Recursion → When a function call itself. This scenario is called Recursion.

Principle of Mathematical Induction :→

Expectation → $1+2+3+4+5+\dots+n = \frac{n(n+1)}{2}$, check for $n=1$

L.H.S : (1), R.H.S : $\frac{1\times 2}{2}$: (1)

faith → let's assume that, it is true for $n=k$;

$$1+2+3+4+5+\dots+k = \frac{k(k+1)}{2} \quad \text{——— (1)}$$

merging of faith & expectation

Solve for $(n=k+1)$

L.H.S.

$$\underbrace{1+2+3+4+\dots+k}+(k+1)$$

from eq (1)

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

R.H.S.

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

hence proved. L.H.S = R.H.S.

① print decreasing :→

pd(n=5) → 5 4 3 2 1

Recursion

High level Analysis (over view from top)

1. Expectation.
2. faith
3. Merging of faith & Expectation.

low level Analysis (deep understanding of Rec)

1. Stack diagram
2. Base case.
3. Code / Recursion flow

Expection → Solution of problem, which we want to solve.

faith → small portion of Expectation / smallest problem than Expectation.

printdecreasing (n=5) → 5 4 3 2 1.

High level Analysis

① Expectation → pd(5) = 5 4 3 2 1

② faith → pd(4) = 4 3 2 1 } Recursion

③ Merging of faith & Expectation

pd(5) = 5 4 3 2 1.

pd(5) : syso(5) pd(4)

pd(5) ← Expectation

syso(5) done by myself

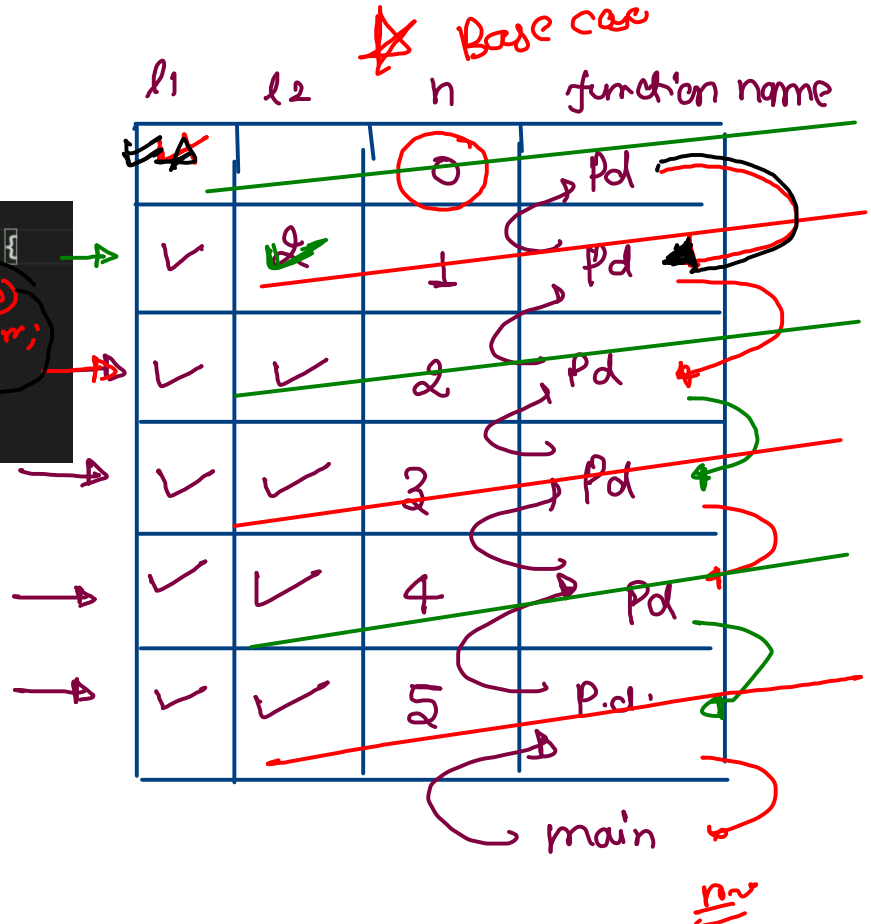
pd(4) done by Recursion

In general term → pd(n) = syso(n) pd(n-1) = }

Low level Analysis

```java
public static void printDecreasing(int n) {
    // done by myself
1.  System.out.println(n);
    // faith
2.  printDecreasing(n - 1);
}
```

if(n == 0) return;




```
Console -
5
4
3
2
1
```

```
main() {
    pr.dec(5);
    syso("Hello World");
}
```

Code - flow ✓

Stack Diagram ✓

Base case ✓

5 4 3 2 1

Hello World

print Increasing. (High level Analysis)

printIncreasing(n=5) ⇒ 1 2 3 4 5

Expectations, pi(5) ⇒ 1 2 3 4 5

faith, pi(4) ⇒ 1 2 3 4 } Recursion.

Merging of faith & Expectation

pi(5) = 1 2 3 4 5

pi(5) = pi(4) Syso(5)

pi(4) — Done by Recursion

Syso(5) — Done by myself

General term

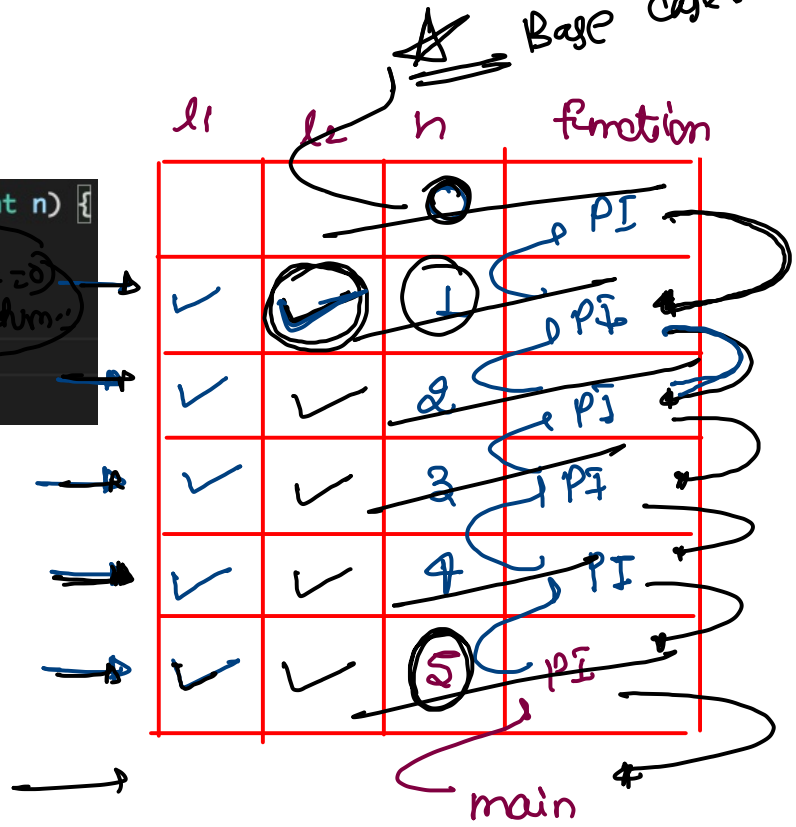
pi(n) = pi(n-1) Syso(n);

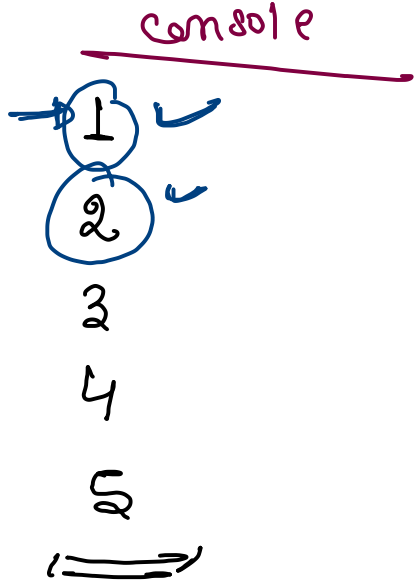
pi(n-1) — Re

Low Level Analysis

```
public static void printIncreasing(int n) {
    // faith
1.  printIncreasing(n - 1);
    // my work
2.  System.out.println(n);
}
```

if(n==0) return..





Returning condition from function

→ (I) Return keyword STACK

→ (II) All lines are Executed

print decreasing Increasing → (High level Analysis)

pdi(5) → 5 4 3 2 1 1 2 3 4 5

pdi(3) → 3 2 1 1 2 3

Expectation → pdi(5): 5 4 3 2 1 1 2 3 4 5

faith → pdi(4) = 4 3 2 1 1 2 3 4

Merging of faith & Expectation

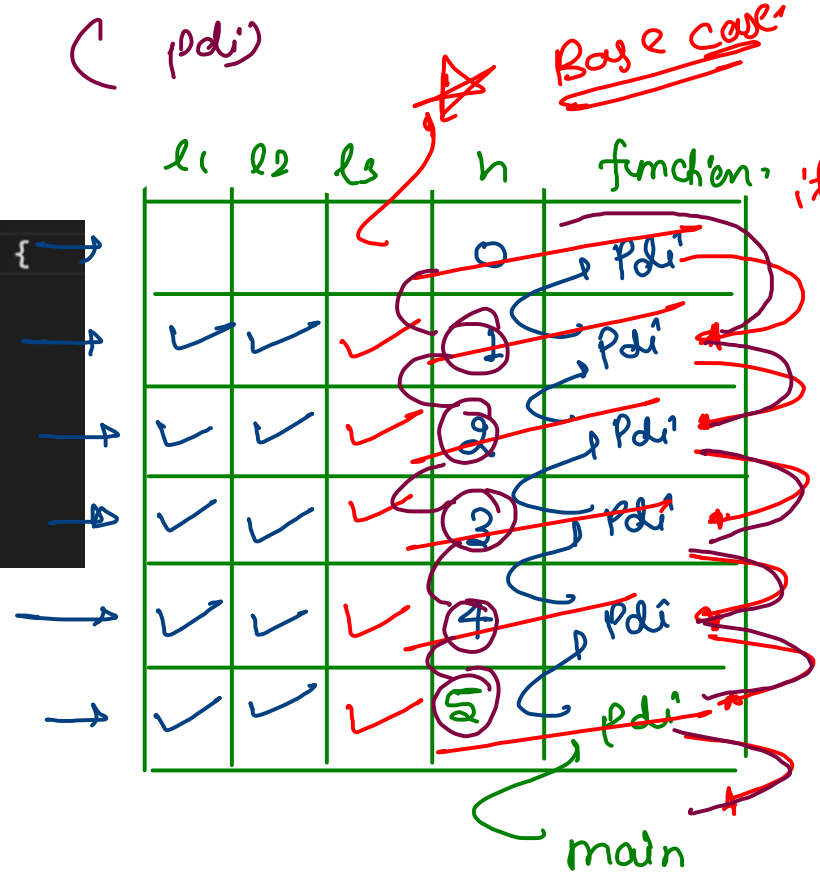
pdi(5): 5 | 4 3 2 1 1 2 3 4 | 5

pdi(5) = Sys.o(5) | faith pdi(4) | Sys.o(5)

pdi(n): Sys.o(n) | pdi(n-1) | Sys.o(n)

Low level Analysis (pdi)

```java
public static void pdi(int n) {
    // done by myself
1   System.out.println(n);
    // faith on recursion
2   pdi(n - 1);
    // done by myself
3   System.out.println(n);
}
```



Single → Multiple time visit

Console

5
4
3
2
1
1
2
3
4
5

STACK

Return factorial (High level)

factorial → $n! = n \times (n-1) \times (n-2) \times (n-3) \times (n-4) \times \dots \times 1$

Expectation fact(5) = 5! ☛ (120)

faith. fact(4) = 4! } Recursion.

Merging of faith & Expectation

fact(5) = 5 x 4!

fact(5) = (5) * fact(4)

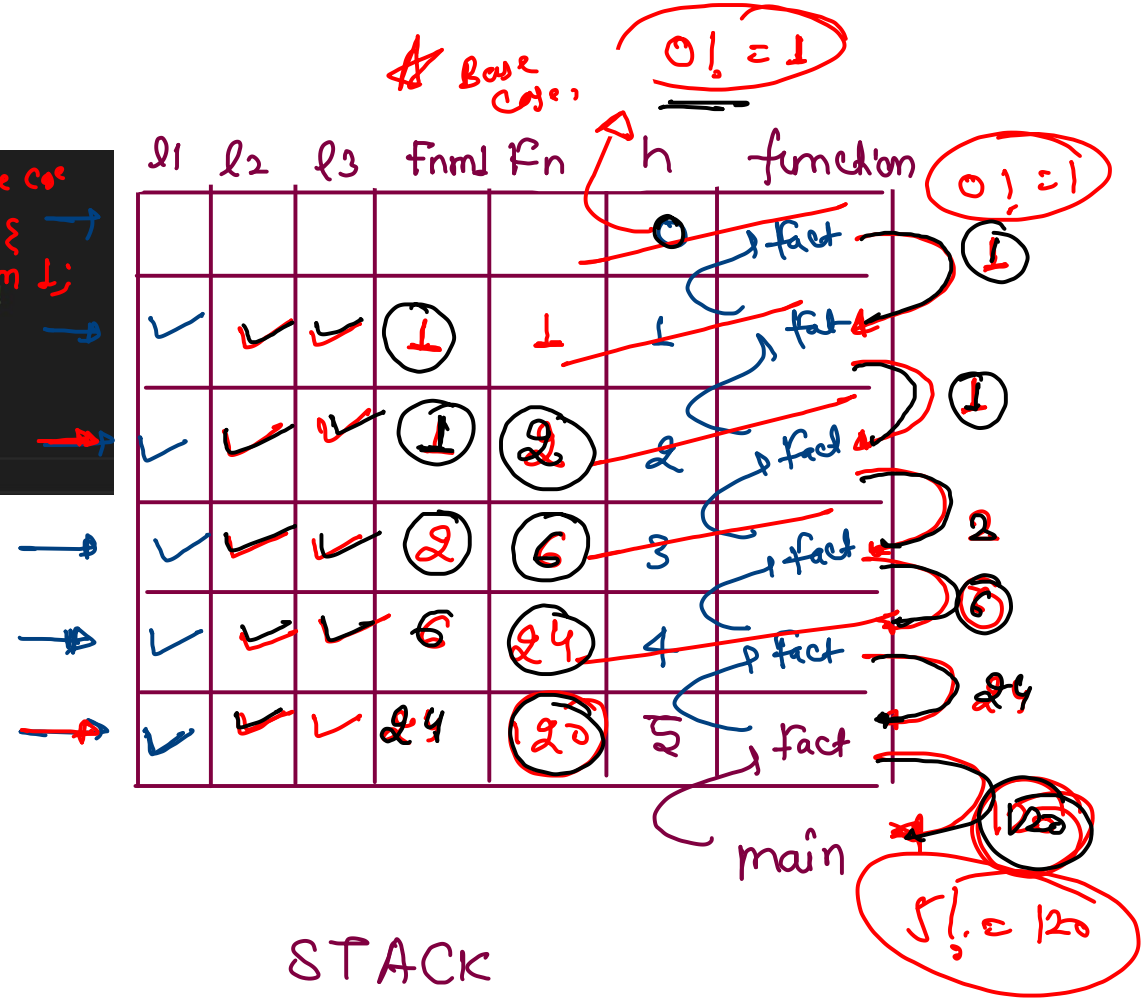
Done by Myself — Done by Recursion.

Golden (*) [Whenever function is return something, either store and use it, OR directly use it]

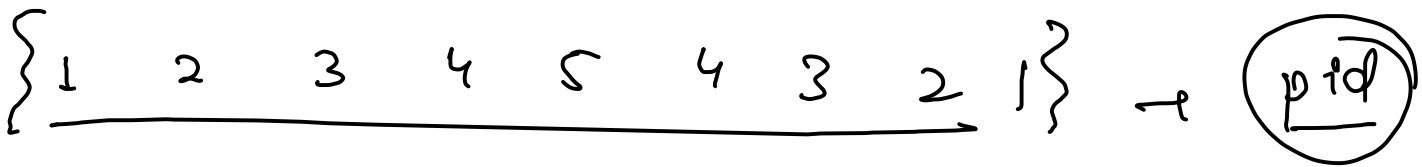
Low level Analysis

```java
public static int factorial(int n) {

    // faith
    int fnm1 = factorial(n - 1); // (n - 1)!
    // merging
    int fn = n * fnm1; // n * (n - 1)!

    return fn; // n!
}
```

Base case
if(n==0){ return 1; }



STACK



Print Increasing Decreasing - (High Level Analysis)

Expected... pid(5, 1) → 1 2 3 4 5 4 3 2 1

faith. pid(5, 2) → 2 3 4 5 4 3 2

Merge of faith and expectation

pid(5, 1) = (1) 2 3 4 5 4 3 2 (1)

pid(5, 1) → Sysо(1) pid(5, 2) Sys(0)

In general term

pid(n, i) = Sysо(i) pid(n, i+1) Sysо(i)

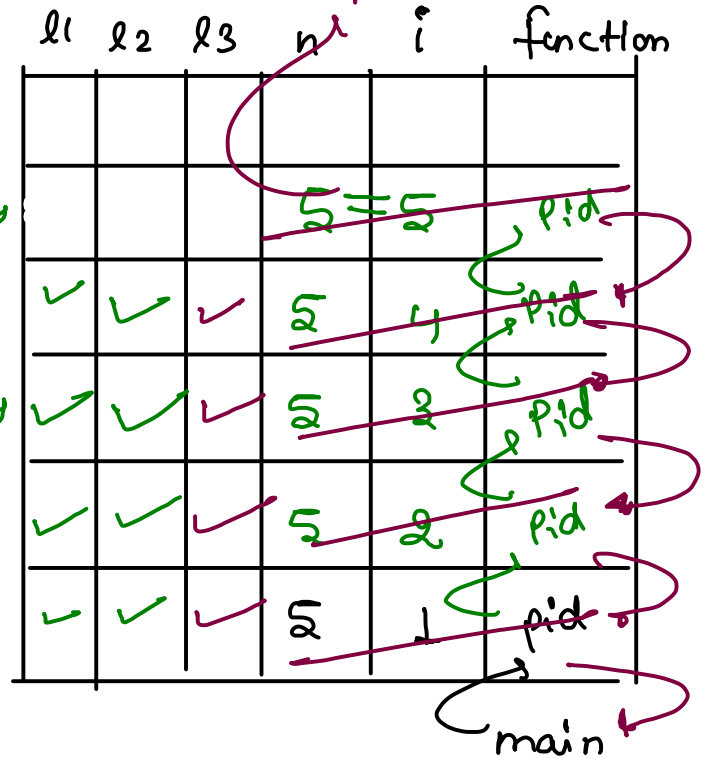
pid (low level Analysis)

```
public static void pid(int n, int i) {
    // my work
1.  System.out.println(i);
    // faith
2.  pid(n, i + 1);
    // my work
3.  System.out.println(i);
}
```

Base case
if(n==i){
    Sysо(n);
    return;
}

Print

★ Base case → print return.



Console-

1
2
3
4
5
4
3
2
1
=

power - linear (High level Analysis)

x^n $x = 2$, $n = 5$ 2^5

Expectation, power(2, 5) = 2^5

faith, power(2, 4) → 2^4 } Recursion.

Merging power(2, 5) = 2 × 2^4

power(2, 5) = xnm1 = power(x, n-1) (2, 4);

power(x, n) = 1

xn = xnm1 * x

redun xn.

low level Analysis

```
public static int power(int x, int n) {

    int xnm1 = power(x, n - 1); // faith
    int xn = xnm1 * x;
    return xn;
}
```

x = 2

l1	l2	l3	xnm1	xn	n	function
					0	power = 0
✓	✓	✓	1	2	1	Power (2)
✓	✓	✓	2	4	2	Power (2)
✓	✓	✓	4	8	3	Power (2)
✓	✓	✓	8	16	4	Power (2)
✓	✓	✓	16	32	5	Power (2)
						main

$x^0 = 1$

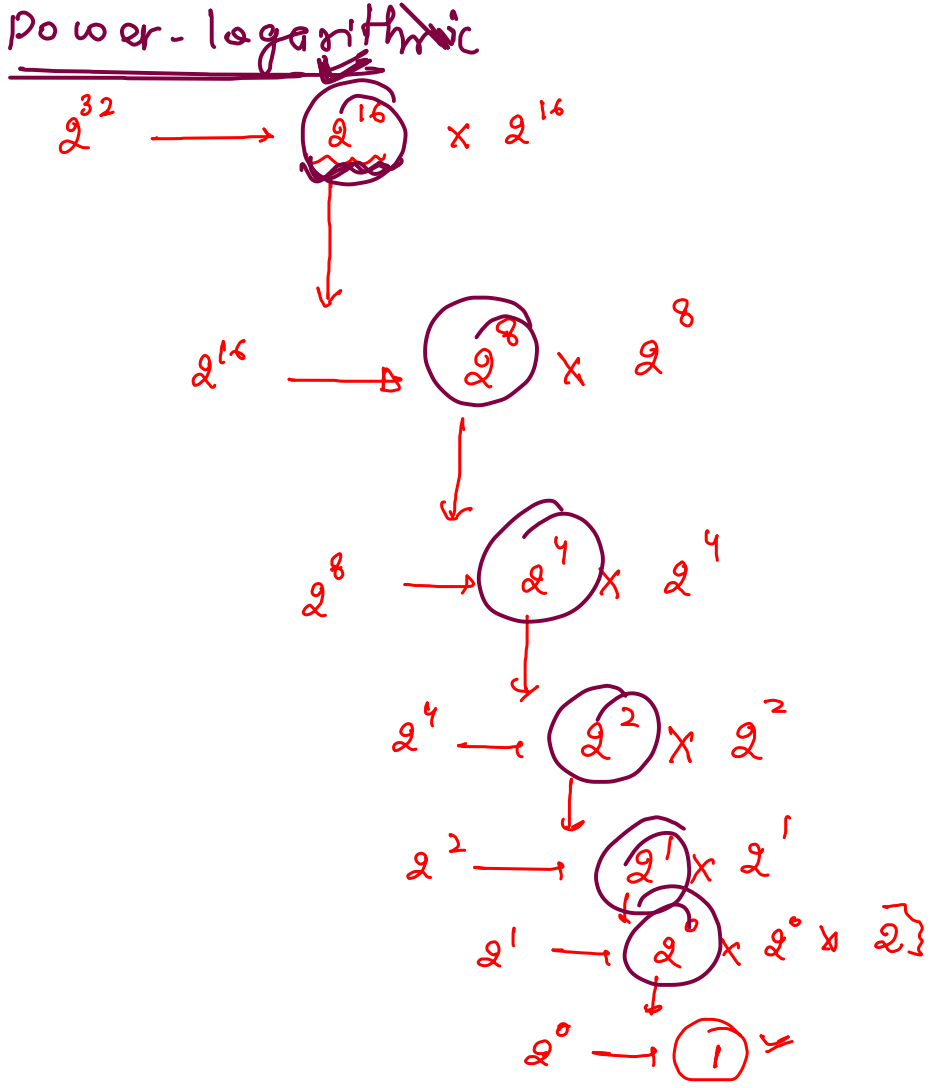
Base case

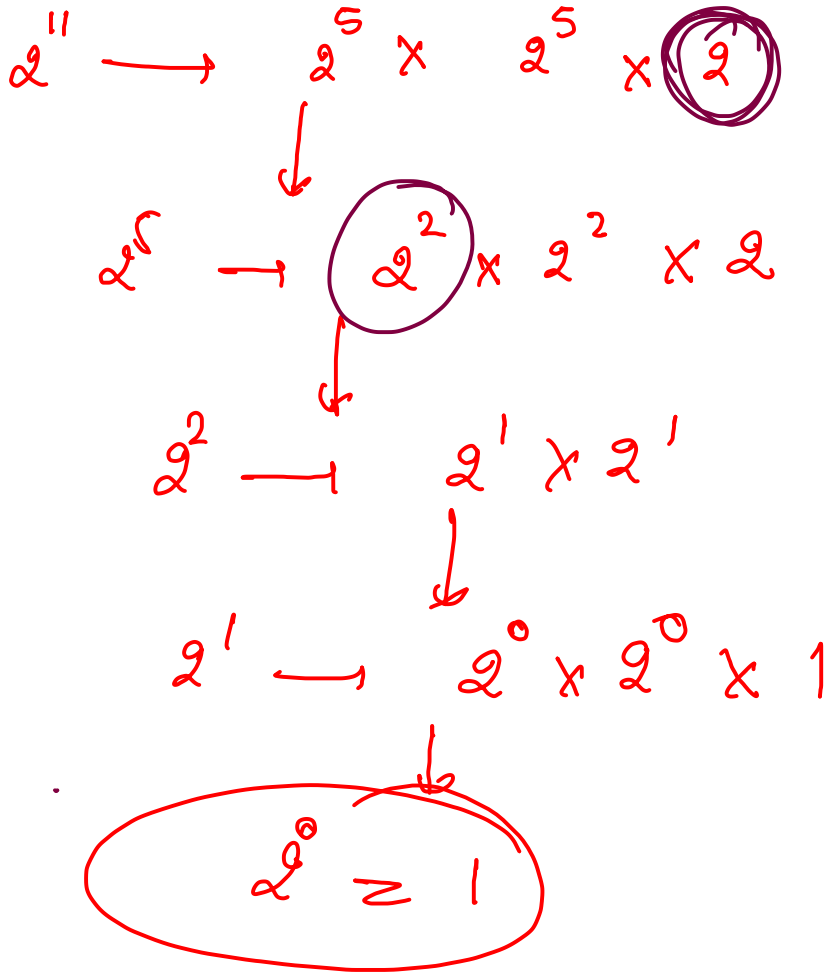
```
if(n == 0)
    return 1;
```

$x^0 = 1$

l's = 1

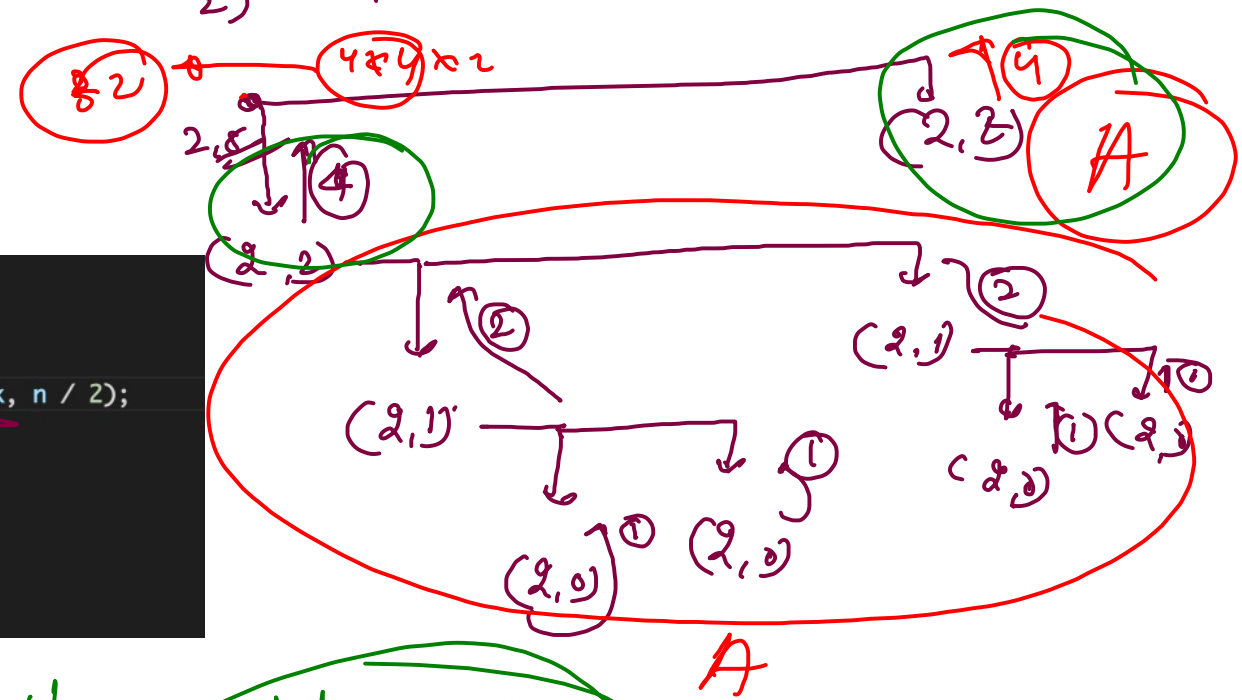
Power - logarithmic





Expectation: pow(x, n) = x^n

faith → pow(x, n/2) → half power




```java
public static int powerFakeBtr(int x, int n) {
    if(n == 0) return 1;

    int xn = powerFakeBtr(x, n / 2) * powerFakeBtr(x, n / 2);

    if(n % 2 != 0) {
        xn *= x;
    }
    return xn;
}
```

T(n)

Complexity = linear power

log n → Power better.

```
public static int powerBtr(int x, int n) {
    if(n == 0) return 1;

    int halfPower = powerBtr(x, n / 2);
    int xn = halfPower * halfPower;
    if(n % 2 != 0) {
        xn *= x;
    }
    return xn;
}
```

log n

Sir discuss

Sumeet

linear power → 32 level

Smart power.

2^{32} → 2^{16} → 2^{8} → 2^{4} → 2^{2} → 2^{1} → 2^{0}

256, 16, 4, 2, 1

$\log(32) = \log_2(2^5) = 5 \log_2 2$

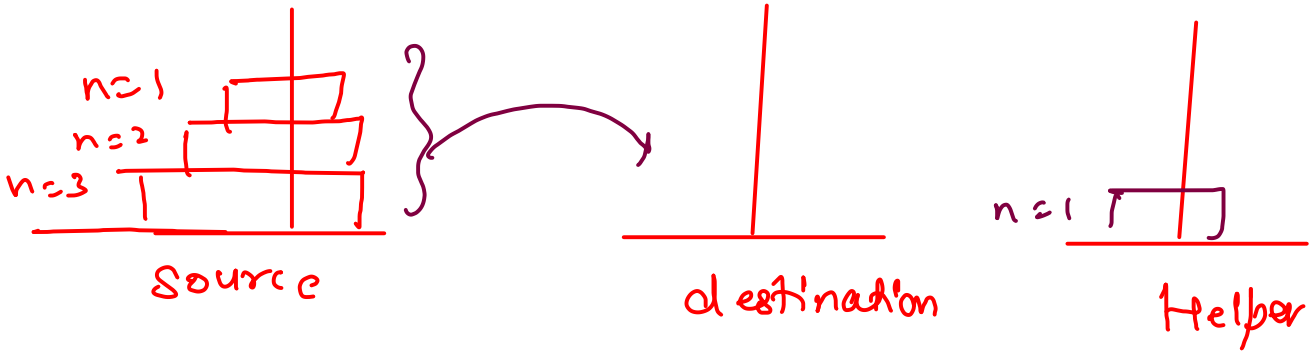
times taken

$n >>> \log n$

$= 5$

n discs.

n=3

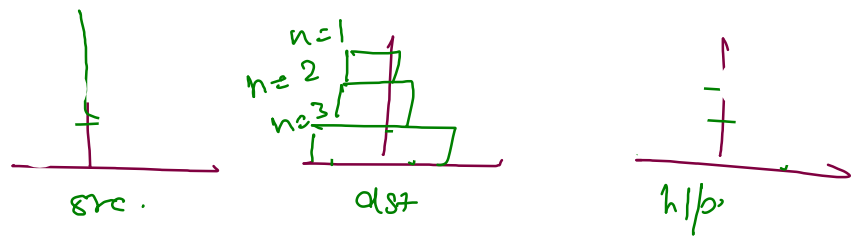


print steps to
move n discs from source
to destination using Helper
with following two rules

(1) → move 1 disc at a time

(2) → Heavy disc on light is not allowed

(3) → steps should be min.



Steps.

(1) move 1 src to dst
(2) move 2 src to hlp
(3) move 1 dst to hlp
(4) move 3 src to dst
(5) move 1 hlp to src
(6) move 2 hlp to dst
(7) move 1 src to dst