

Recursion Basics

In This Lecture

1. Basics of Recursion ✓
2. Recursion Stack ✓
3. Linear Recursion and Tree Recursion ✓
4. The sum of n Natural Numbers ✓
5. Fibonacci Number ✓

Basics of Recursion

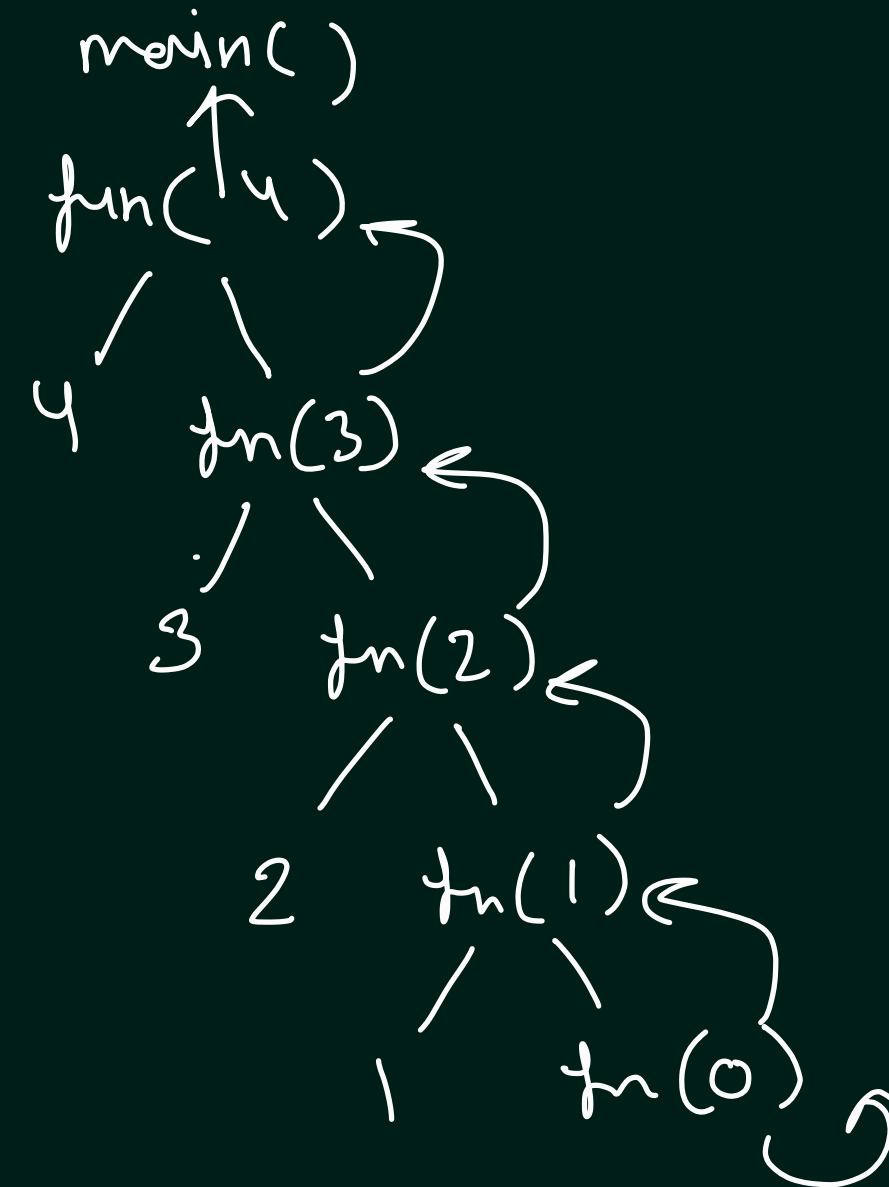
Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

base
case

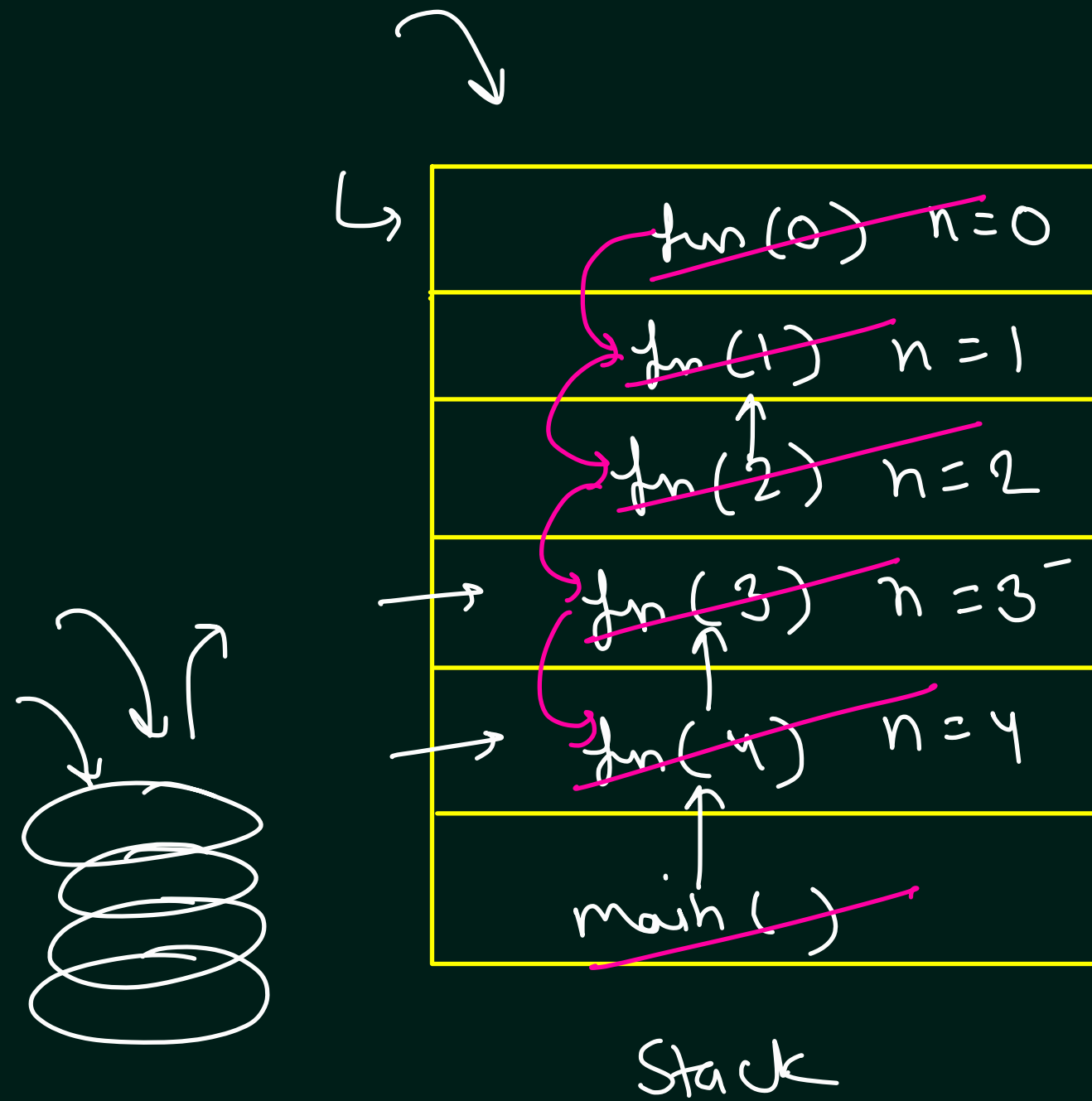
```
2 usages
static void fun(int n) {
1 | if(n > 0) { ←
2 |     System.out.println(n);
3 |     fun(n - 1);
  | }
  }
```

$n = 4/3/2/1/0$

o/p : 4 3 2 1



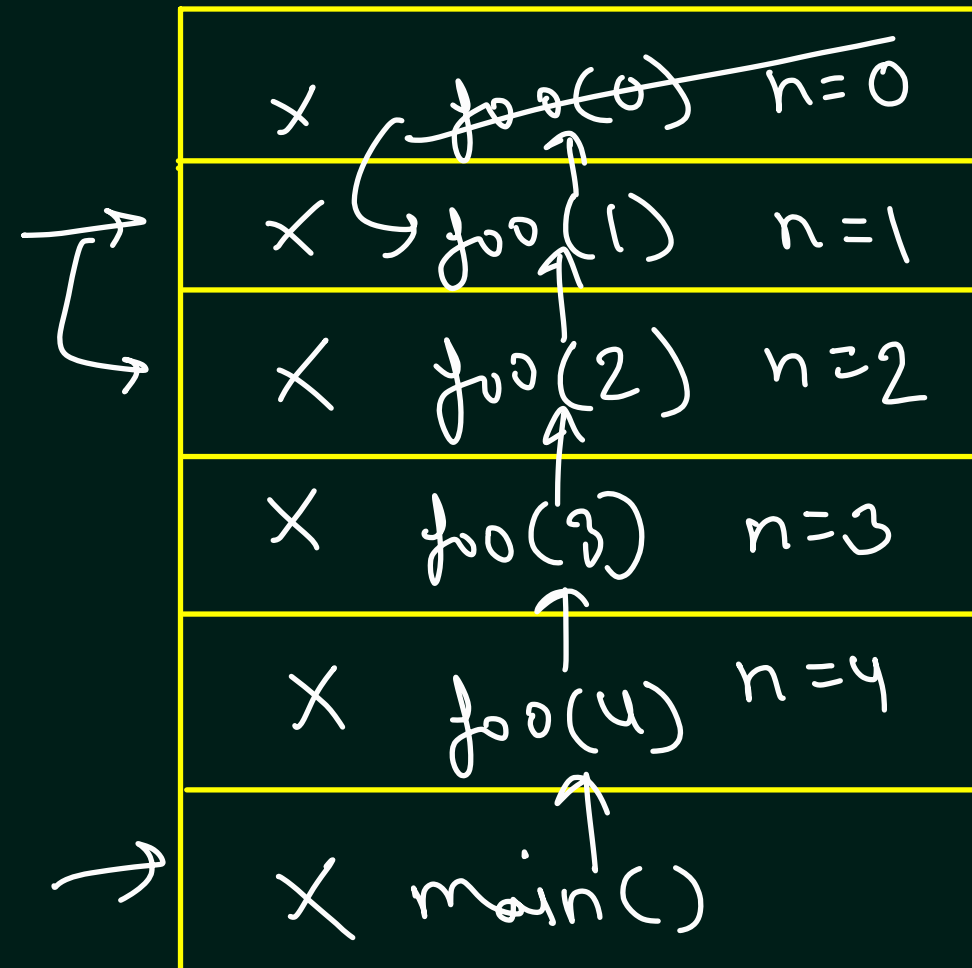
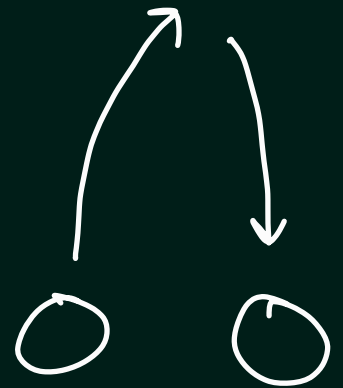
Recursion Stack



op/ : 4 3 2 1

```
2 usages
static void fun(int n) {
    if(n > 0) {
        System.out.println(n);
        fun(n - 1);
    }
}
```

Recursion Stack

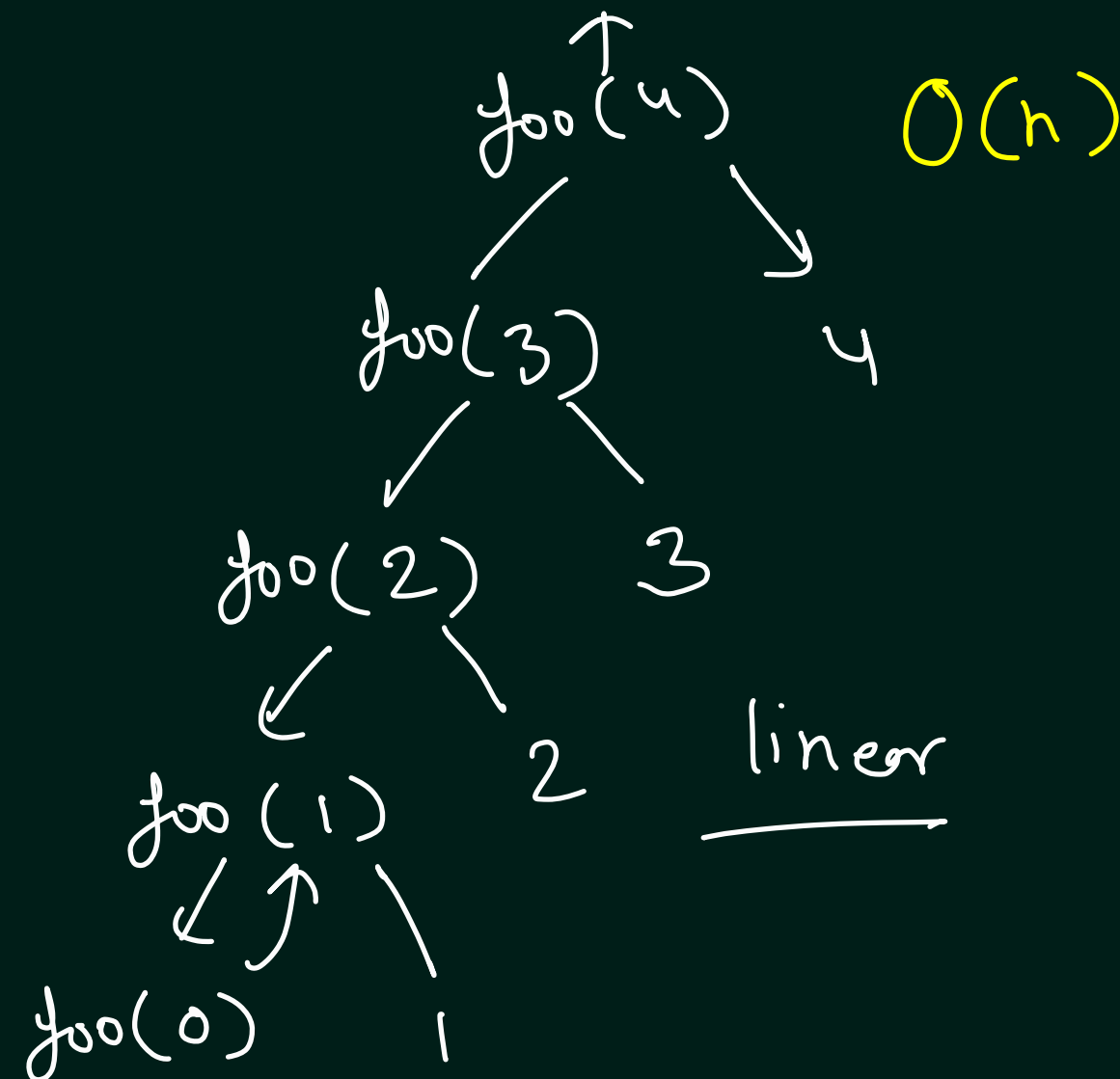


Op: 1 2 3 4

```

main()
  ↓
2 usages
static void foo(int n) {
    if(n > 0) {
        foo(n: n - 1);
        System.out.println(n); ← desc
    }
}
  
```

Arrows show the call from main() to foo(2), and the recursive call from foo(2) to foo(1), and then to foo(0). The return path is indicated by arrows pointing up from each level to the one below it.



Linear Recursion & Tree Recursion

↑

```

fn(n) {
  if ( ) {
    → fn(n-1);
  }
}

```

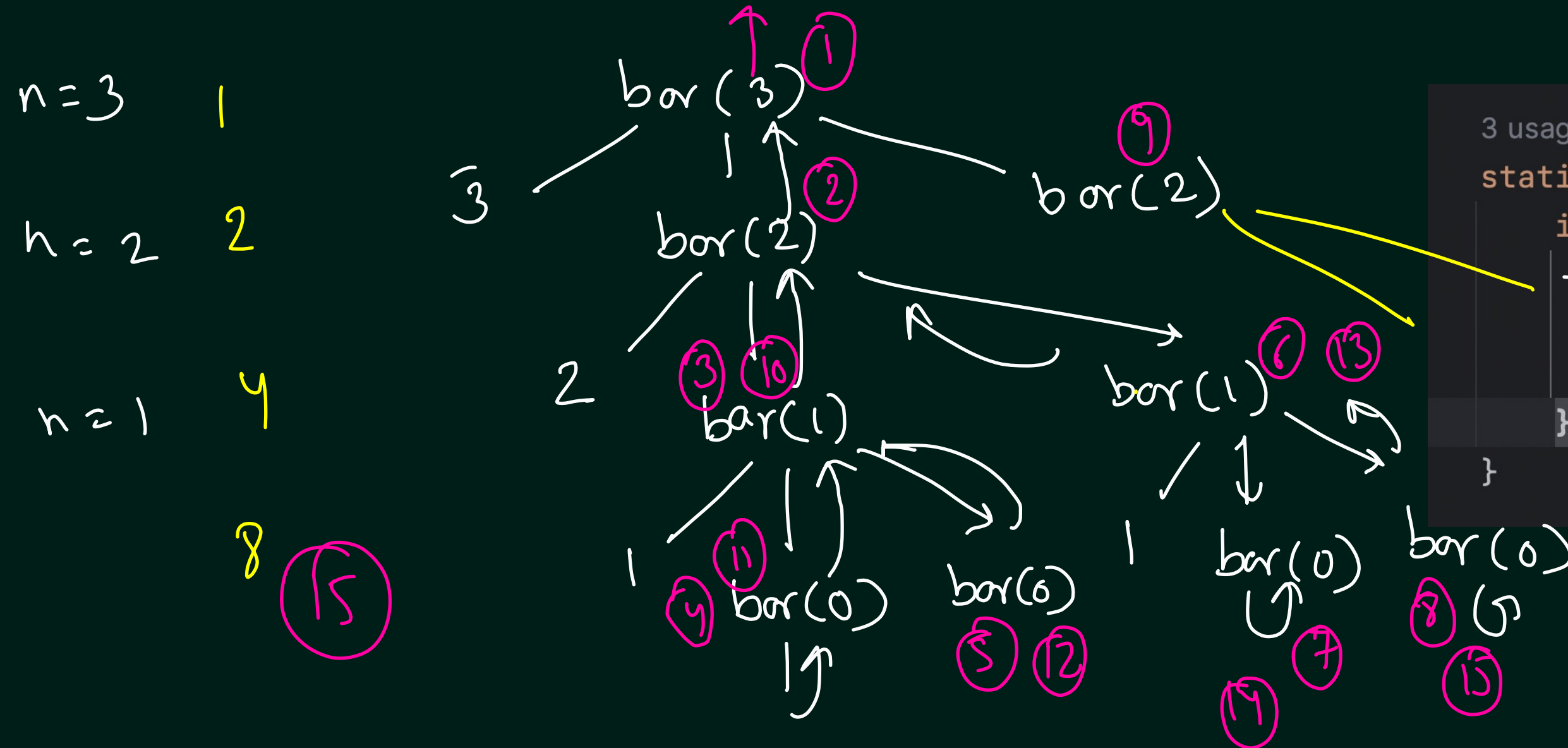
```

fn(n) {
  if ( ) {
    → fn(n-1);
    → fn(n-1);
  }
}

```



Linear Recursion & Tree Recursion



```

3 usages      n=3
static void bar(int n) {
    if(n > 0) {
        → System.out.println(n);
        → bar(n: n-1);
        → bar(n: n-1); bar(2)
    }
}
  
```

$$\rightarrow 2^{n+1} - 1$$

op: 3 2 1 1 2 1 1

$$2^4 - 1 = 15$$

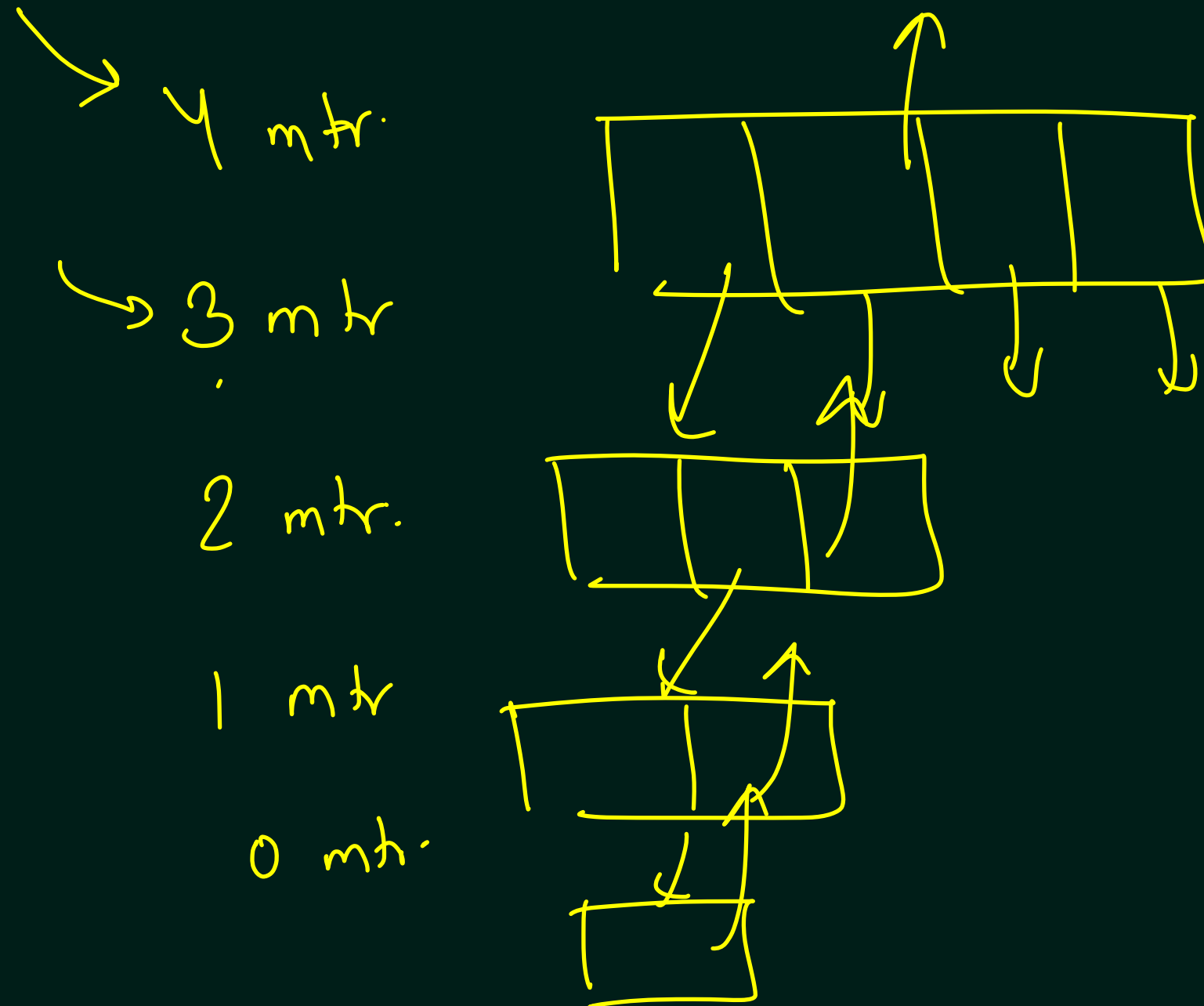
$$[2^0 + 2^1 + 2^2 + 2^3 \dots 2^n = 2^{n+1} - 1] \rightarrow [O(2^n)]$$

3 Steps To Solve Recursion Problem

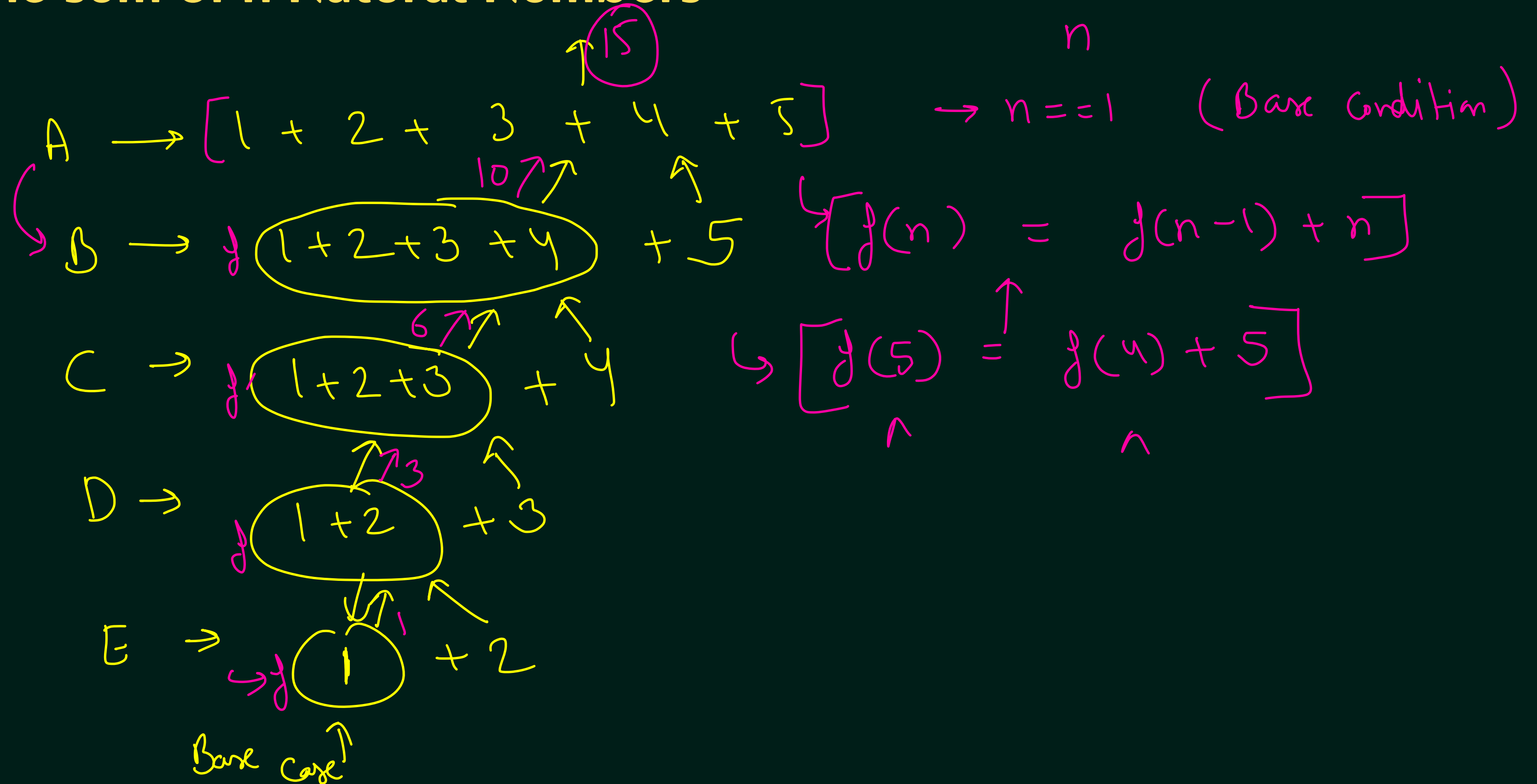
→ Step 1: Find the Base Case

→ Step 2: Find the relation between the Problem and the Sub Problem

Step 3: Generalise the relation found in the step 2



The sum of n Natural Numbers



The sum of n Natural Numbers

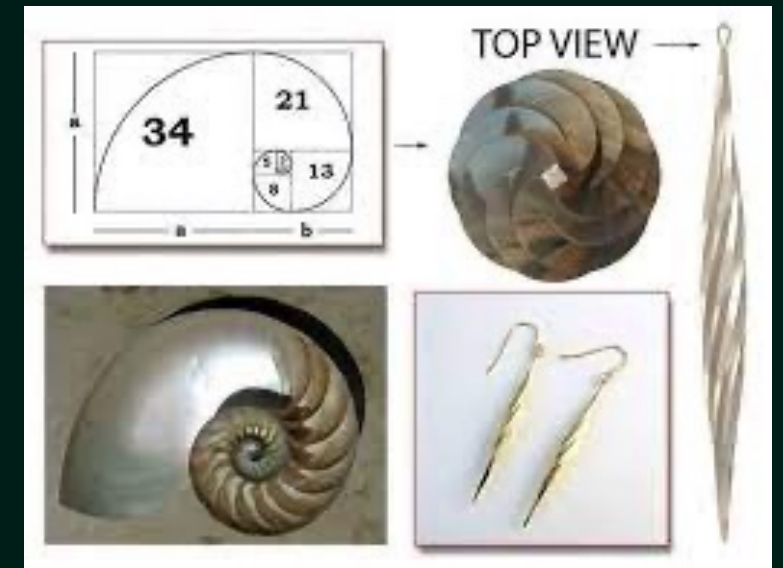
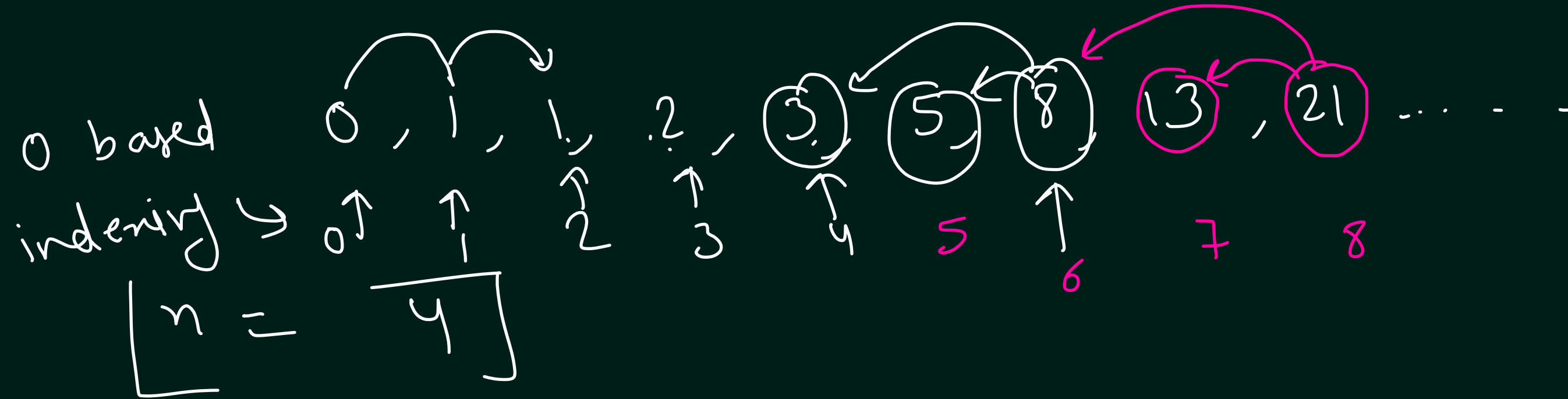
$$\begin{array}{l}
 f(5) + 5 \quad \uparrow 15 \\
 \swarrow \uparrow 10 \\
 f(4) + 4 \\
 \swarrow \uparrow 6 \\
 f(3) + 3 \\
 \swarrow \uparrow 3 \\
 f(2) + 2 \\
 \swarrow \uparrow 1 \\
 f(1)
 \end{array}$$

```

15 → System.out.println(sumOfN(n: 5)); ← main
    }

2 usages      n = 5
static int sumOfN(int n) {
    if(n == 1) return 1;
    return sumOfN(n: n-1) + n;
}
                    10 + 5
  
```

Fibonacci Number

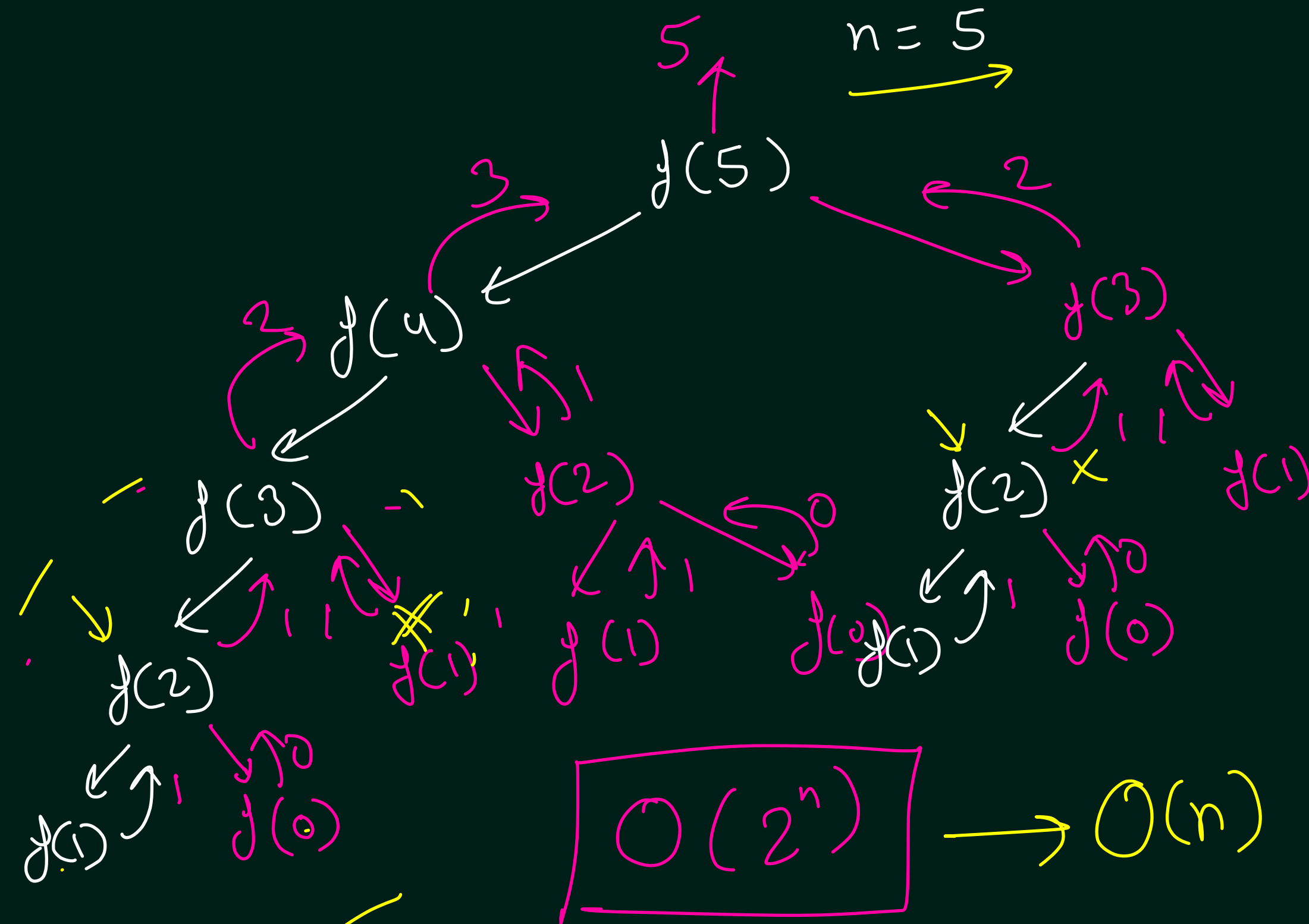


$$\rightarrow f(n) = f(n-1) + f(n-2) \leftarrow$$

$$\begin{aligned} f(8) &= f(7) + f(6) \\ &= 13 + 8 = 21 \end{aligned}$$

$$f(2) = f(1) + f(0) = 1$$





$$\boxed{O(2^n)} \rightarrow O(n)$$

memoization

0	1	1	2	3	5
0	1	2	3	4	5

3 usages

5

```
static int fibOfN(int n) {
    if(n <= 1) return n;
    return fibOfN(n-1) + fibOfN(n-2);
}
```

$$\text{fib}(4) = 3 \quad \text{fib}(3) = 2$$