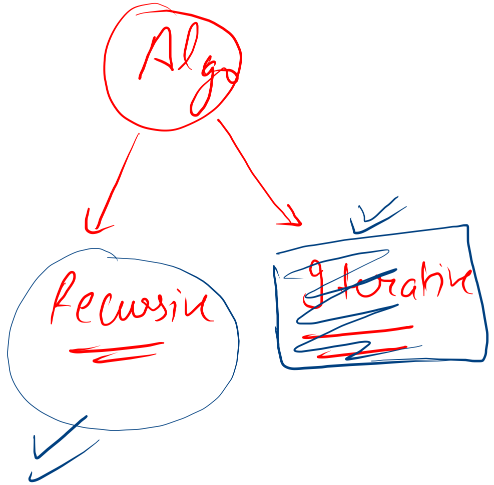
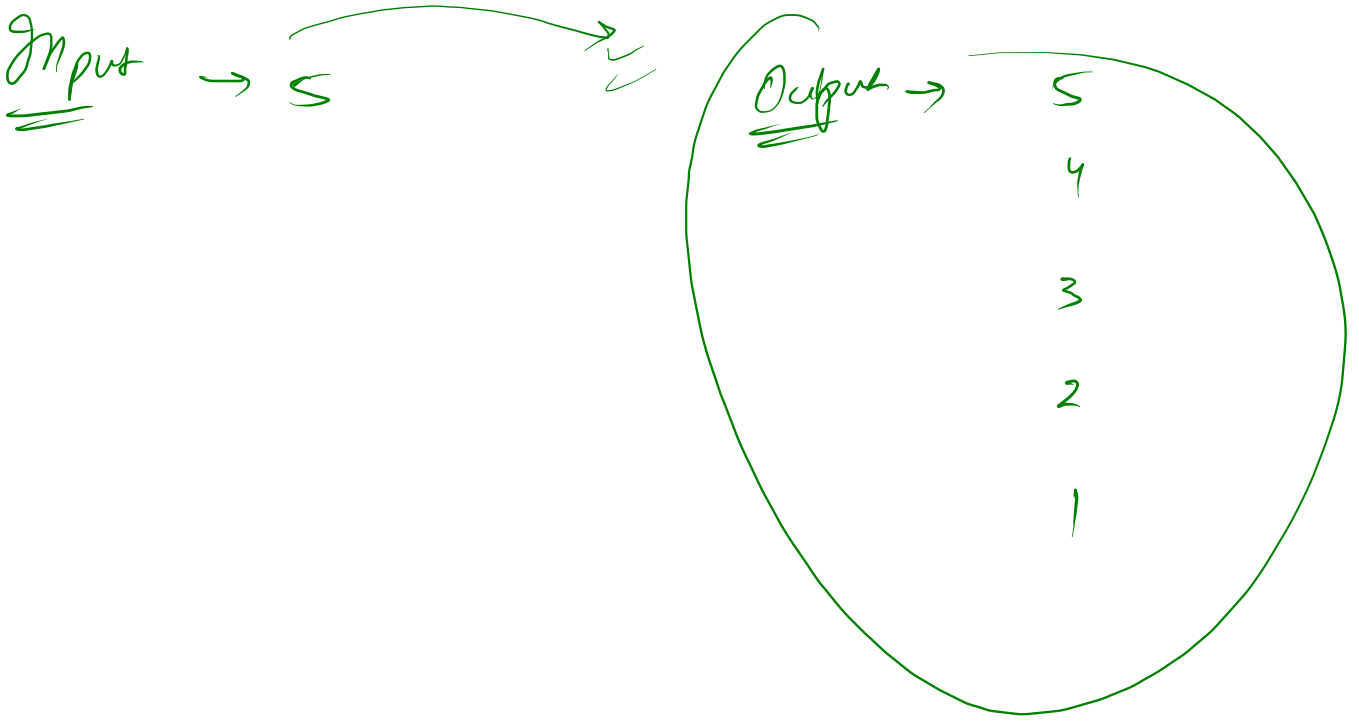
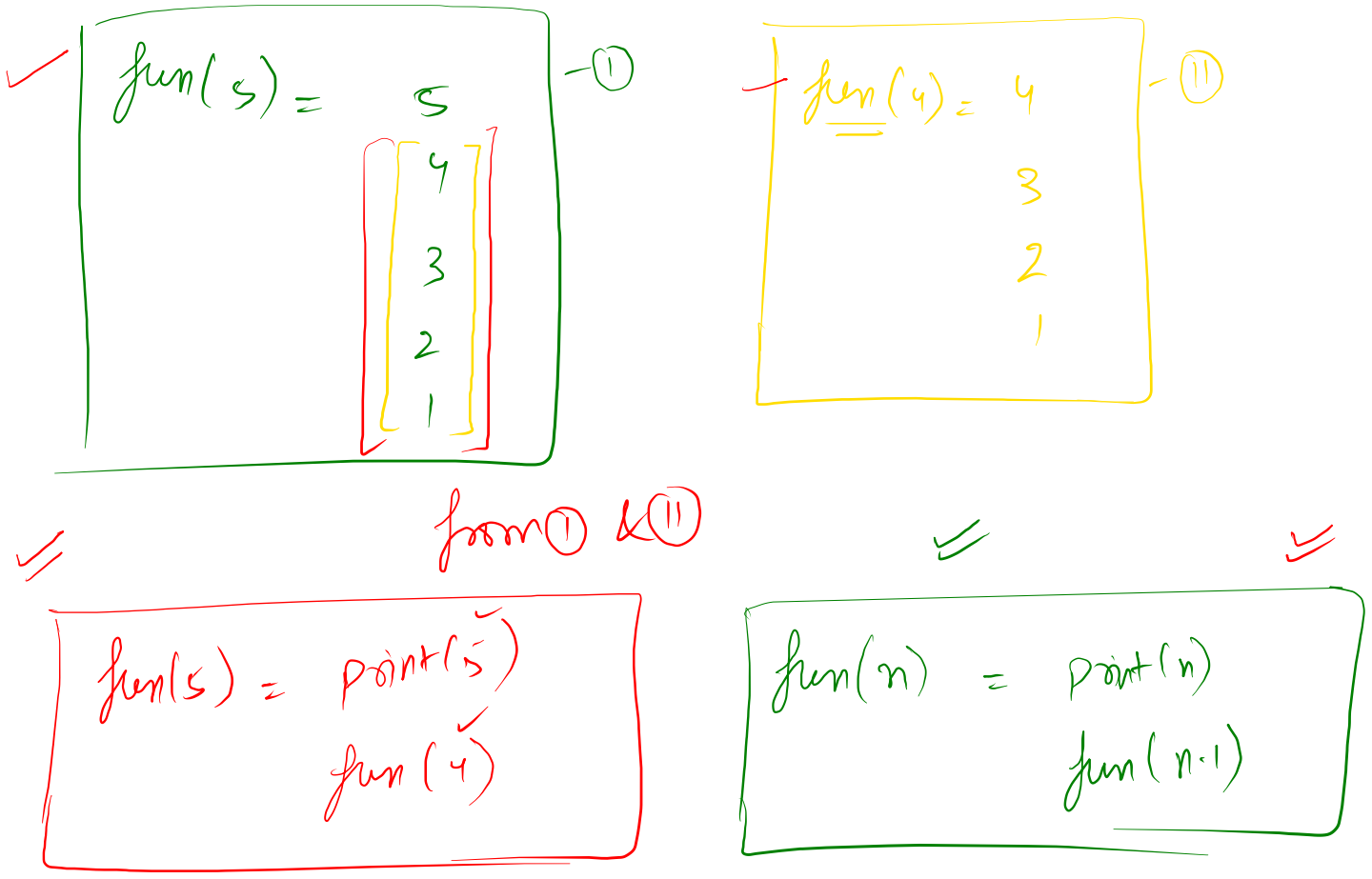
Recursion

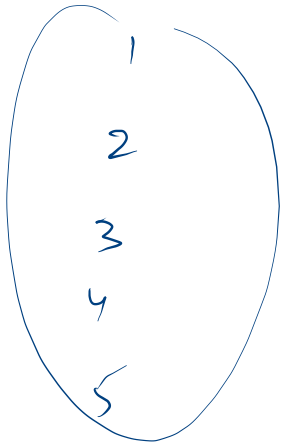
```
p s void fun() {

    ______
    ______
    ______
    ______

    fun();

}
```

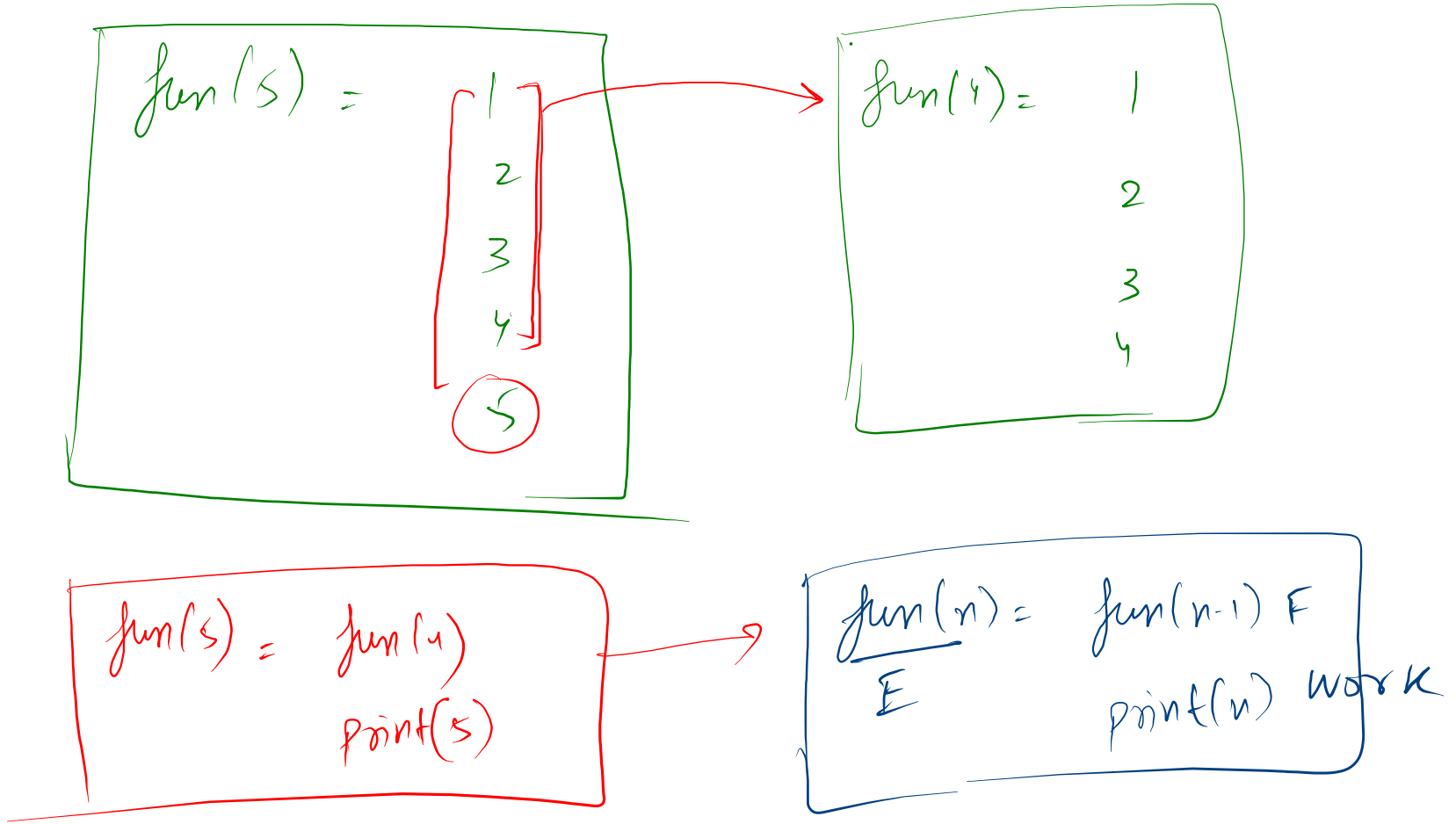



Input → 5
Output →
5
4
3
2
1


fun(5) = 5
4
3
2
1
-①
fun(4) = 4
3
2
1
-②
from ① & ②
fun(5) = print(5)
fun(4)
fun(n) = print(n)
fun(n-1)

input = 5 →

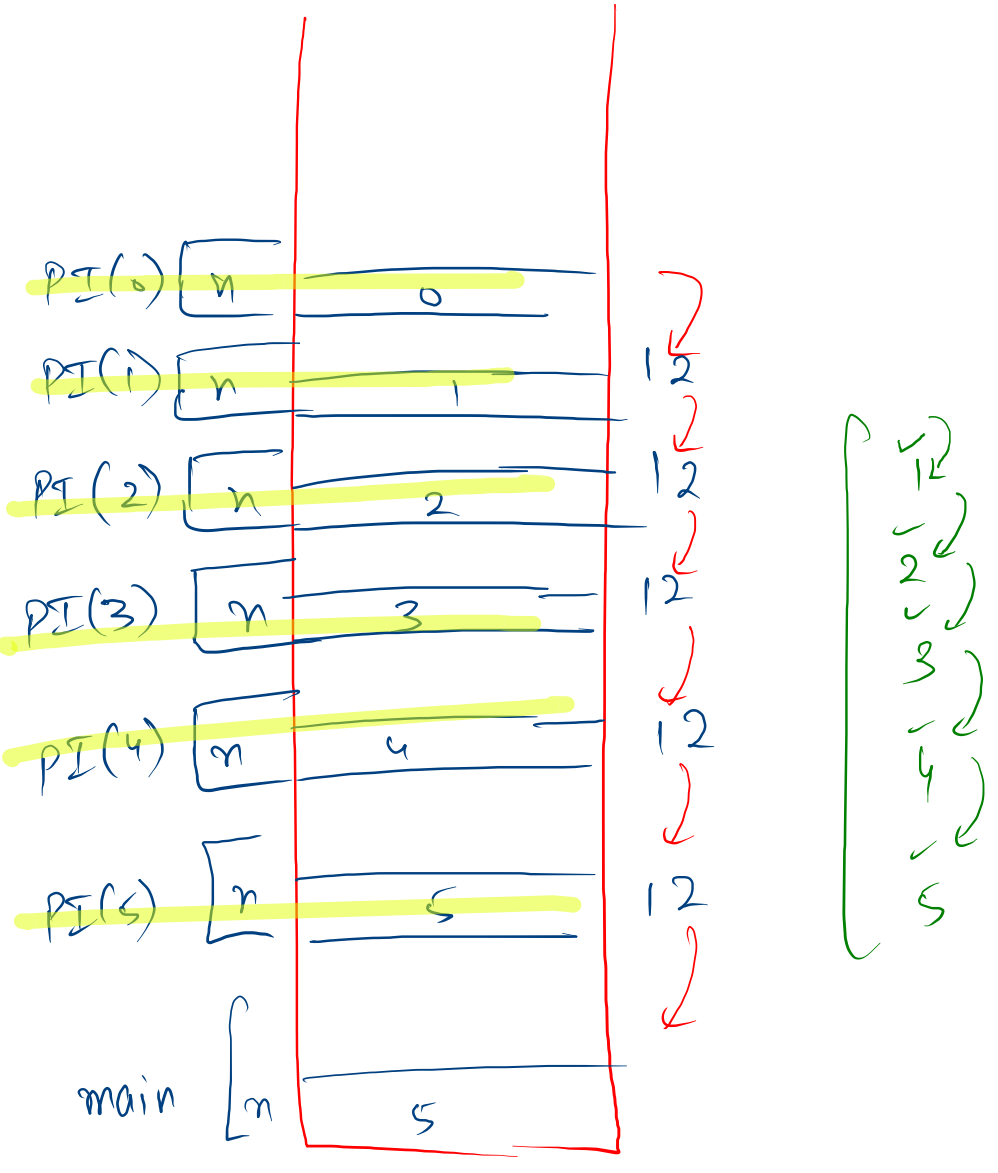



fun(5) =
1
2
3
4
5
fun(4) =
1
2
3
4
fun(5) = fun(4)
print(5)
fun(n) = fun(n-1) F
E
print(n) work

```java
public static void printIncreasing(int n){
    printIncreasing(n-1); -①
    System.out.println(n); -②
}
```

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

1
2
3
4
5

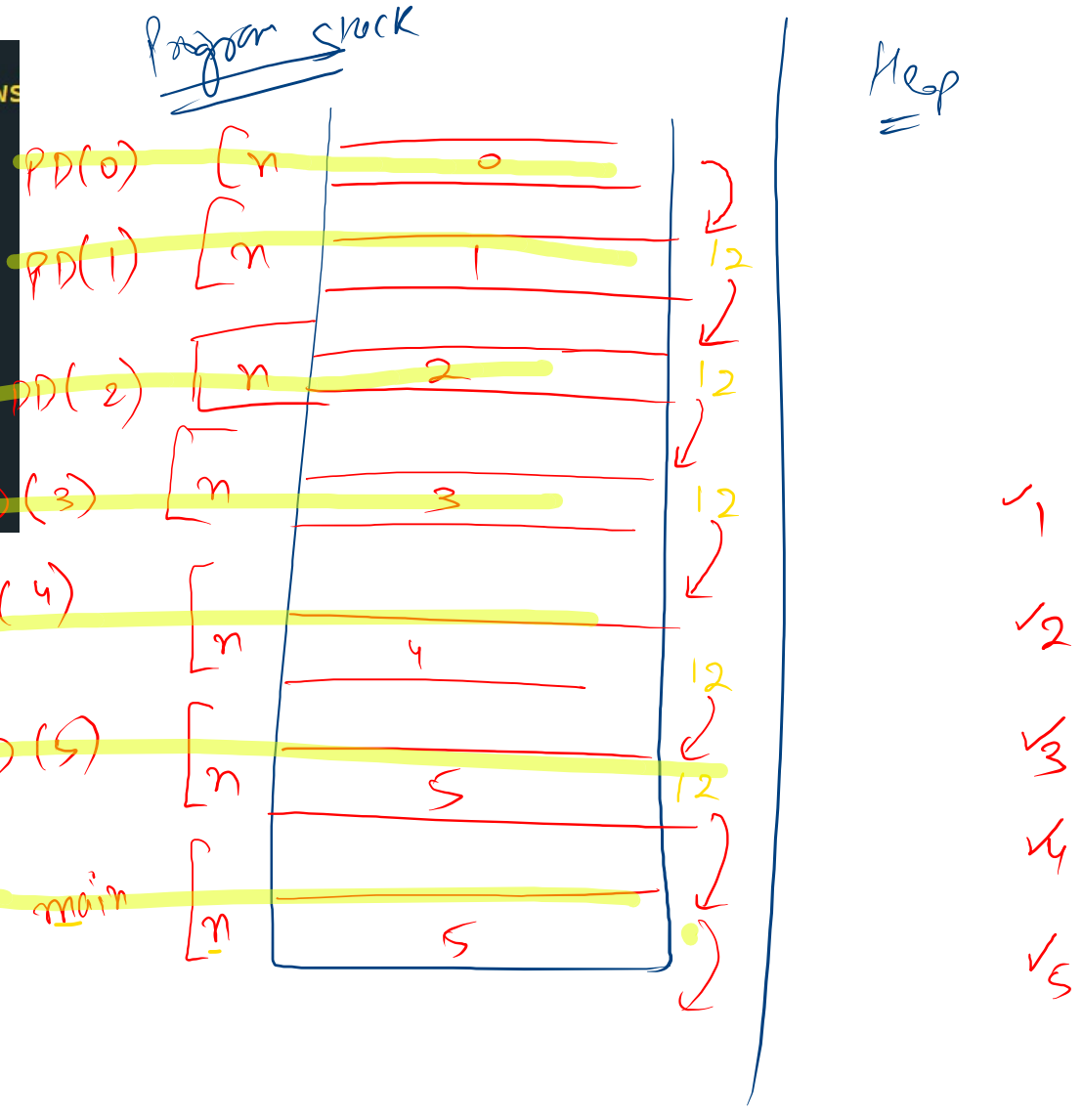


1
2
3
4
5

```java
public static void main(String[] args) throws
    Scanner scn = new Scanner(System.in);

    int n = scn.nextInt(); // 5

    printDecreasing(n);
}

public static void printDecreasing(int n){
    System.out.println(n); -①
    printDecreasing(n-1); -②
}
```

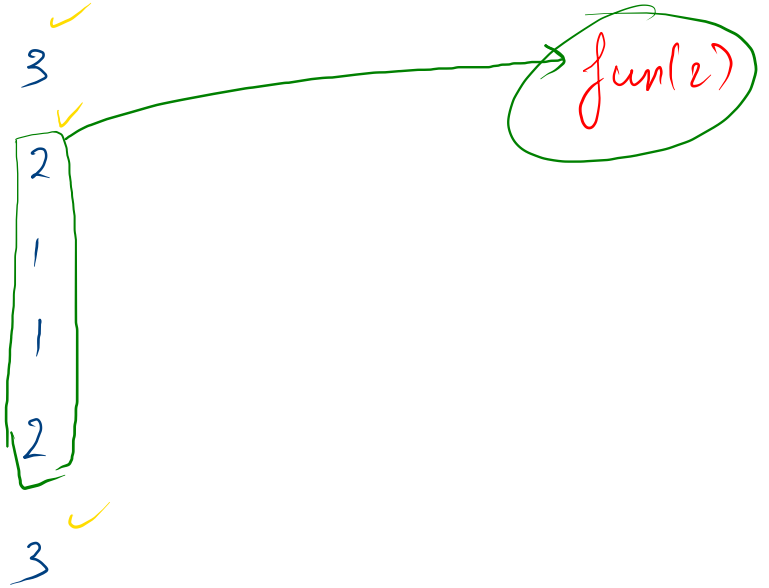
```java
if(n == 0){
    return;
}
```

5
4
3
2
1



✓1
✓2
✓3
✓4
✓5

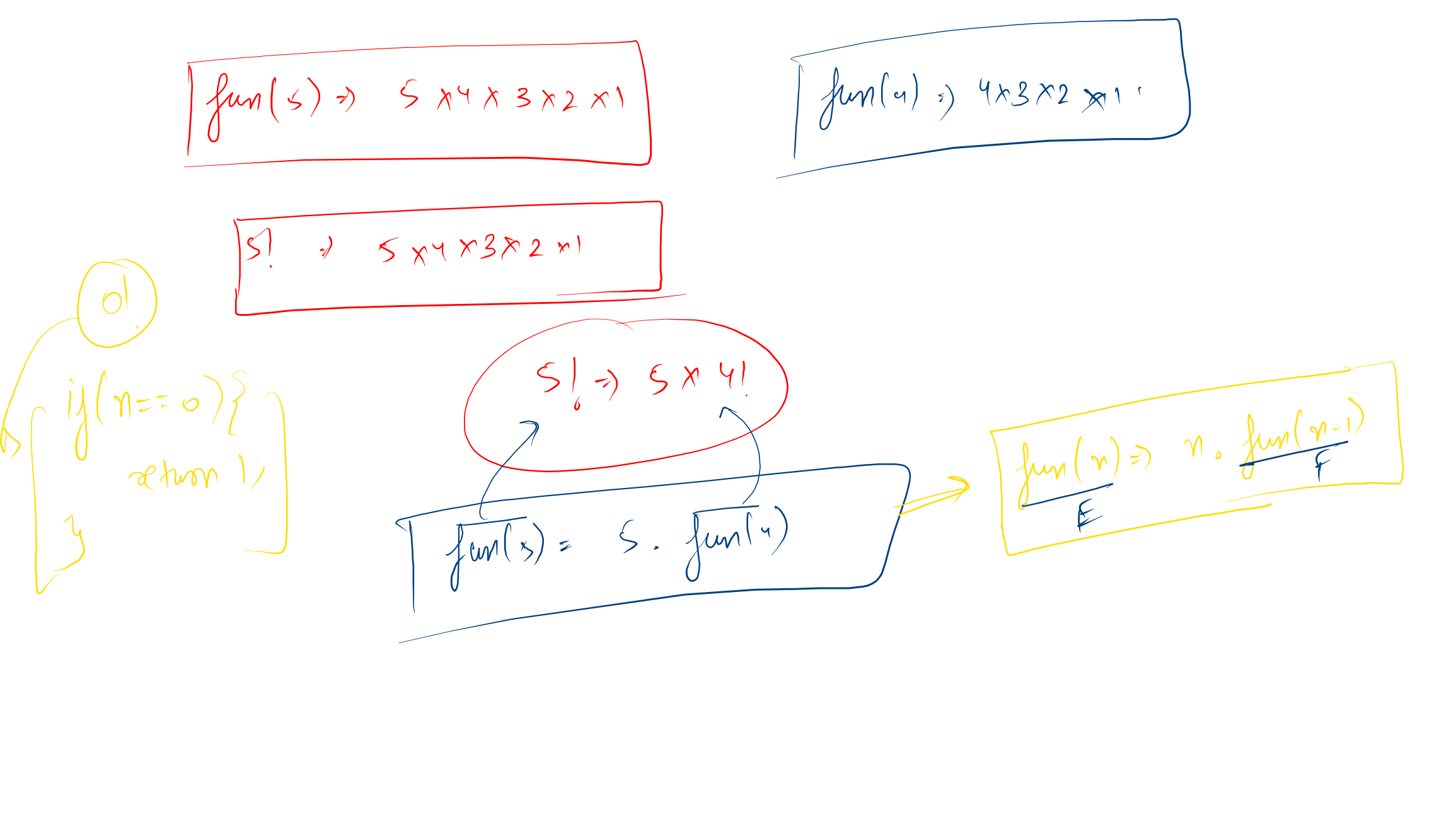
fun(3) :



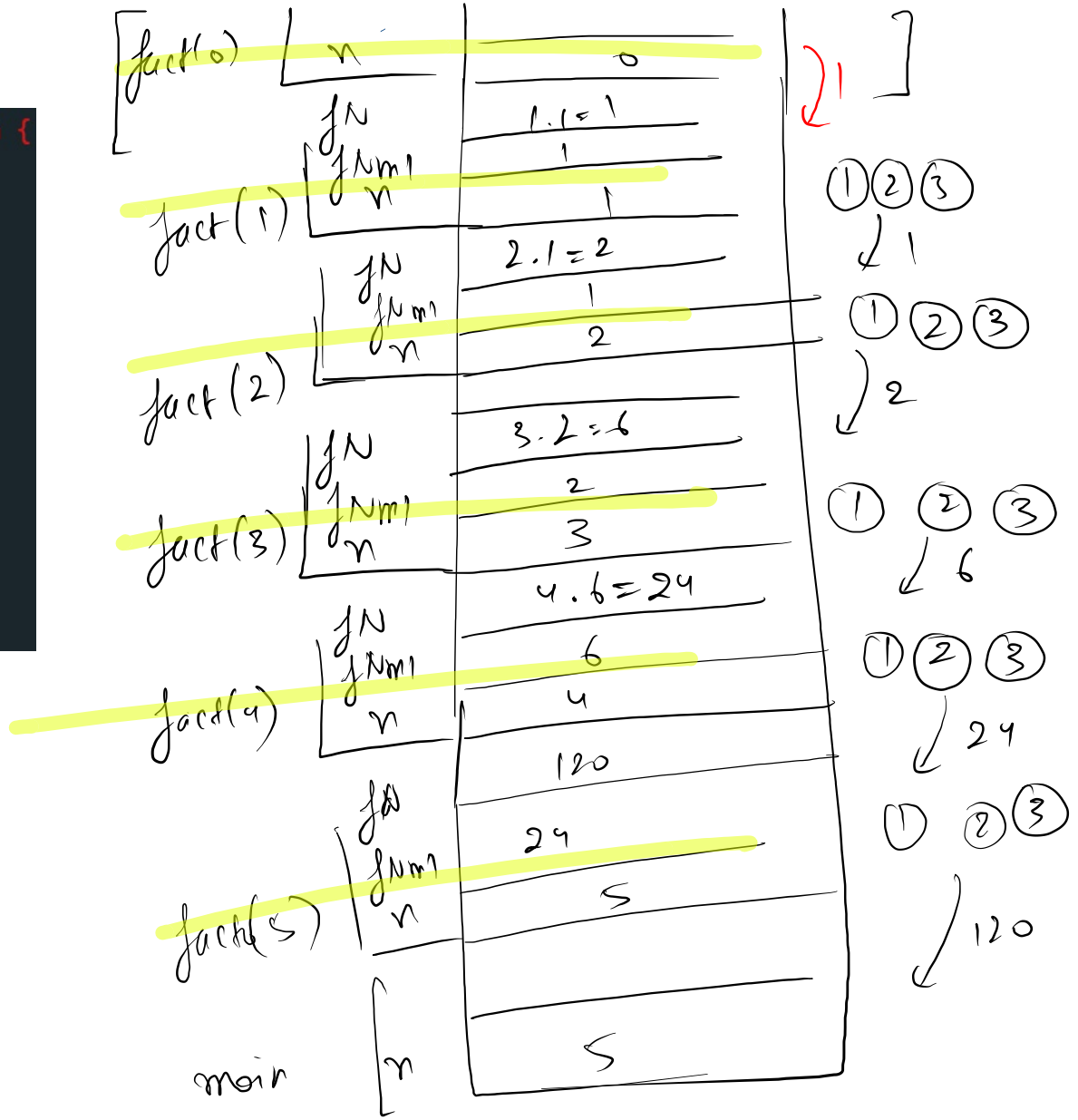
⇒ 2 1 1 2

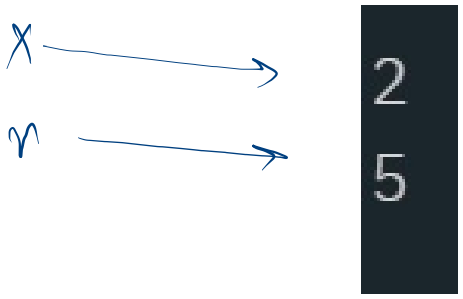
fun(3) = printf(3)
fun(2)
printf(3)

fun(n) ⇒ printf(n)
fun(n-1)
printf(n)


fun(5) => 5 x 4 x 3 x 2 x 1
fun(4) => 4 x 3 x 2 x 1
5! => 5 x 4 x 3 x 2 x 1
0!
if(n==0){
return 1;
}
5! => 5 x 4!
fun(5) = 5 . fun(4)
fun(n) => n . fun(n-1)
F
F

```java
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    int fact = factorial(n);
    System.out.println(fact);
}

public static int factorial(int n){
    if(n == 0){
        return 1;
    }
    int factNm1 = factorial(n-1);   -①
    int factN = n * factNm1;        -②
    return factN;                   -③
}
```





x^n

pow(2, 5) ⇒ 2^5

pow(2, 4) ⇒ 2^4

pow(2, 5) = 2 · pow(2, 4)

2^5 = 2 · 2^4

pow(x, n) = pow(x, n-1) · x

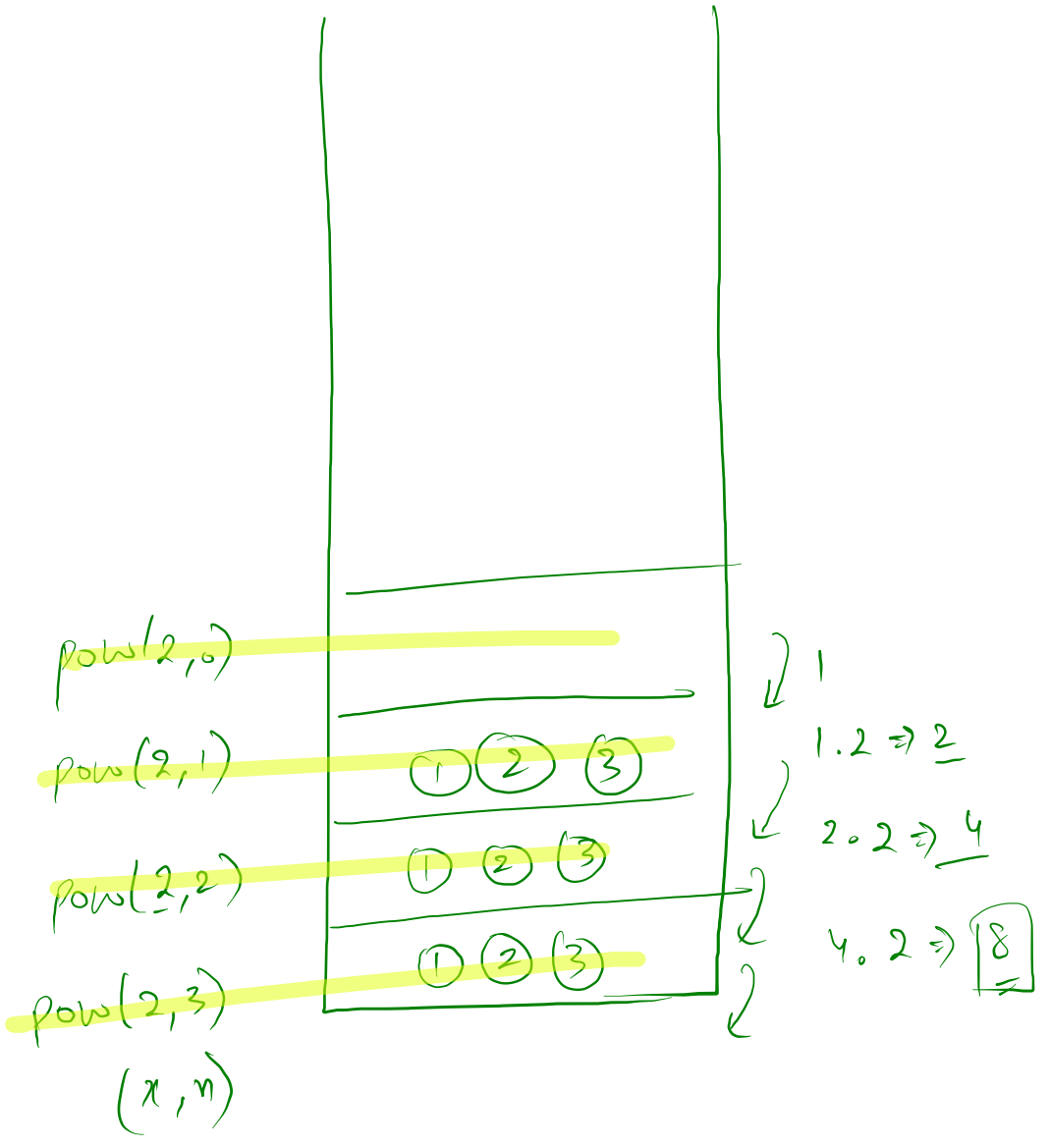
E F

```java
public static int power(int x, int n){
    int xPowNm1 = power(x,n-1);   - 1
    int xPowN = x * xPowNm1;      - 2

    return xPowN;                 - 3
}
```

2 3

```java
if(n == 0){
    return 1;
}
```

8



Mathematics

$2^{10} \Rightarrow 2^5 \cdot 2^5$

C.S. → maths

logic

$x^n = x \cdot x^{n-1}$

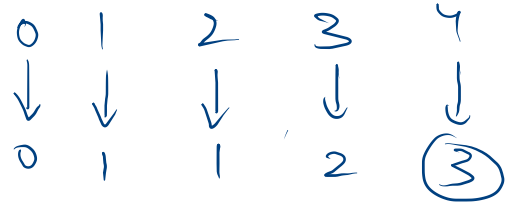
$x^n = x^{n/2} \cdot x^{n/2}$

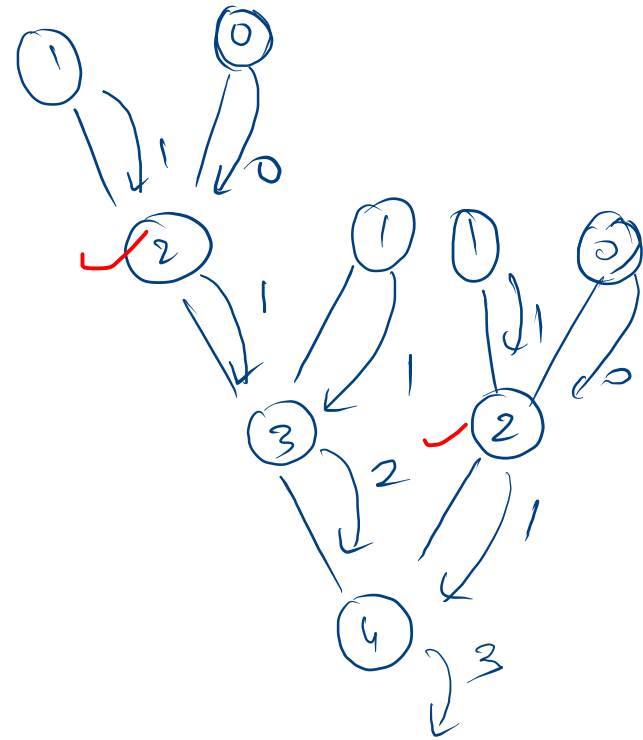
$2^9 = 2^4 \cdot 2^4 \cdot 2$

$$(x^n) = \begin{cases} (x^{n/2} \cdot x^{n/2}) & , \quad n \Rightarrow \text{even} \\ (x^{n/2} \cdot x^{n/2} \cdot x) & , \quad n \Rightarrow \text{odd} \end{cases}$$

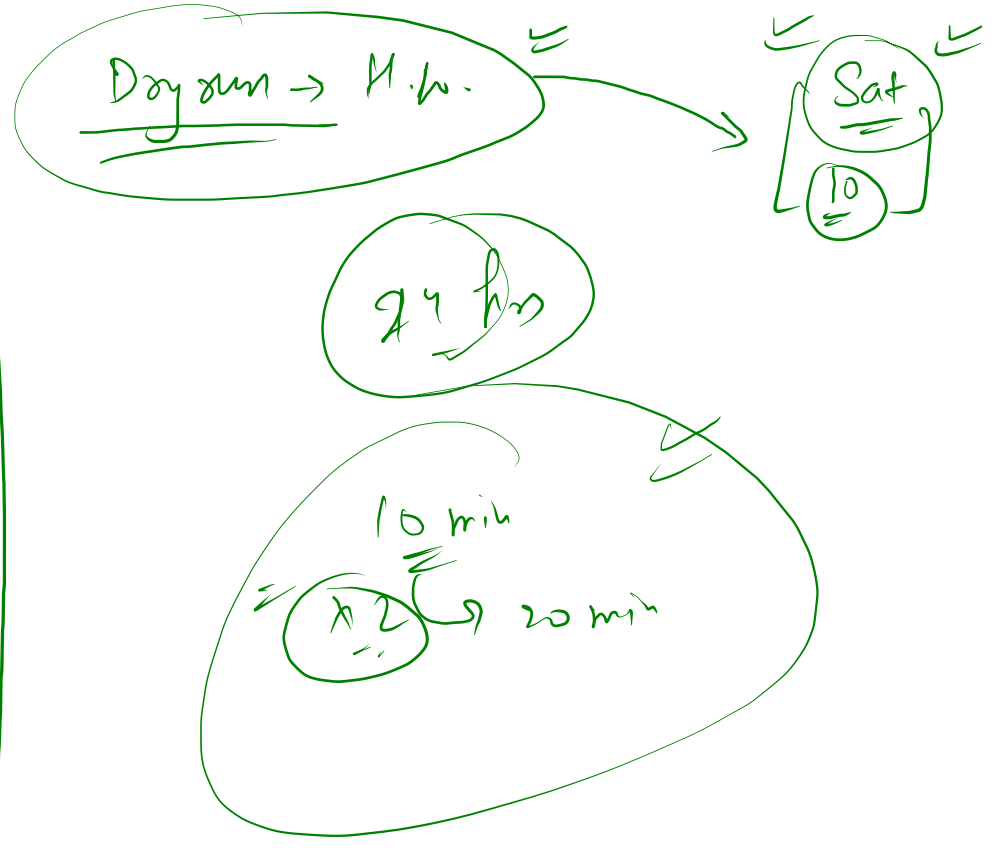
```java
public static int fib(int n){
    if(n == 0 || n == 1){
        return n;
    }
    int fibNm1 = fib(n-1);
    int fibNm2 = fib(n-2);

    int fibN = fibNm1 + fibNm2;

    return fibN;
}
```

n = 4





```java
public static int powerFake(int x, int n){
    if(n == 0){
        return 1;
    }
    if(n%2 == 0){
        return powerFake(x,n/2) * powerFake(x,n/2);
    }else{
        return powerFake(x,n/2) * powerFake(x,n/2) * x;
    }
}

public static int powerSmart(int x, int n){
    if(n == 0){
        return 1;
    }
    int xPowNb2 = powerSmart(x,n/2);
    if(n%2 == 0){
        return  xPowNb2 * xPowNb2;
    }else{
        return xPowNb2 * xPowNb2 * x;
    }
}
```



x n
(2, 10)

Input1 -> 1
Output1 -> 1 1 1

Input2 -> 2
Output2 -> 2 1 1 1 2 1 1 1 2

Input2 -> 3
Output3 -> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

$1 <= n <= 10$

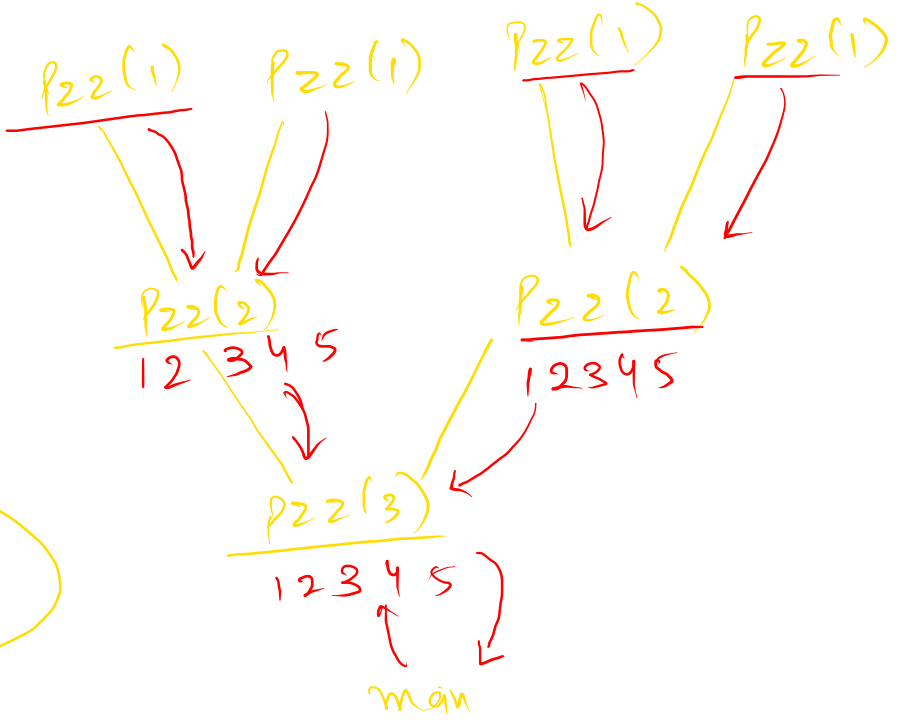
fun(2) ⇒
print(2)
fun(1)
print(2)
fun(1)
print(2)

fun(n) →
print(n)
fun(n-1) - F
print(n)
fun(n-1) - F
print(n)

```java
public static void pzz(int n){
    if(n == 1){
        System.out.print("1 1 1 ");
        return;
    }

    System.out.print(n+" ");   1
    pzz(n-1);                  2
    System.out.print(n+" ");   3
    pzz(n-1);                  4
    System.out.print(n+" ");   5
}
```

n=3

3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3



Tower of Hanoi

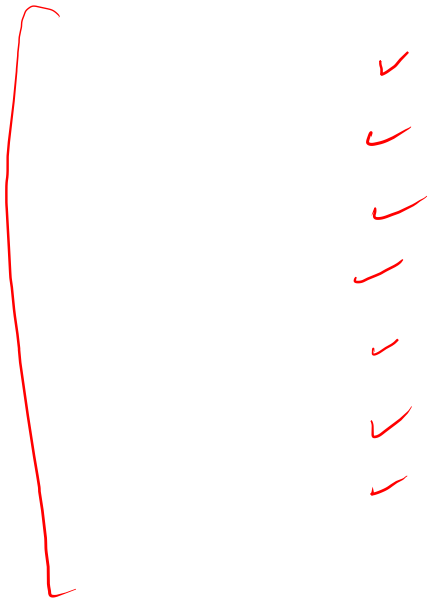
Task → (move all discs from src → dest)

Rule
- only disc on be moved at a time
- order must never be violated.

```
3
10
11
12
```

no. of disc

src 10 | dest 11 | Helper 12



src

dest

Helper

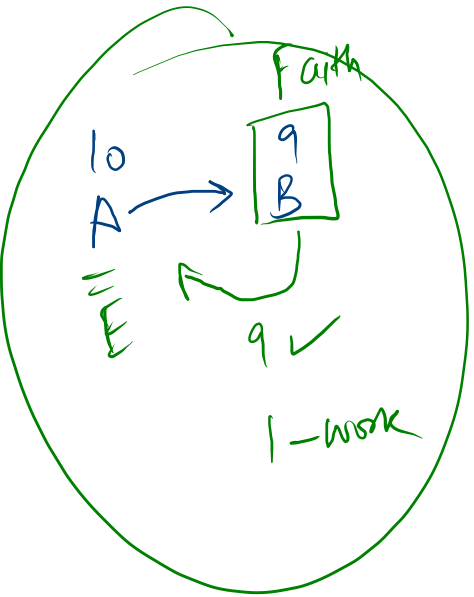
10

11

12

work + faith → Expectation

Expectation / faith model



n S D H

ToH(3, 10, 11, 12) →

Expectation

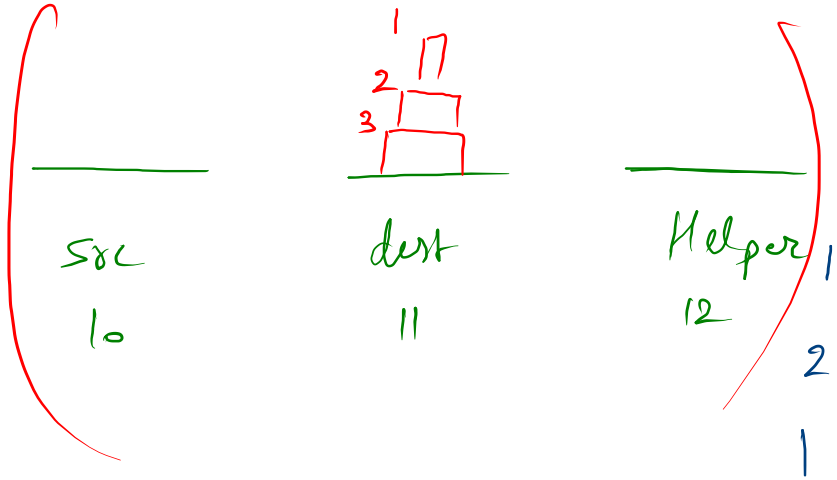
n-1 S H D

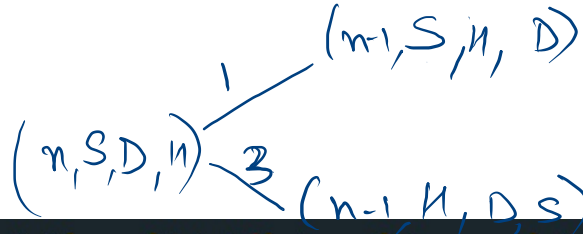
ToH(2, 10, 12, 11) faith

print(3 [10 → 11]) → work

ToH(2, 12, 11, 10) faith

n-1 H D S






```
public static void toh(int n, int src, int dest, int helper){
    if(n == 0){
        return;
    }
    toh(n-1,src,helper,dest); - 1
    System.out.println(n+"["+src+" -> "+dest+"]"); - 2
    toh(n-1,helper,dest,src); - 3
}
```