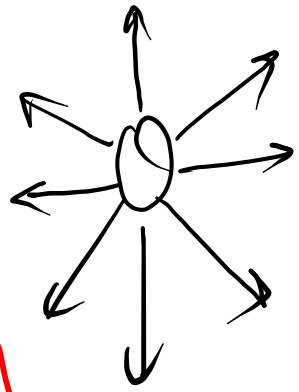
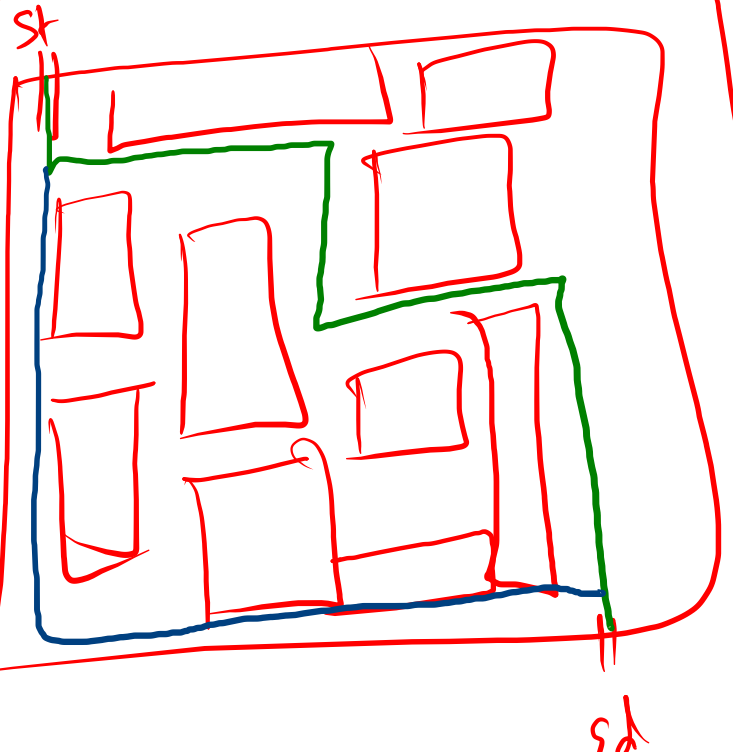
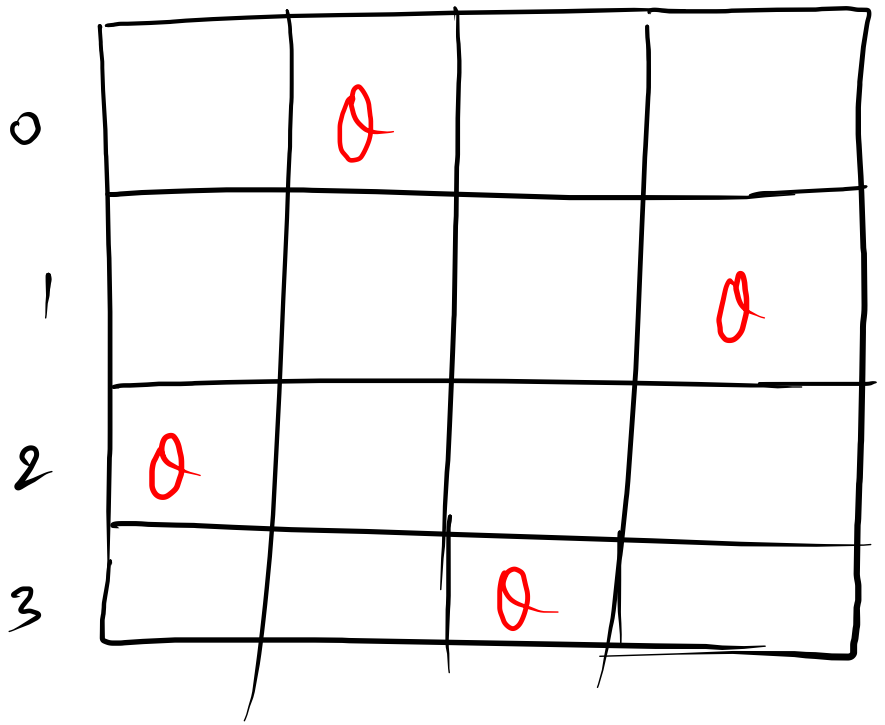
Recursion / backtracking

- Introduction (movement) / direction
- corners
- array list





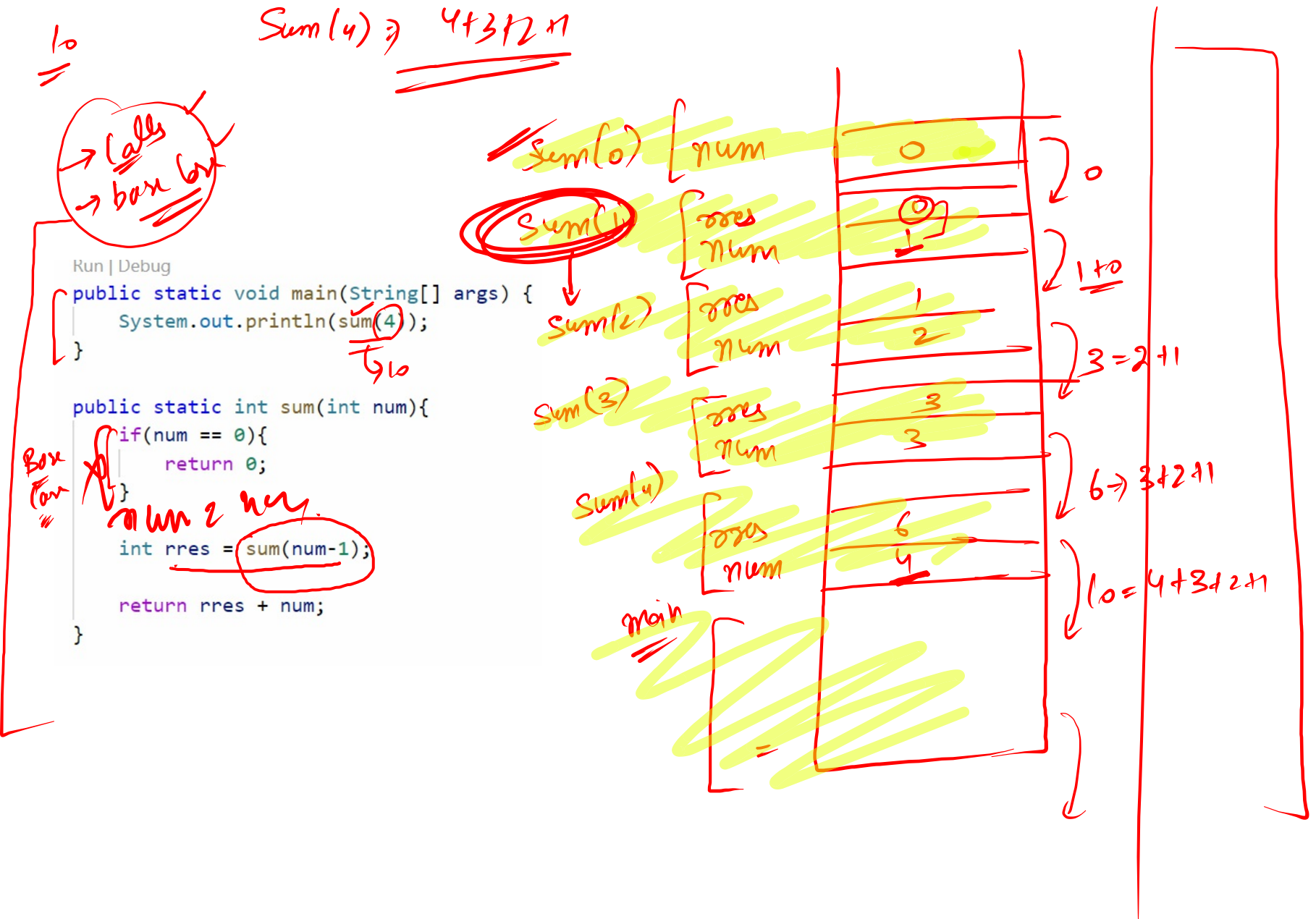


Recursion

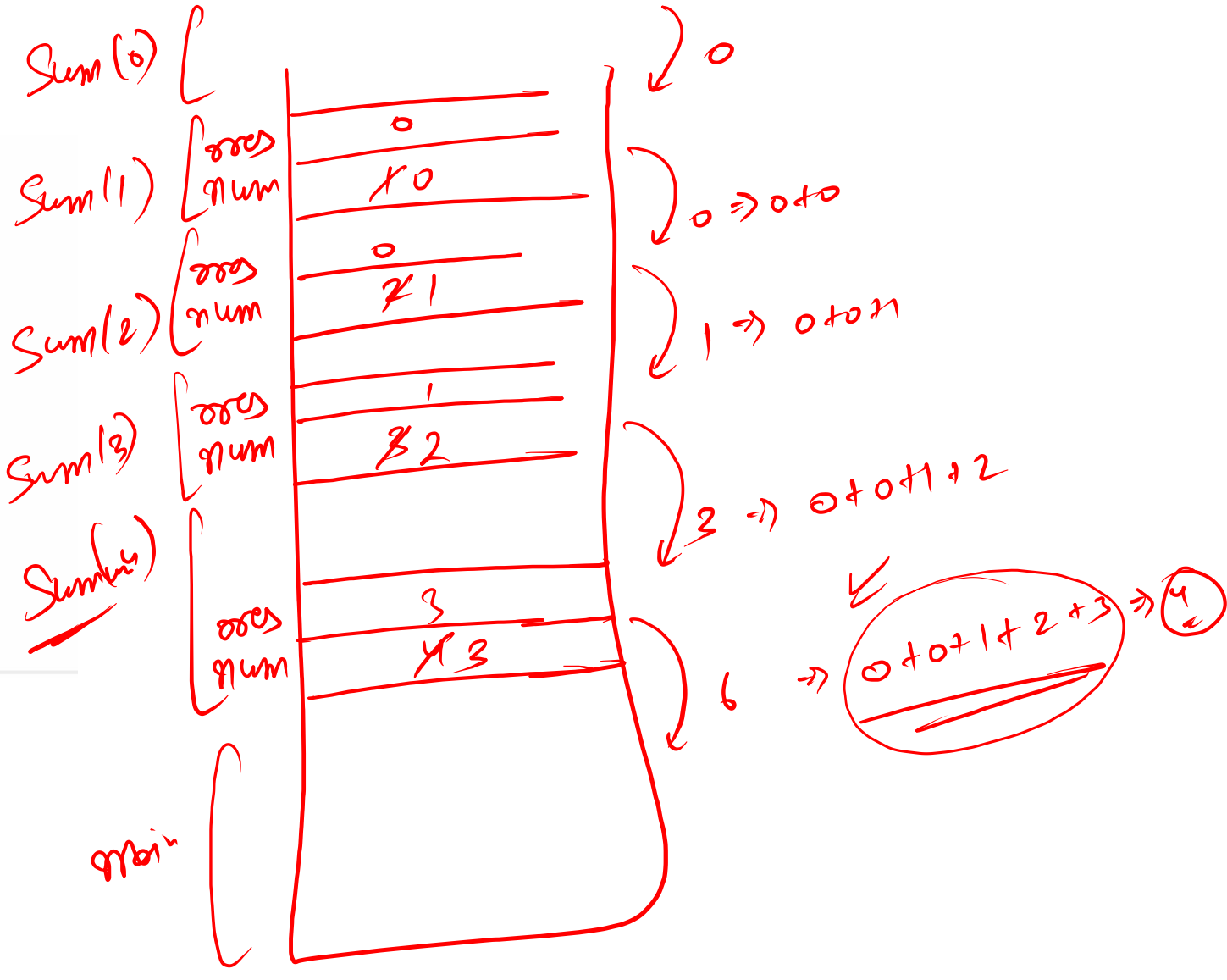
```
public static void func(...) {
    ...
    ...
    ...
    fun();
    ...
    ...
}
```

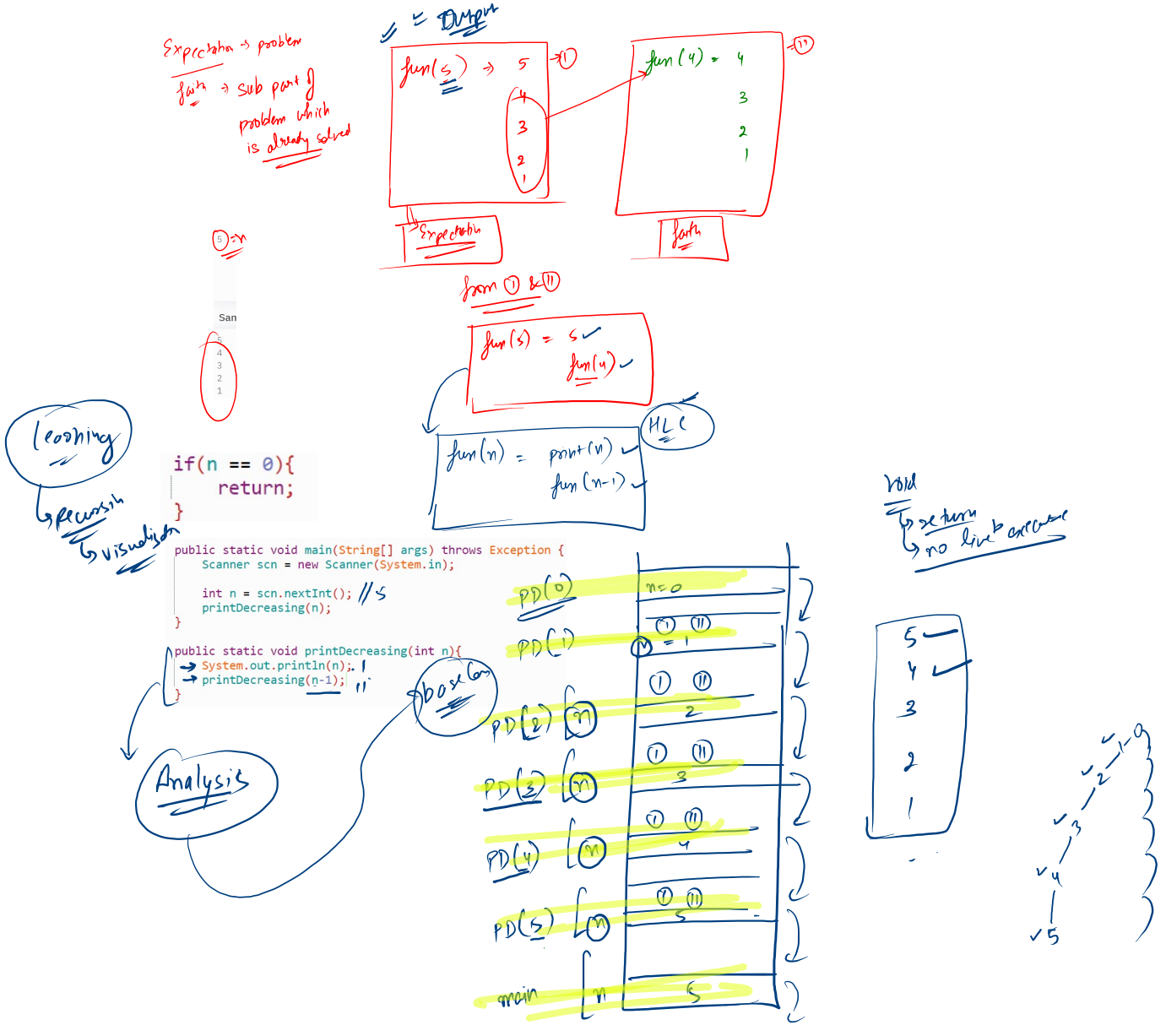
Run | Debug

```java
public static void main(String[] args) {
    System.out.println(sum(4));
}

public static int sum(int num){
    if(num == 0){
        return 0;
    }

    int rres = sum(num-1);

    return rres + num;
}
```



```java
public class L001 {
    Run | Debug
    public static void main(String[] args) {
        System.out.println(sum(4));
    }

    public static int sum(int num){
        if(num == 0){
            return 0;
        }
        num =num -1;
        int rres = sum(num);

        return rres + num;
    }
}
```

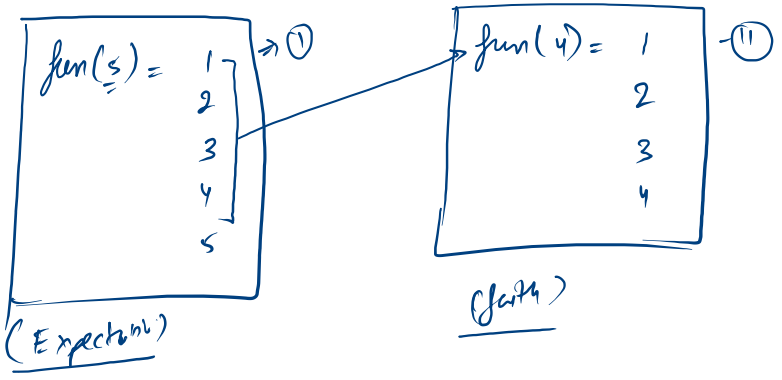


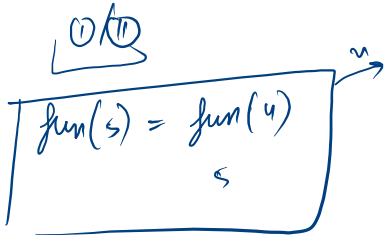

Expectation → problem
faith → sub part of problem which is already solved
fun(5) → 5
fun(4) = 4 3 2 1
Expectation
faith
from 1 & 11
fun(5) = 5
fun(4)
fun(n) = print(n)
fun(n-1)
HLC
learning
recursion
visualize
if(n == 0){
return;
}
public static void main(String[] args) throws Exception {
Scanner scn = new Scanner(System.in);
int n = scn.nextInt(); // 5
printDecreasing(n);
}
public static void printDecreasing(int n){
System.out.println(n);
printDecreasing(n-1);
}
box
Analysis
PD(0)
PD(1)
PD(2)
PD(3)
PD(4)
PD(5)
main
n=0
m = 1
2
3
4
5
5
4
3
2
1
void
return
no live execution

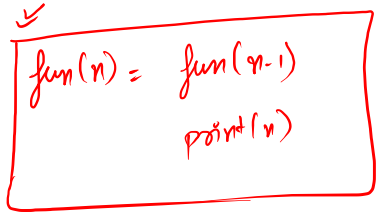
5

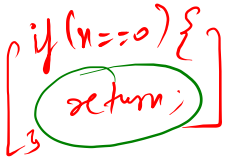
Samp

1
2
3
4
5

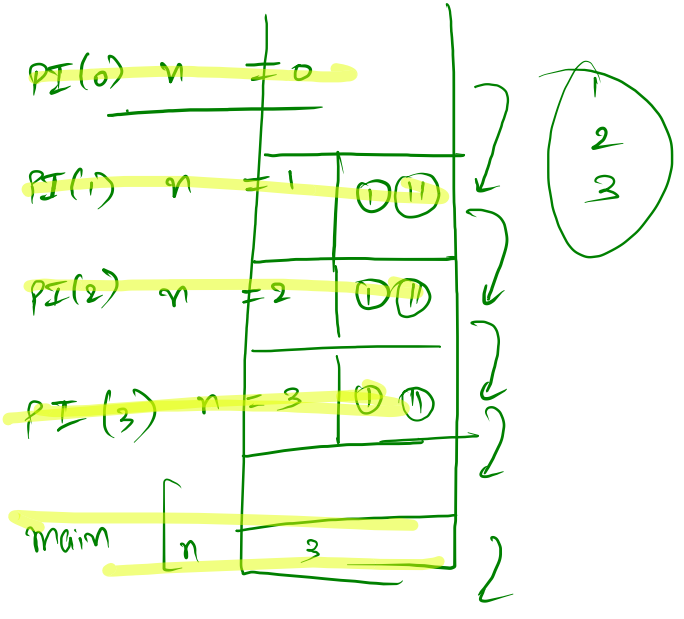


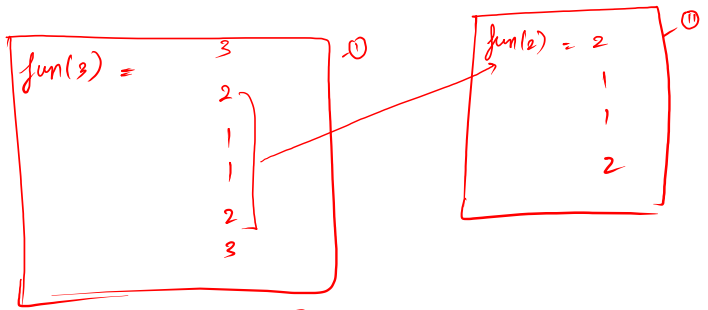




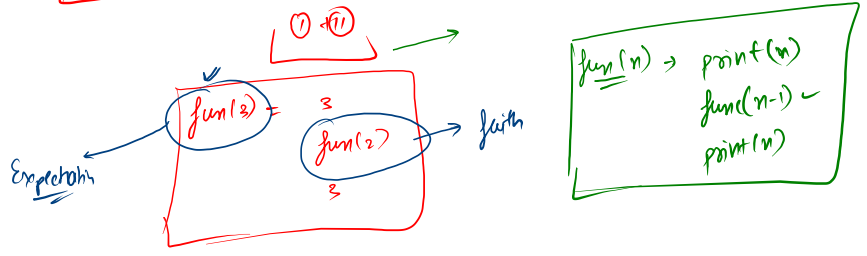



```java
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt(); //3
    printIncreasing(n);
}

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





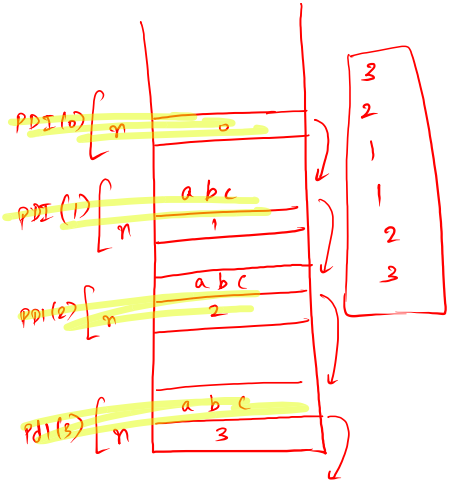

```
3
Sampl
3
2
1
1
2
```

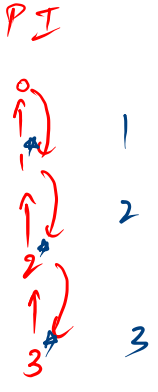


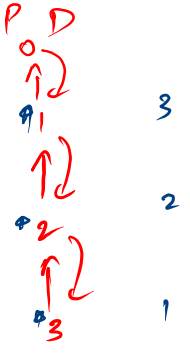
n=3

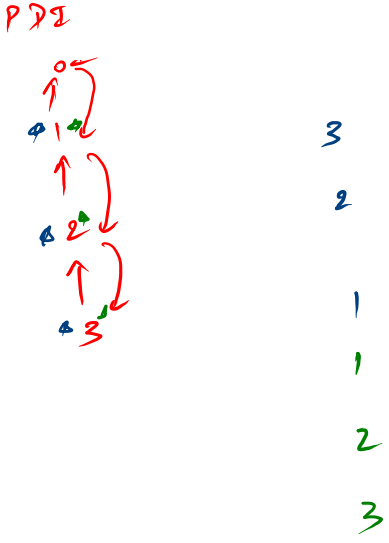
```
if(n==0){
    return;
}
```

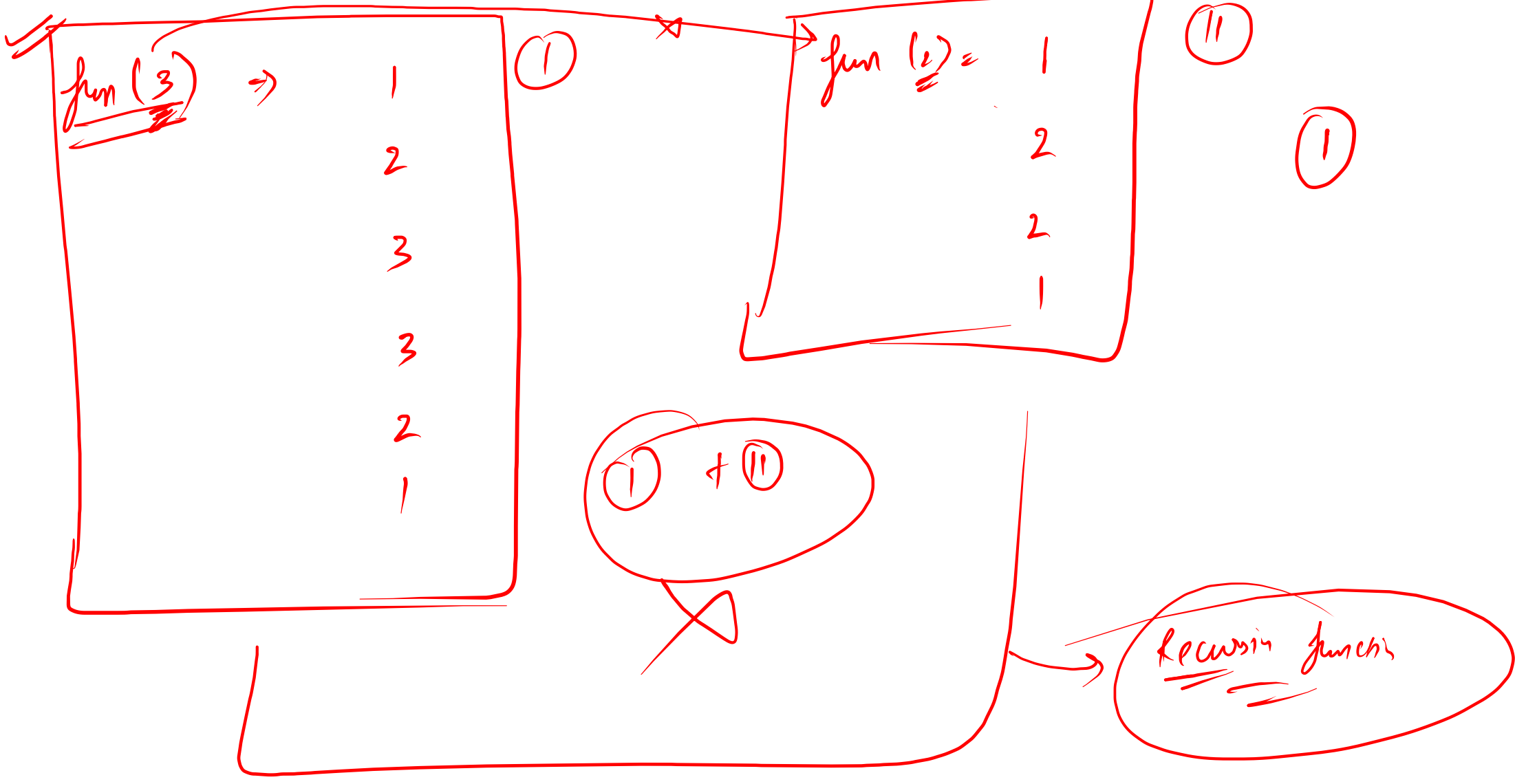
```
public static void pdi(int n){
    System.out.println(n); a
    pdi(n-1); b
    System.out.println(n); c
}
```

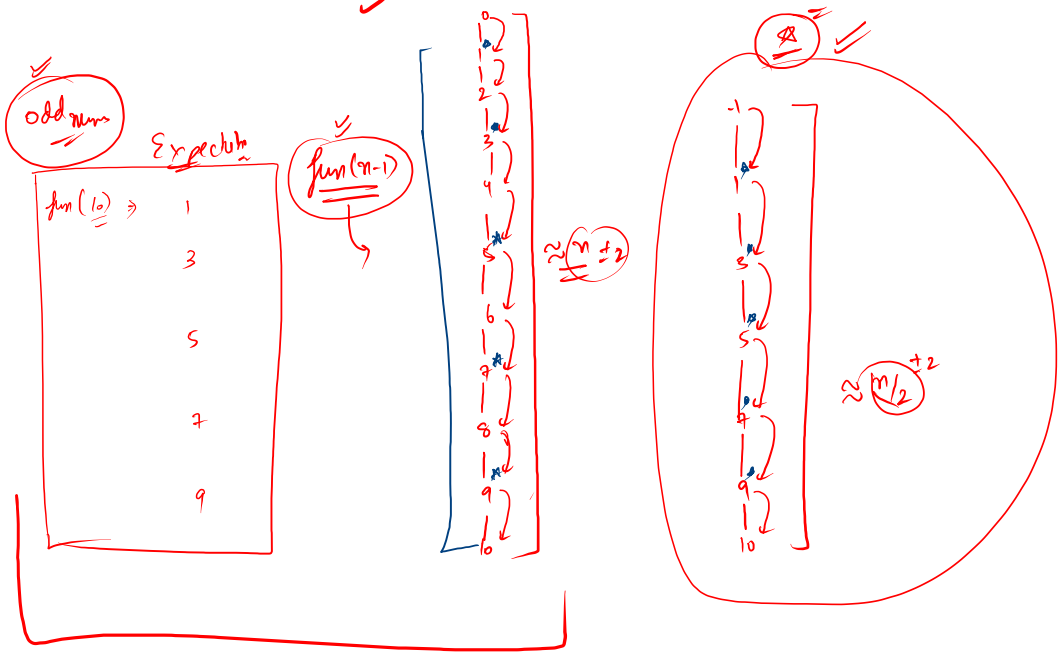








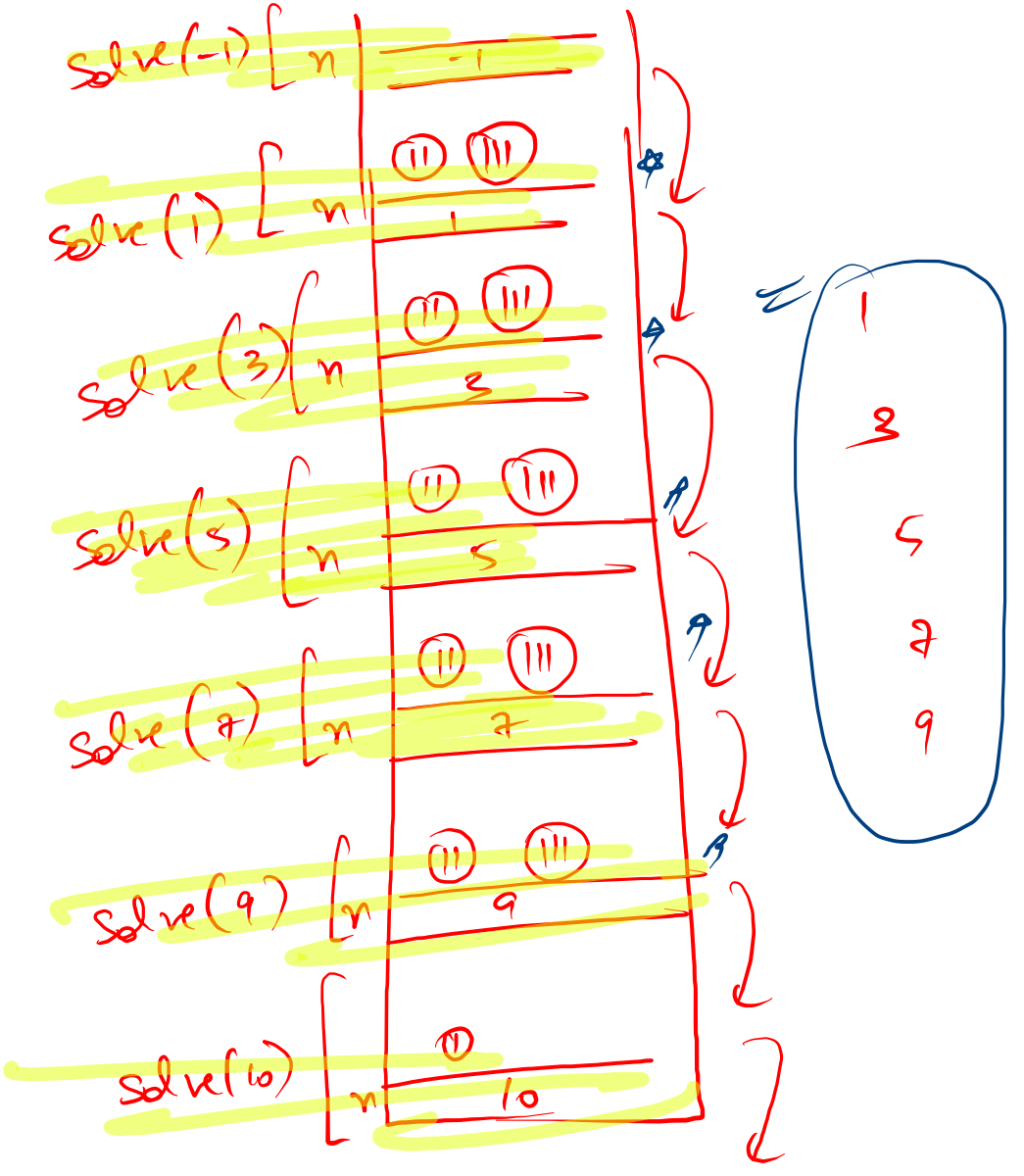

fun(3) ⇒
1
2
3
3
2
1
①
fun(2) =
1
2
2
1
②
①
①+②
Recursive function

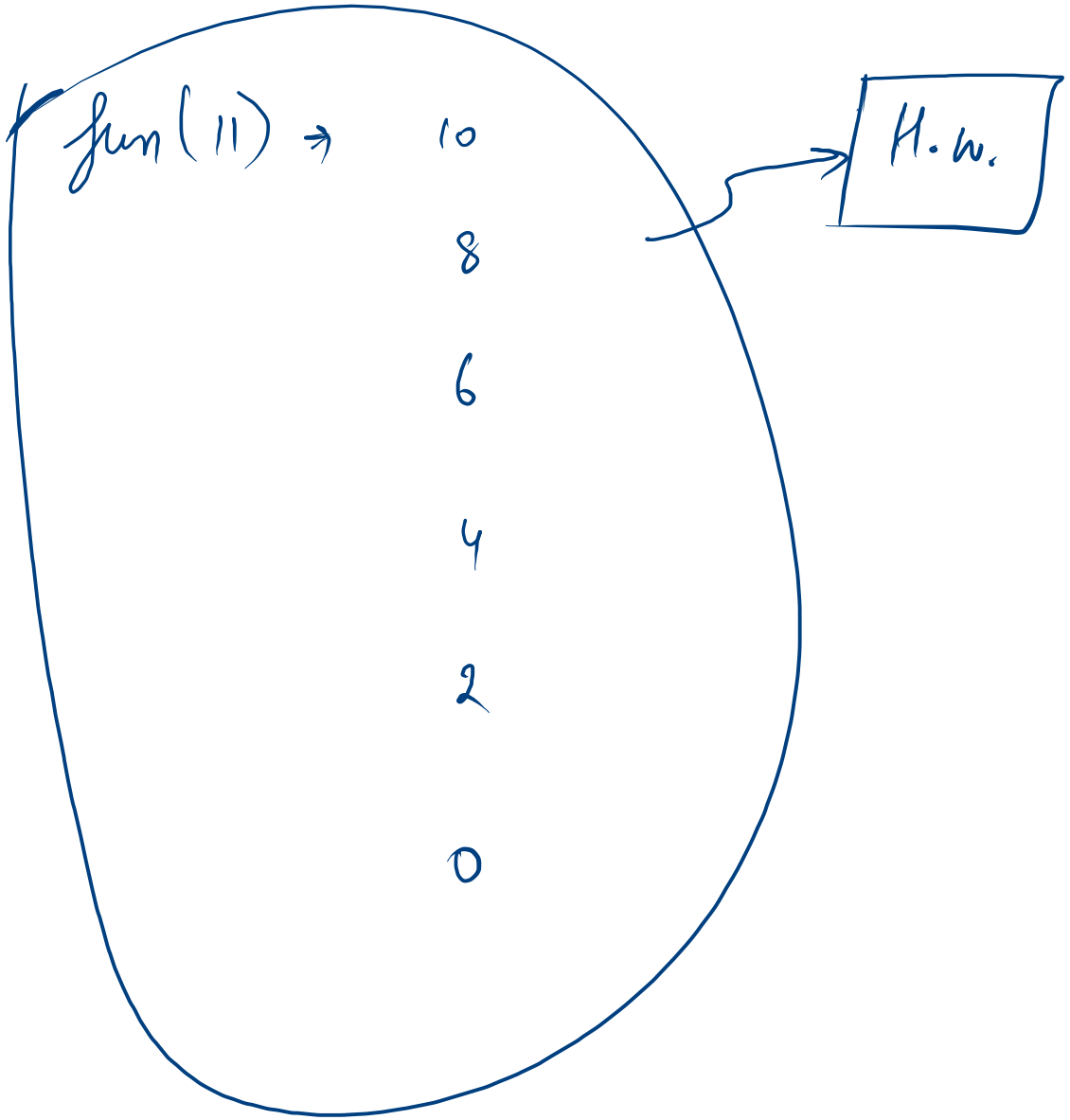


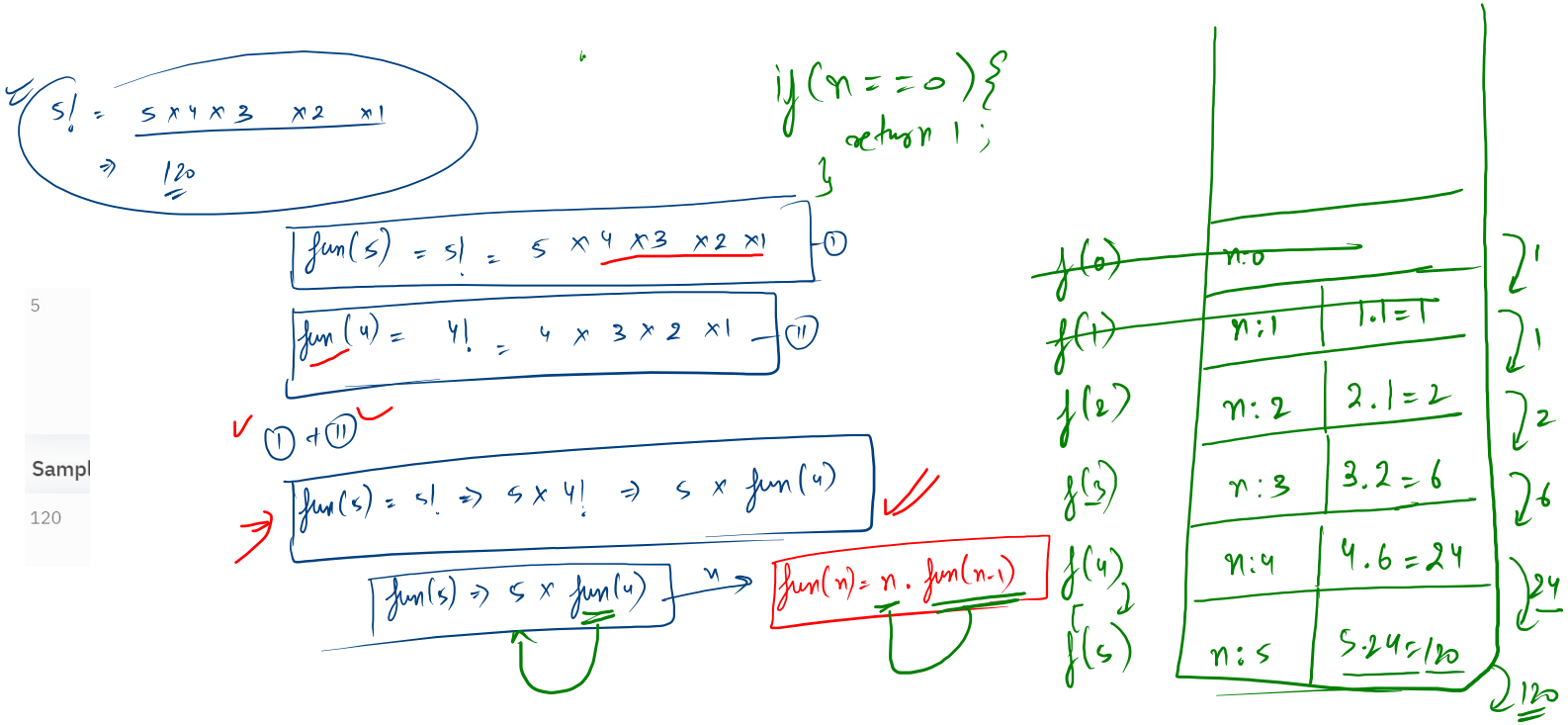

```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    solve(n);
}

public static void solve(int n){

}
```

```
psvm fun(int n) {
    if(n==0){
        return;
    }
    fun(n-1);
    if(n%2==1){
        System.out.println(n);
    }
}
```

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

public static void solve(int n){
    if(n <= 0){
        return;
    }
    if(n % 2 == 0){
        solve(n-1);
    }else{
        solve(n-2);
        System.out.println(n);
    }
}
```




fun(11) →
10
8
6
4
2
0
H.W.




```java
public static int factorial(int n){
    if(n == 0){
        return 1;
    }
    return n * factorial(n-1);
}
```

5

Sampl

120

$x^n \Rightarrow 2^5 \Rightarrow 32$

2 ⇒ x

5 ⇒ n

$fun(x, n) \Rightarrow x^n$

$2^5 \Rightarrow 2 \cdot 2^4$

E F

$fun(2, 5) \Rightarrow 2^5 \Rightarrow 32$

$\Rightarrow 2 \cdot 2^4$

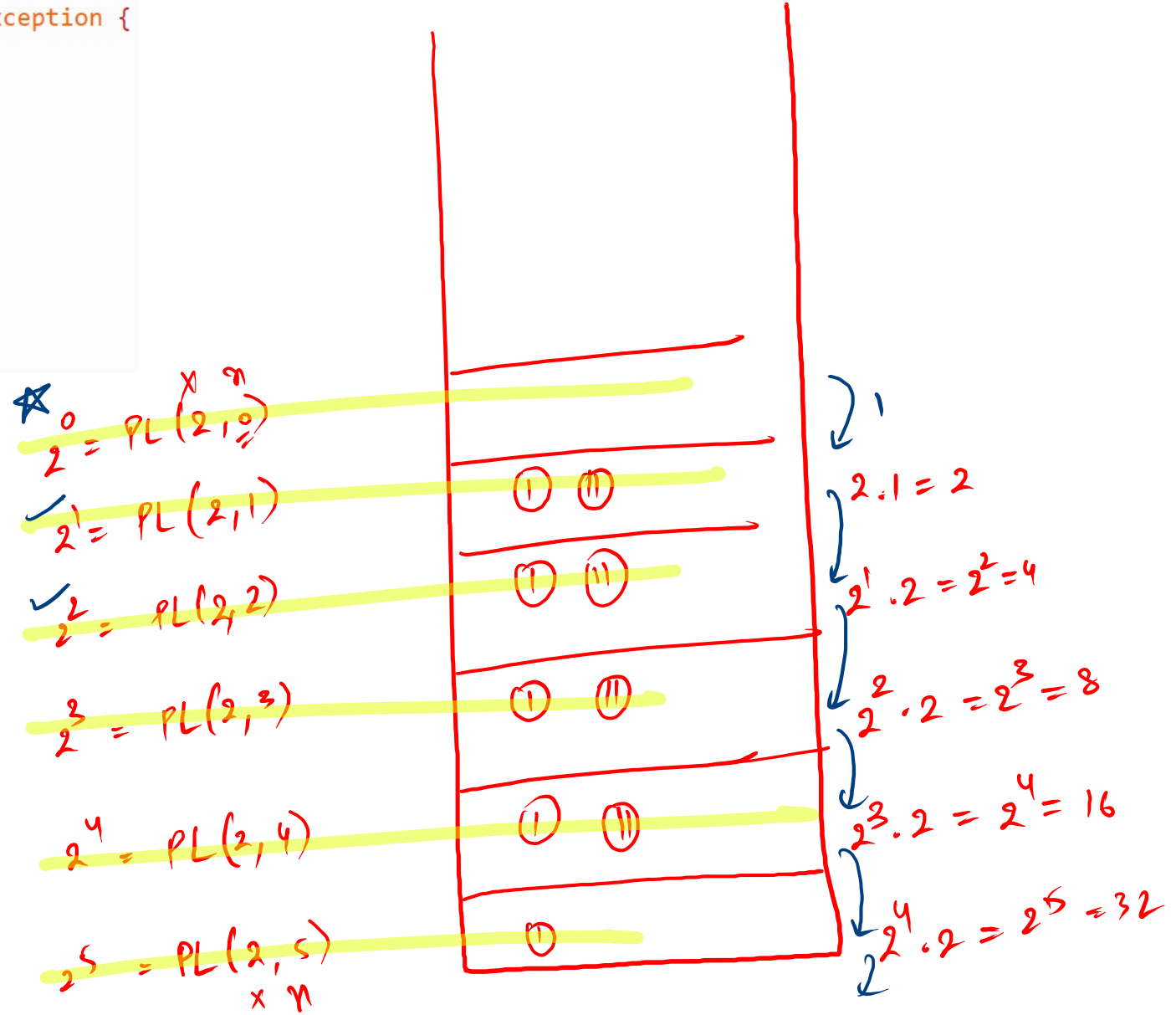
$fun(2,5) \Rightarrow 2 \cdot fun(2,4)$

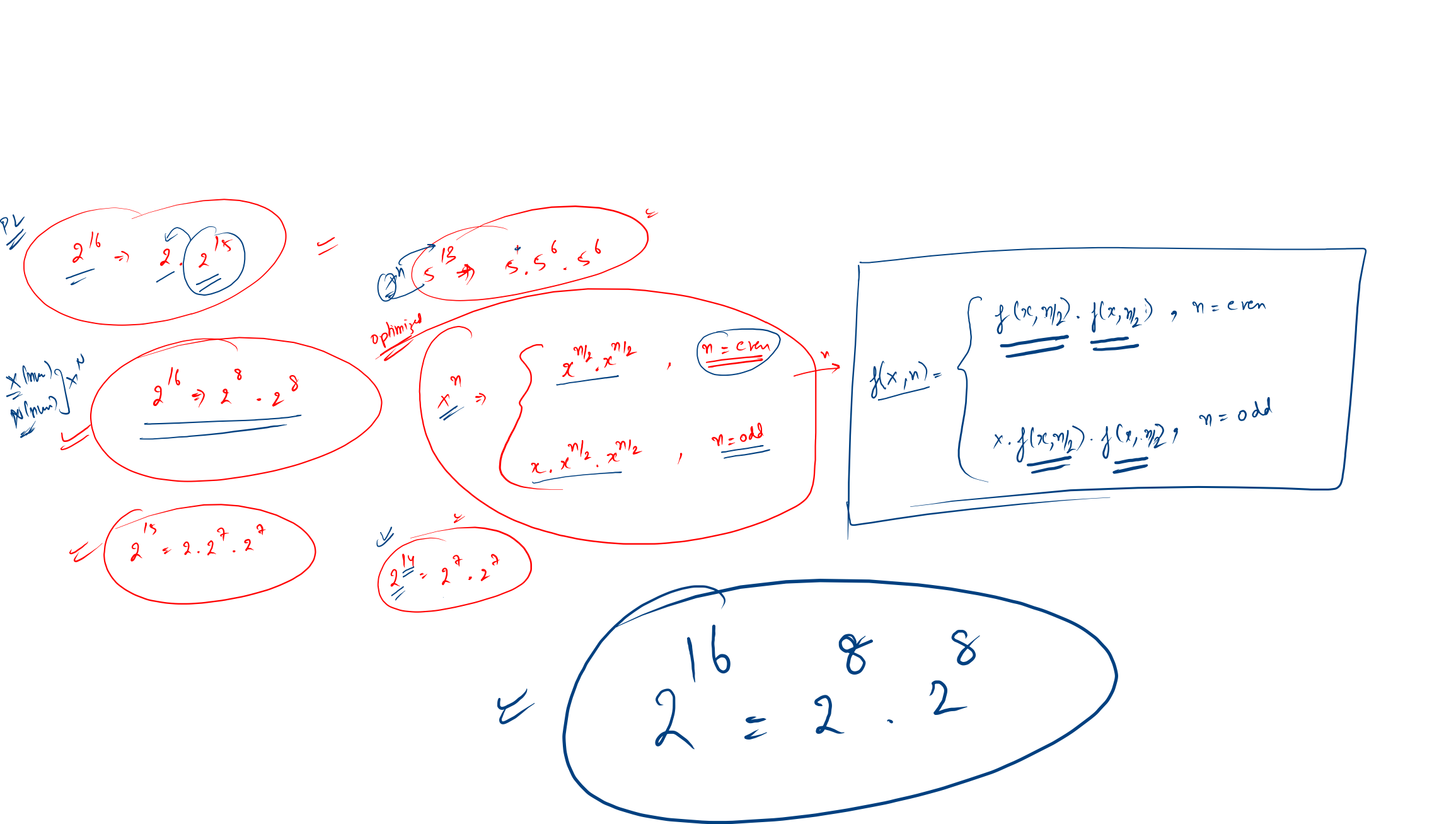
$fun(x,n) = x \cdot fun(x, n-1)$

```java
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int x = scn.nextInt(); // 2
    int n = scn.nextInt(); // 5
    System.out.println(powerLinear(x,n));
}

public static int powerLinear(int x, int n){
    int xPowNm1 = powerLinear(x,n-1); (I)
    int xPowN = x * xPowNm1; (II)
    return xPowN;
}
```

$num^0 = 1$

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



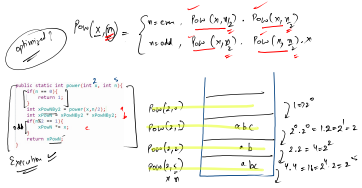

$2^{16} \Rightarrow 2 \cdot 2^{15}$
$5^{18} \Rightarrow 5 \cdot 5^{6} \cdot 5^{6}$
optimized
$x^n \Rightarrow$
$x^{n/2} \cdot x^{n/2}$, n = even
$x \cdot x^{n/2} \cdot x^{n/2}$, n = odd
$f(x,n) = \begin{cases} f(x,n/2) \cdot f(x,n/2), & n = \text{even} \\ x \cdot f(x,n/2) \cdot f(x,n/2), & n = \text{odd} \end{cases}$
$2^{16} \Rightarrow 2^{8} \cdot 2^{8}$
$2^{15} = 2 \cdot 2^{7} \cdot 2^{7}$
$2^{14} = 2^{7} \cdot 2^{7}$
$2^{16} = 2^{8} \cdot 2^{8}$

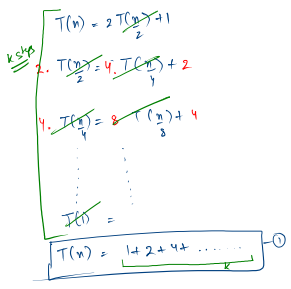
```java
public static int power(int x, int n){
    if(n == 0){
        return 1;
    }
    int xPownBy2 = power(x,n/2);
    int xPownN = xPownBy2 * xPownBy2;
    if(n%2 == 1){
        xPownN *= x;
    }
    return xPownN;
}
```

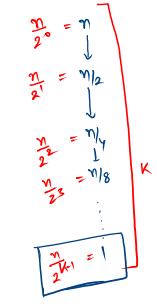
$$Pow(x,n) = \begin{cases} Pow\left(x,\frac{n}{2}\right) \cdot Pow\left(x,\frac{n}{2}\right), & n \Rightarrow even \\ x \cdot Pow\left(x,\frac{n}{2}\right) \cdot Pow\left(x,\frac{n}{2}\right), & n \Rightarrow odd \end{cases}$$

2 recursive calls

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 1$$







$$\frac{n}{2^{k-1}} = 1$$

$$n = 2^{k-1}$$

$$\log n = k-1$$

$$k = 1 + \log\frac{n}{2} \quad (11)$$

total terms $\left(1+\log\frac{n}{2}\right)$

(1) + (11)

$$T(n) = \underbrace{1 + 2 + 4 + \ldots}_{1+\log_2 n} \Rightarrow a\left[\frac{r^n - 1}{r-1}\right] \Rightarrow 2^{1+\log_2 n} - 1$$

$$T(1) \Rightarrow 2^1 \cdot 2^{\log_2 n} - 1$$

$$T(n) \Rightarrow 2 \cdot n^{\log_2 2} - 1$$

$$T(n) \Rightarrow 2n - 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 1$$

$$2.\ T\left(\frac{n}{2}\right) = 4.\ T\left(\frac{n}{4}\right) + 2$$

$$4.\ T\left(\frac{n}{4}\right) = 8\ T\left(\frac{n}{8}\right) + 4$$

$$T(1) =$$

$$pow(x,n) = \begin{cases} pow(x,\frac{n}{2}) \cdot pow(x,\frac{n}{2}) & , n \Rightarrow even \\ x \cdot pow(x,\frac{n}{2}) \cdot pow(x,\frac{n}{2}) & , n \Rightarrow odd \end{cases}$$

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

recurrence relation

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + 1$

$T(n) = 2T(\frac{n}{2}) + 1$

$2.\ T(\frac{n}{2}) = 4.T(\frac{n}{4}) + 2$

$4.\ T(\frac{n}{4}) = 8.T(\frac{n}{8}) + 4$

k steps

$T(1) =$

$T(n) = 1 + 2 + 4 + \ldots\ldots$ (k terms) — (I)

$\frac{n}{2^0} = n$

$\frac{n}{2^1} = n/2$

$\frac{n}{2^2} = n/4$

$\frac{n}{2^3} = n/8$

$\frac{n}{2^{k-1}} = 1$ — K

$\frac{n}{2^{k-1}} = 1$

$n = 2^{k-1}$

$\log_2 n = k-1$

$k = 1 + \log_2 n$ — (II)

total terms $(1 + \log_2 n)$

(I) + (II)

$T(n) = \underbrace{1 + 2 + 4 + \ldots}_{1+\log_2 n} \Rightarrow a\left[\frac{r^k - 1}{r-1}\right] \Rightarrow 2^{1+\log_2 n} - 1$

$T(n) \Rightarrow 2^1 . 2^{\log_2 n} - 1$

$T(n) \Rightarrow 2 . n^{\log_2 2} - 1$

$T(n) \Rightarrow 2.n - 1 \approx n$ ops

```java
public static int powerActualSmart(int x, int n){
    if(n == 0){
        return 1;
    }
    int xPowNBy2 = powerActualSmart(x,n/2);
    int xPowN = xPowNBy2 * xPowNBy2;
    if(n%2 == 1){
        xPowN *= x;
    }
    return xPowN;
}
```

$T(n) = T(\frac{n}{2}) + 1$

$T(n) = T(\frac{n}{2}) + 1$

$T(\frac{n}{2}) = T(\frac{n}{4}) + 1$

$T(\frac{n}{4}) = T(\frac{n}{8}) + 1$

$T(\frac{n}{8}) = T(\frac{n}{16}) + 1$

.

.

.

$T(1) = T(0) + 1$

k steps

$T(n) = 1 + 1 + 1 + 1 + 1 \ldots + 1 = k$

k times

k steps =

$n = n/2^0$

$n/2 = n/2^1$

$n/4 = n/2^2$

.

.

.

$1 = n/2^{k-1}$

$\frac{n}{2^{k-1}} = 1$

$k = \log_2 n + 1$

$k \propto \log_2 n$

$\log_2 n + 1$

$T(n) = \log_2 n + 1$