Principle of mathematical Induction

Expectation $1 + 2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2}$

faith → it is true for $n = k$

$1 + 2 + 3 + 4 + \cdots + k = \frac{k(k+1)}{2}$

put the of $n = k+1$

L.H.S ⇒

$$1 + 2 + 3 + 4 + \cdots + k + k+1$$

$$= \frac{k(k+1)}{2} + k+1$$

$$= (k+1)\left[\frac{k}{2} + 1\right]$$

$$= \frac{(k+1)\left[(k+1)+1\right]}{2}$$

Check for $n = 1$

R.H.S → $\frac{n(n+1)}{2}$, $n \to k+1$

$$\frac{(k+1)\left[(k+1)+1\right]}{2}$$

L.H.S = R.H.S. Hence proved

High level analysis

→ Faith, and Expectation,

→ Merging of faith and expectation

→ Flow of Recursive code

Low level Analysis

→ Base case

→ Understanding of function calls and variables of stack (Recursive)

* Faith is smaller problem than Expectation

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

$0! = 1$

$n! = n \times (n-1)!$

Example: $n = 5$.

Golden → {If is Recursion is Returning something, either store it or use it.}

Expectation, $fact(n) = n \times n-1 \times n-2 \times \dots 1$

$5 \times 4 \times 3 \times 2 \times 1 = 120$

faith → $fact(n-1) = n-1 \times n-2 \times n-3 \times \dots 1$

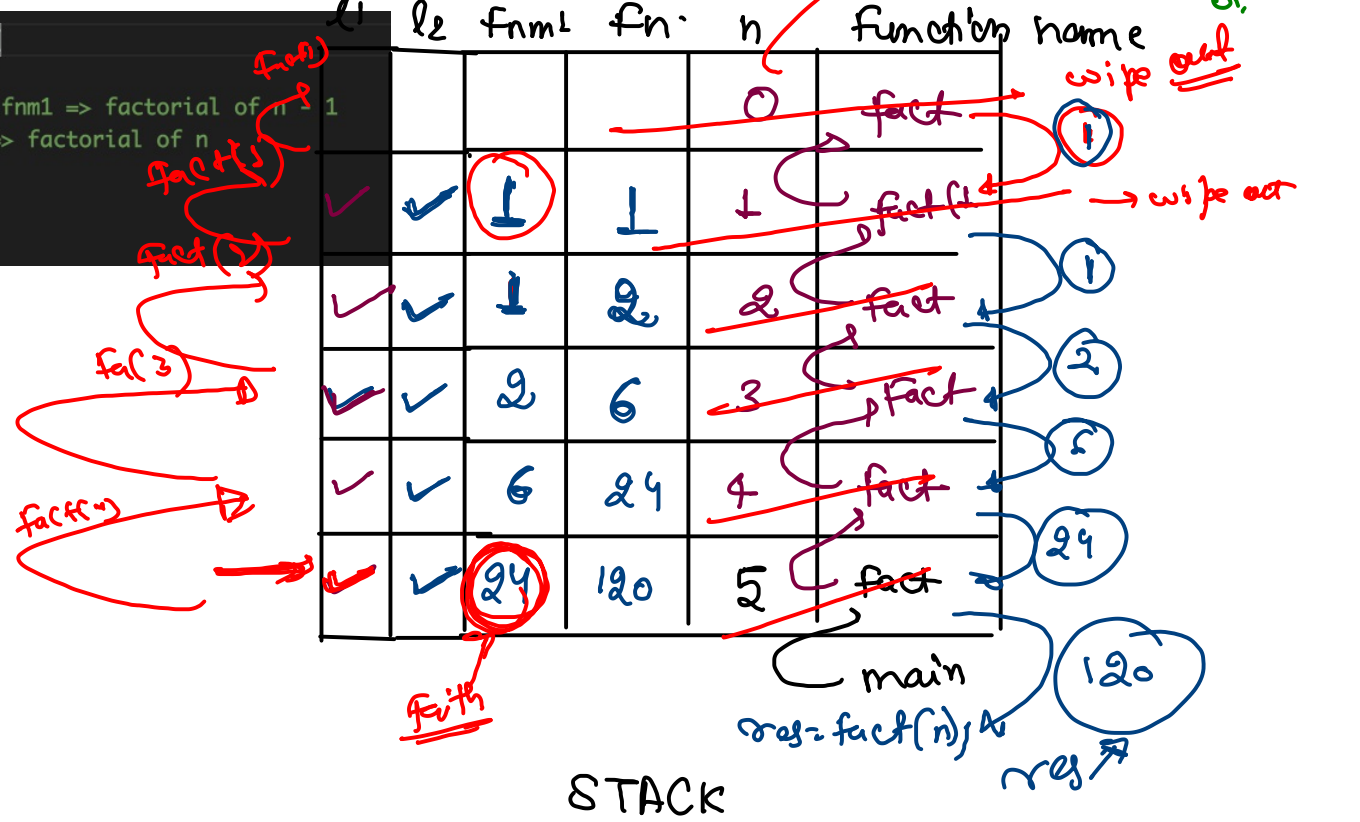
Merging of faith and Expectation,

$fact(n) = n \times \underbrace{n-1 \times n-2 \times \dots}_{fact(n-1)} \times 1$

General term → Expectation $fact(n) = \underbrace{n \times}_{\text{Done by myself}} \underbrace{fact(n-1)}_{\text{Done by Recursive}}$

low level analysis

```
public static int fact(int n) {
    // 2. faith
1.  int fnm1 = fact(n - 1); // fnm1 => factorial of n - 1
2.  int fn = fnm1 * n;  // fn => factorial of n

    return fn;
}
```



Linear - Power. (High level)

Power - $x \wedge n$. $x = 2,\ n = 5$

Expectation, power(2, 5) $\rightarrow 2^5 \Rightarrow 2 \times 2 \times 2 \times 2 \times 2 = 32$.

faith power(2, 4) $\rightarrow 2^4 \rightarrow 2 \times 2 \times 2 \times 2$

Merging of faith and Expectation,

power(2,5) = $\underbrace{2 \times 2 \times 2 \times 2 \times 2}$

power (2,5) = power (2,4) * 2.

general term, power(x, n) = power(x, n-1) * x;

Linear Power (Loco level Analysis)

```java
public static int power(int x, int n) {
    // faith
    int xnm1 = power(x, n - 1); // xnm1 = x ^ (n - 1)
    int xn = xnm1 * x;

    return xn;
}
```

$x = 2$

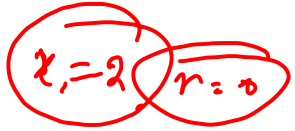
$n = 5$

Base case →

if (n == 0) {

return 1;

}

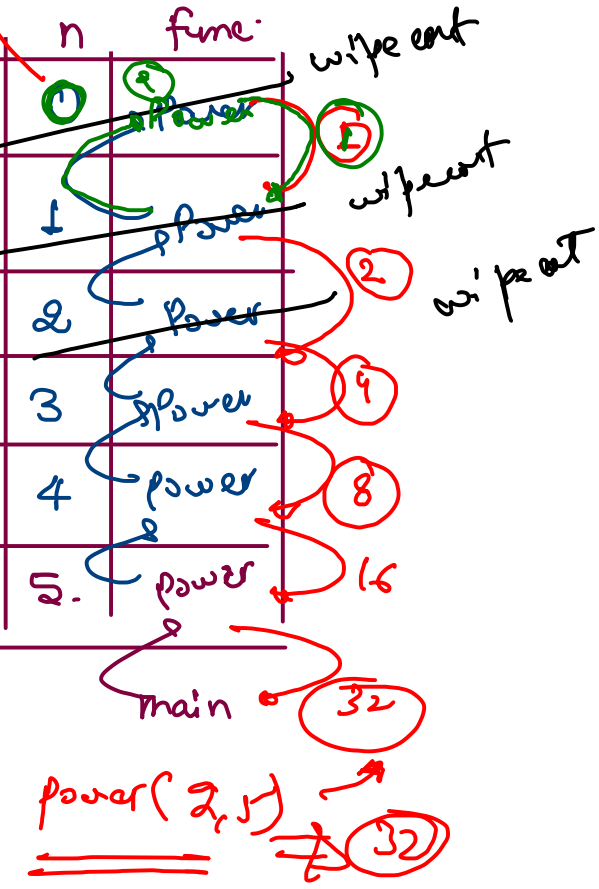


$x^0 = 1$

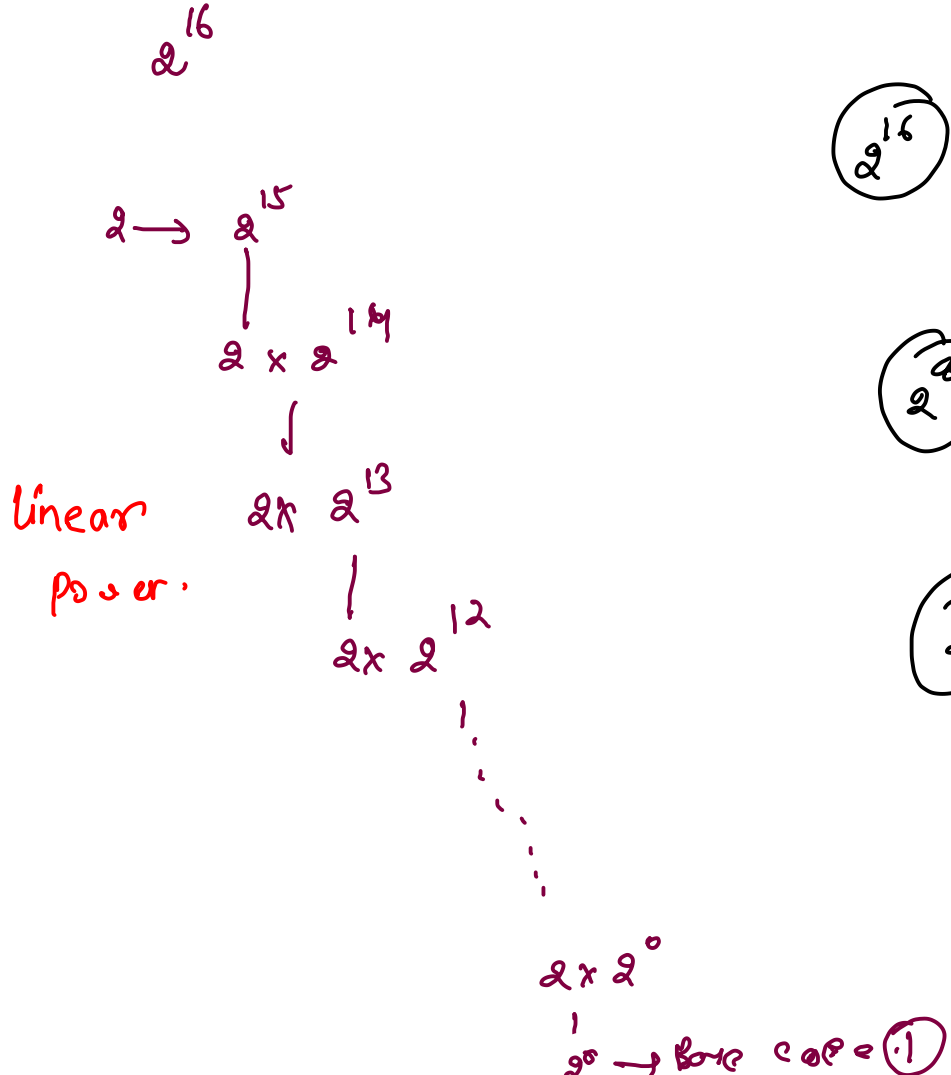
Smallest problem

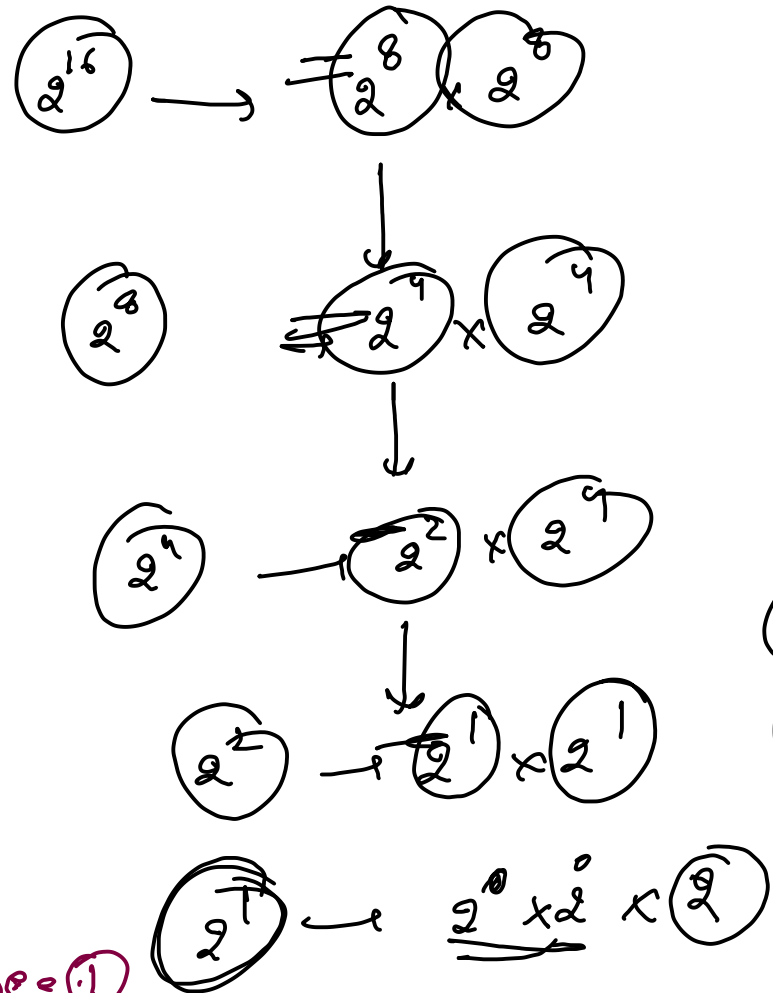
Base case

l_1	l_2	l_3	xnm1	xn	n	func.
					0	power
✓	✓	✓	1	2	1	power
✓	✓	✓	2	4	2	power
✓	✓	✓	4	8	3	power
✓	✓	✓	8	16	4	power
✓	✓	✓	16	32	5	power



power logarithm.





$2^{23} \longrightarrow 2^{11} \times 2^{11} \times 2$

$2^{11} \longrightarrow 2^{5} \times 2^{5} \times 2$

$2^{5} \longrightarrow 2^{2} \times 2^{2} \times 2$

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

$2^{1} \longrightarrow 2^{0} \times 2^{0} \times 2$

$2^{0} \longrightarrow 1$ Base

Expectation : pow(x, n) →

faith → pow(x, n/2).

Merging →

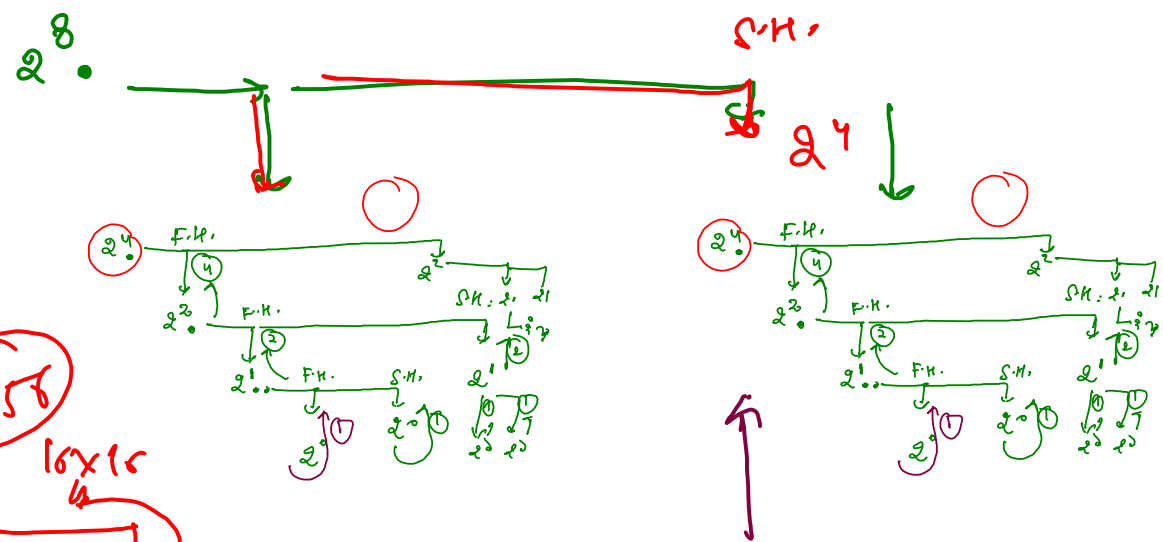
pow(x, n) = x^n/2 * x^n/2

power(x, n) = faith * faith var.

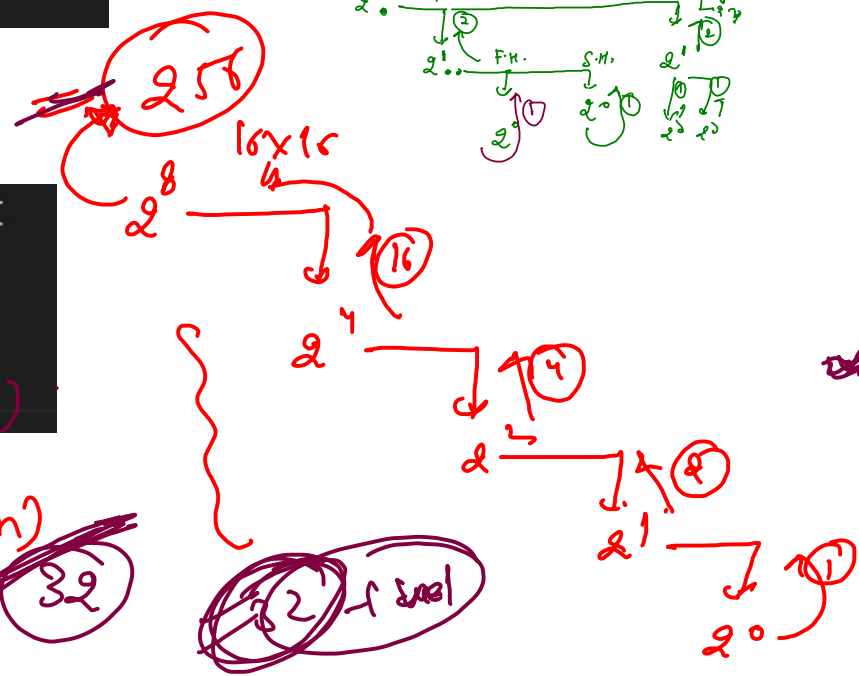
2^8

x^n = powerFakeBtr$(x, n/2)$ * powerFakeBtr$(x, n/2)$

```java
public static int powerFakeBtr(int x, int n) {

    int firstHalf = powerFakeBtr(x, n / 2);
    int secondHalf = powerFakeBtr(x, n / 2);

    int xn = firstHalf * secondHalf;

    return n % 2 == 0 ? xn : xn * x;
}
```




```java
public static int powerBtr(int x, int n) {
    int halfPower = powerBtr(x, n / 2);
    int xn = halfPower * halfPower;

    return n % 2 == 0 ? xn : xn * x;
}
```



$\log_2(n) = \log_2(10^9)$

$= \log_2(2^{32})$

$\log(n) = 32$

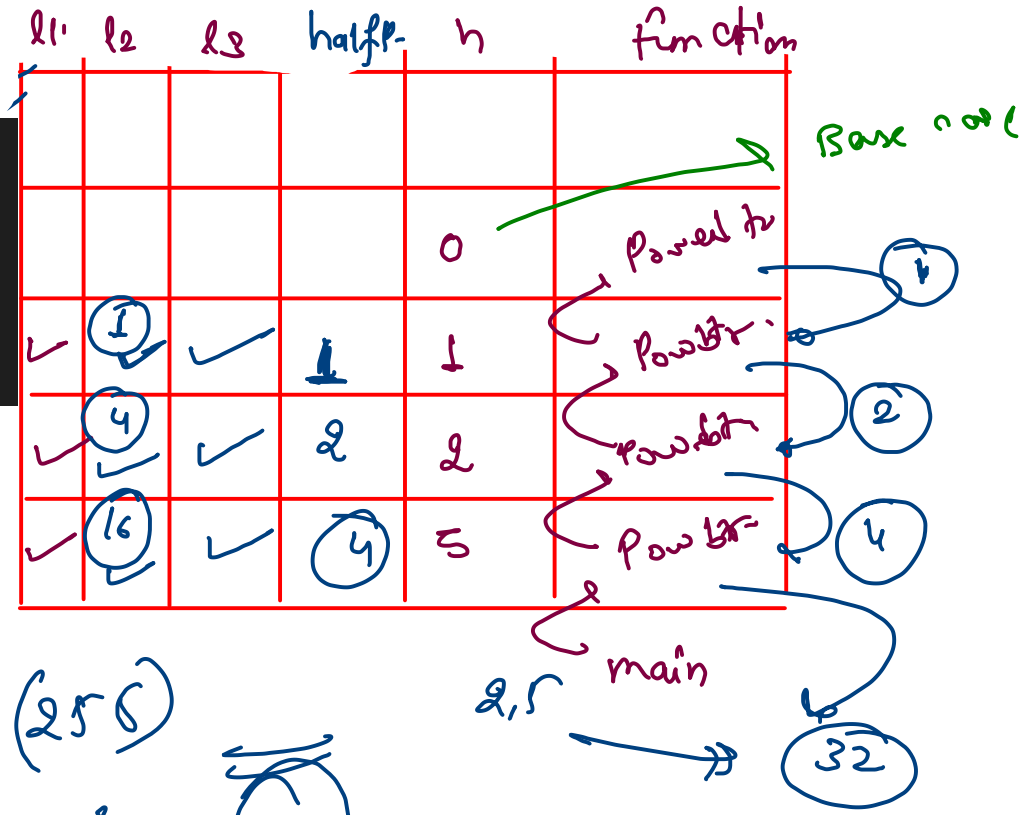
x^n 1000 000 000

n.

Level $\log 10^9$

O(n) JAVA

```java
public static int powerBtr(int x, int n) {
    int halfPower = powerBtr(x, n / 2);
    int xn = halfPower * halfPower;

    return n % 2 == 0 ? xn : xn * x;
}
```



2^5

$256 = 2^8$

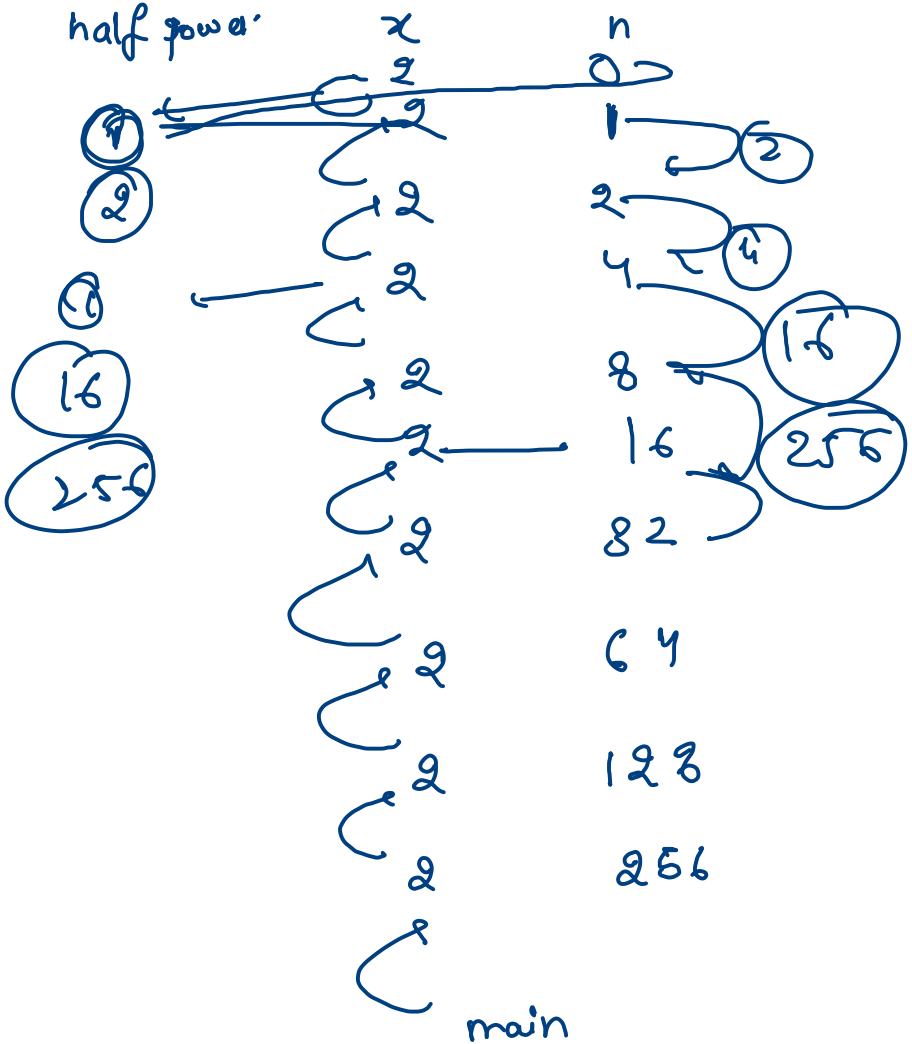
$2 \rightarrow$

$\Rightarrow$

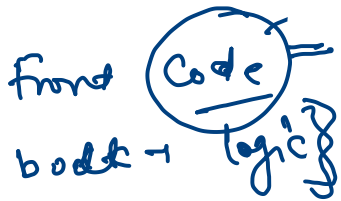
$\log(256)$

$\log(2^8) = 8$

Linear Powers 32



Tree (1) → Code printed



Print Zig Zag :- (High level Analysis) Expectation

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

pzz(3) → 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

faith · pzz(2) = 2 1 1 1 2 1 1 1 2

Merging →

pzz(3) = 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

pzz(3) = 3 | pzz(2) | 3 | pzz(2) | 3

- 3 : print by myself
- pzz(2) : Done by Recursion
- 3 : print by myself
- pzz(2) : Done by Recursion
- 3 : print by myself

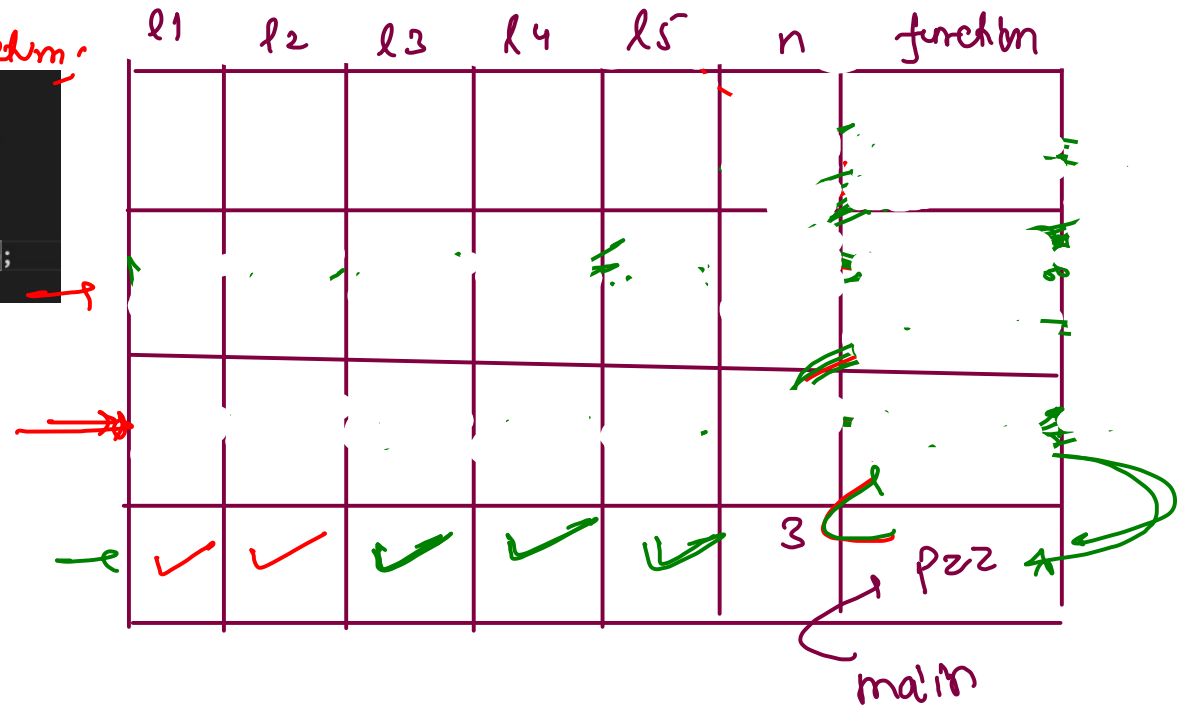
pzz(n) =
- syso(n)
- pzz(n-1)
- syso(n)
- pzz(n-1)
- syso(n)

pzz low level Analysis

Base case -

if(n==0) return;

```
public static void pzz(int n) {
1.  System.out.println("pre : " + n);
2.  pzz(n - 1);
3.  System.out.println("in : " + n);
4.  pzz(n - 1);
5.  System.out.println("post : " + n);
}
```



Console -

pre 3
pre 2
pre 1
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
in 3

> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

Sirf 1s

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

Euler traversal / Euler Diagram

Recursive Code.

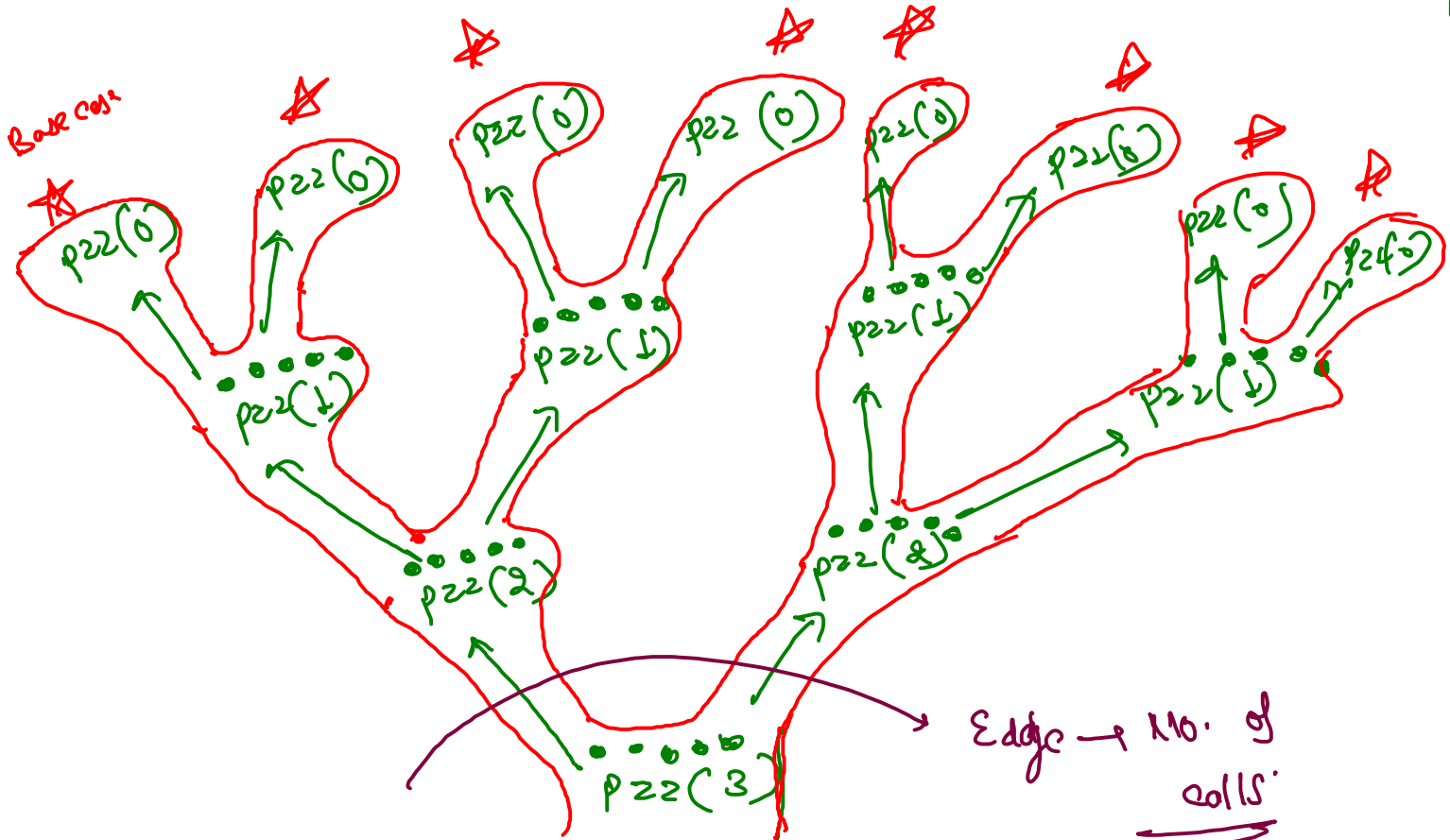
```
fun( ) {
  // Before all calls-
  → Pre Area } Pre. work
  Call 1
  → // In 1 — Area
  Call 2
  → // In 2 — Are.        } In Area
  Call 3
  // After all calls
  → Post Area } Post-work
}
```

Print Decreasing - Pre work

Print Increasing + Post work

* In Area is possible if calls are more than 1

```java
public static void pzz(int n) {
    if(n == 0) return;
1. System.out.println("pre : " + n);
2. pzz(n - 1);
3. System.out.println("in : " + n);
4. pzz(n - 1);
5. System.out.println("post : " + n);
}
```



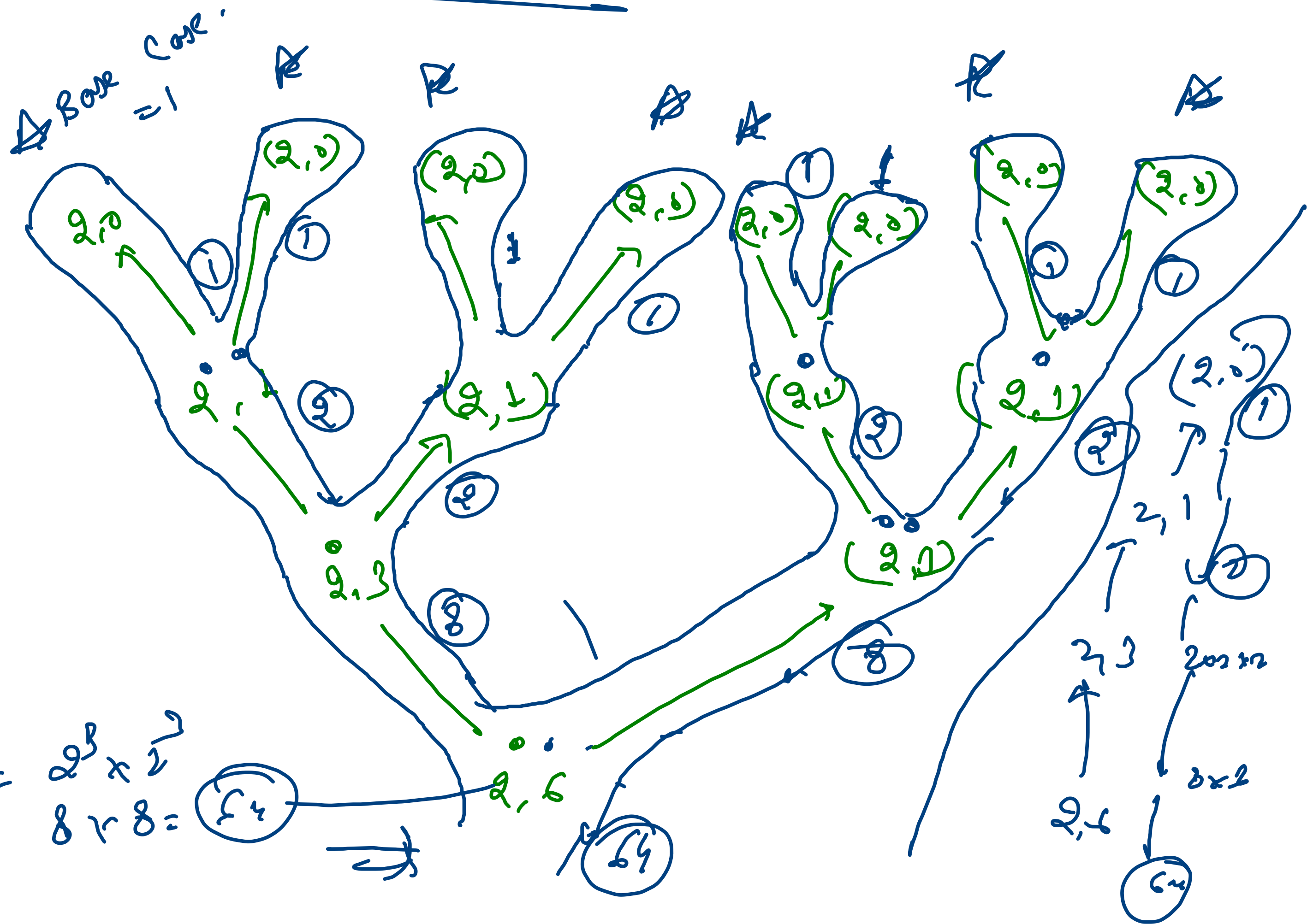
pre 3
pre 2
pre 1
in 1
post 1
in 2
pre 1
in 1
post 1
post 2
in 3

pre 2
pre 1
in 1
post 1
in 2
pre 1
in 1
post 1
post 2
post 3

Dry Run → power fake btr.

Euler Diagram
Euler tree

```
int fun(x, n) {

1. Half(1) = fun(x, n/2);
2. Half(2) = fun(x, n/2);

   x^n = Half1 * Half2;

   return conditional (Depend of return power)

}
```



$2^6 = 2^3 \times 2^3$

$8 \times 8 = 64$

Home work

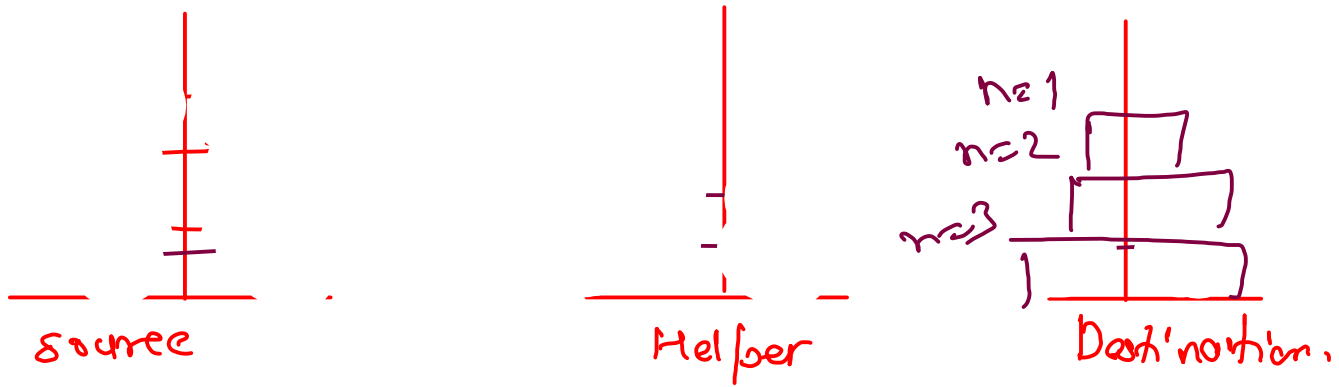
n = 5

```
(*) ps void fun(int n) {
        if(n <= 0)
          return;
        syso(n)
        fun(n-1)
        syso(n)
        fun(n-2)
        syso(n)
        fun(n-3)
        syso(n
    }
```

(*) Tower of Hanoi

Tower of Hanoi

n disks.



Print steps to move n disks from source to destination with following Rules.

Rules:-

1. Pick one disk at a time
2. Heavy disk can't go to light disk.
3. Min. Moves.

1 → source → destination
2 → source → helper
1 → destination → helper
3 → source → destination
1 → helper → source
2 → helper → destination
1 → source → destination