow(x, n) = x * pow(x, n-1) → T.C. = O(n)



Ques : Power function (logarithmic)

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

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

.
.
.
:

$2^{2} = 2 * 2^{1}$

$2^{1} = 2 * 2^{0}$

1

64 calls

$64 \to 63 \to 62 \to 61 \to \cdots \to 3 \to 2 \to 1 \to 0$

Method-2

$2^{64} = 2^{32} \times 2^{32}$

$2^{32} = 2^{16} \times 2^{16}$

$2^{16} = 2^{8} \times 2^{8}$

$2^{8} = 2^{4} \times 2^{4}$

$2^{4} = 2^{2} \times 2^{2}$

$2^{2} = 2^{1} \times 2^{1}$

$64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$

$\log_2(64)$

$2^{x} = n$

$x = \log_2 n$

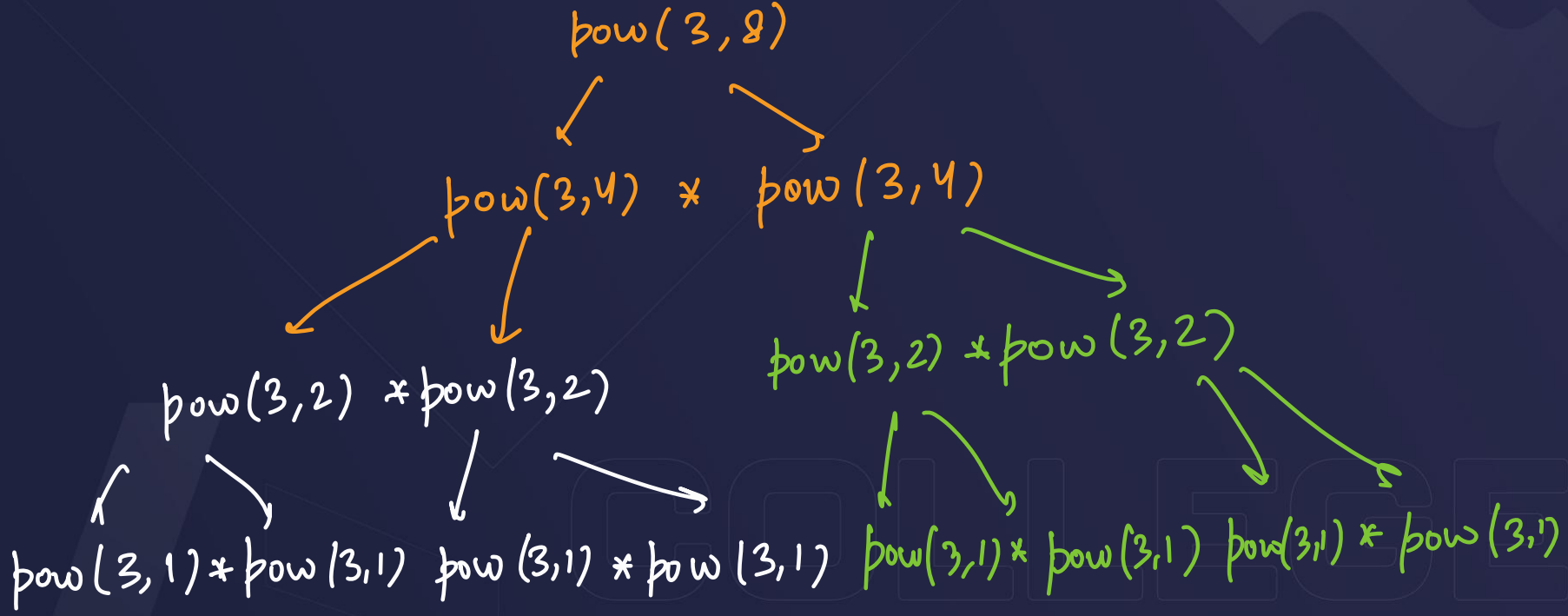
Ques : Power function (logarithmic)

formula :

$pow(x,n) = pow(x,n/2) * pow(x,n/2);$

$x^n = x^{n/2} \times x^{n/2}$

Ques : Power function (logarithmic)

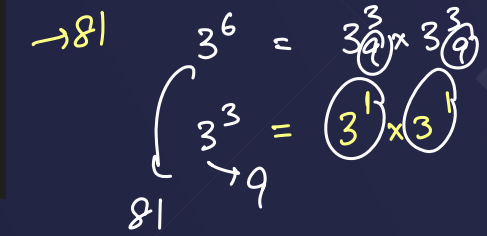


$1 + 2 + 4 + 8 + 16 \ldots n = \boxed{2n - 1}$

```
int pow(int x, int n){
    if(n==1) return x;
    int ans = pow(x,n/2);
    return ans*ans;
}
```

```
int pow(int x, int n){
    if(n==1) return x;
    int ans = pow(x,n/2);
    return ans*ans;
}
```

```
int pow(int x, int n){
    if(n==1) return x;
    int ans = pow(x,n/2);
    return ans*ans;
}
```



$2^{100} = 2^{50} \times 2^{50}$ ✓

$2^{50} = 2^{25} \times 2^{25}$ ✓

$2^{25} = 2^{12} \times 2^{12} \times 2$ ✓✓

$2^{12} = 2^{6} \times 2^{6}$ ✓

$2^{6} = 2^{3} \times 2^{3}$ ✓

$2^{3} = 2^{1} \times 2^{1} \times 2$ ✓✓

formula :

if $(n \% 2 == 0)$

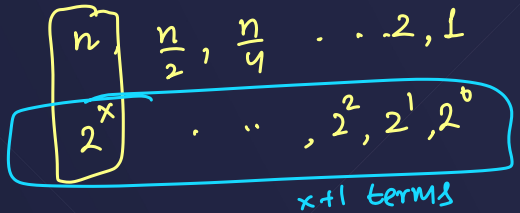
$pow(x,n) = pow(x, n/2) * pow(x, n/2)$

if $(n \% 2 != 0)$

$pow(x,n) = pow(x, n/2) * pow(x, n/2) * x$

Time Complexity

$x^n \rightarrow$



$T.C. = O(\log n)$

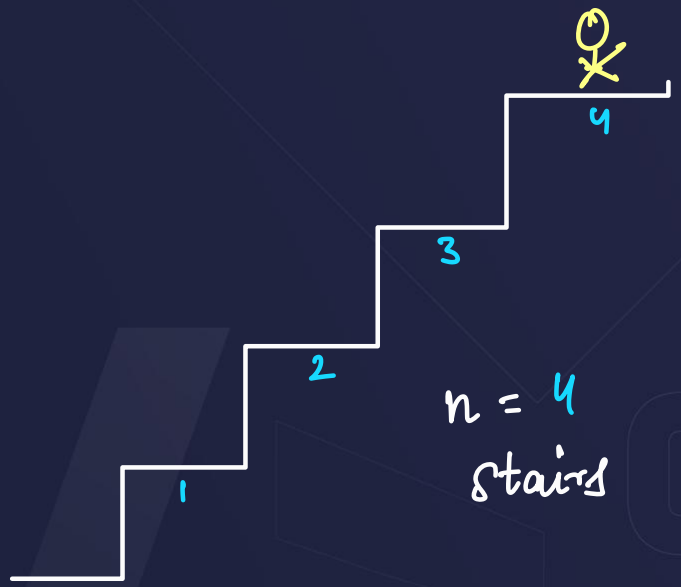
$S.C. = O(\log n)$

$2^x = n$

$\Rightarrow x = \log_2 n$

Ques : Stair Path

Either one step or 2 step and their combinations



No. of ways :→ 5

1 1 1 1

1 1 2

1 2 1

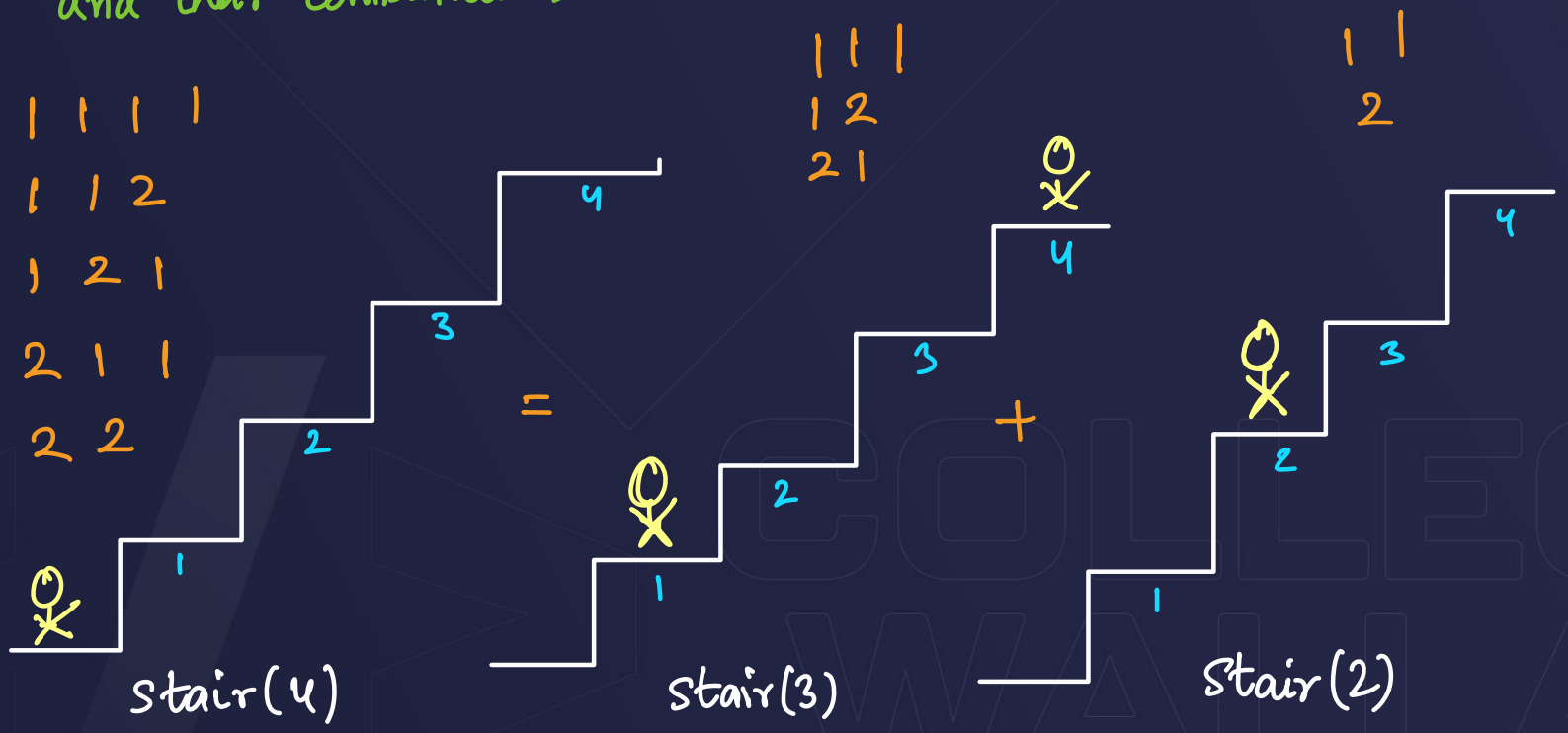
2 1 1

2 2

Ques : Stair Path

Either one step or 2 step and their combinations

Choices → ?



Ques : Stair Path

formula :

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

base case : $stair(2) = 2$

$stair(1) = 1$

n :	1	2	3	4	5	6	7
	1	2	3	5	8	13	21

Ques : Stair Path

1 1 1 1 1

1 1 1 2

1 1 2 1

1 2 1 1

1 2 2

2 1 1 1

2 1 2

2 2 1

8 ways

Ques : Maze path

6 ways

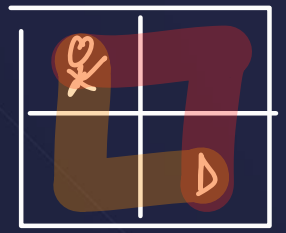
RRDD DRRD
RDRD DRDR
RDDR DDRR

2 - directions
- Right
- Down

Base Case if Q reaches D
then return 1;

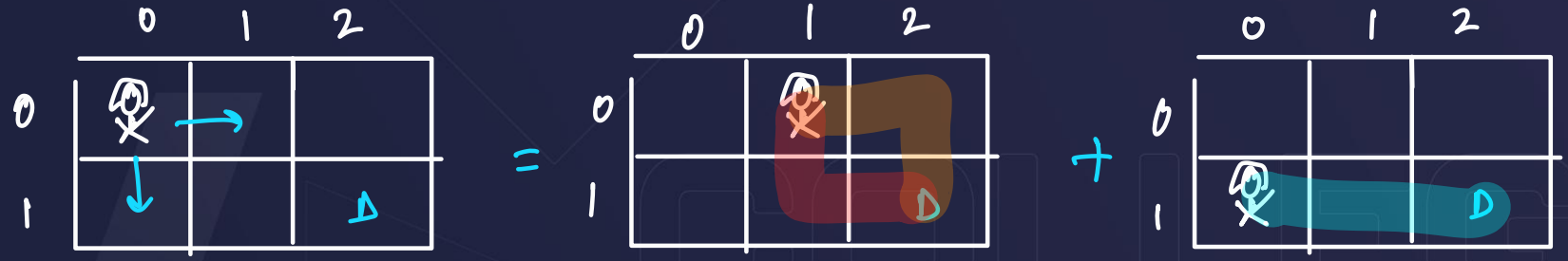
if Q escaps out of matrix
then return 0;

Ques : Maze path



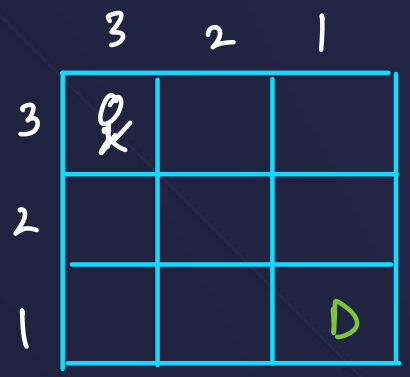
RD

DR



(0,0) → (1,2) = (0,1) → (1,2) + (1,0) → (1,2)

Ques : Maze path



Right col - 1

Down row - 1

* Pre In Post (VVVIMP)

Predict the Output

Kaam → Pre

Call 1

Kaam → 1n

Call 2

Kaam → Post

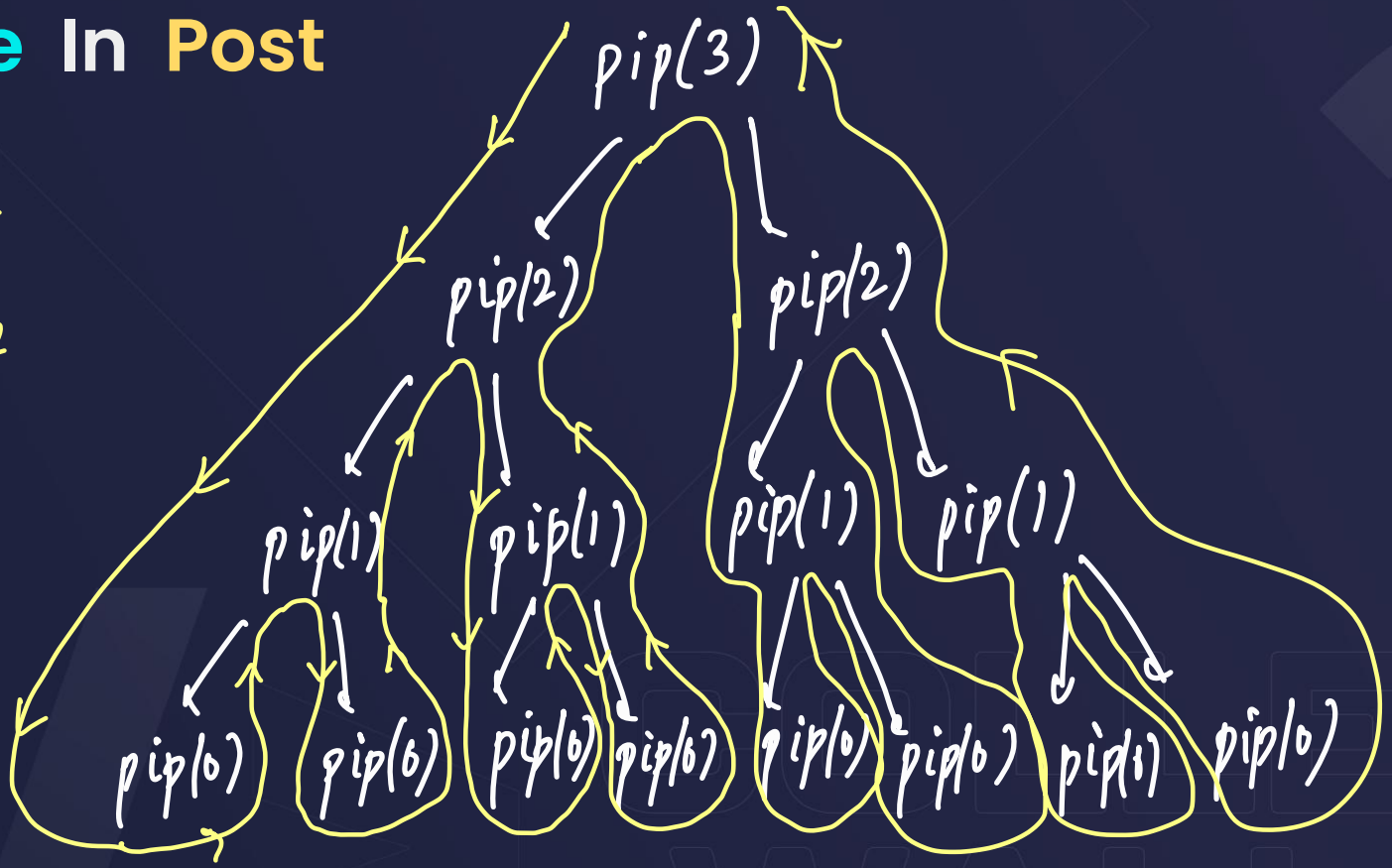
```
void pip(int n){
    if(n==0) return;
    cout<<"Pre "<<n<<endl;
    pip(n-1);
    cout<<"In "<<n<<endl;
    pip(n-1);
    cout<<"Post "<<n<<endl;
}
```

Output

```
Pre 3
Pre 2
Pre 1
In 1
Post 1
In 2
Pre 1
In 1
Post 1
Post 2
In 3
Pre 2
Pre 1
In 1
Post 1
In 2
Pre 1
In 1
Post 1
Post 2
Post 3
```

Pre In Post

Pre
Call 1
In
Call 2
Post



Pre 3
Pre 2
Pre 1
In 1
Post 1
In 2
Pre 1
In 1
Post 1
Post 2

Call Stack

```
void pip(int n){
1   if(n==0) return;
2   cout<<"Pre "<<n<<endl;
3   pip(n-1);
4   cout<<"In "<<n<<endl;
5   pip(n-1);
6   cout<<"Post "<<n<<endl;
}
```

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	4321112111232111211123432111211123211121112 34

Next Lecture

Recursion on Arrays and Strings

More problems on Recursion