

## Recursion & Backtracking

## Principle of Mathematical Induction (PMI)

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

### ① Trivial Case

$$\sum_{i=1}^0 = \frac{0(0+1)}{2} = 0$$

$$\sum_{i=1}^1 i = 1 = \frac{1(1+1)}{2} = 1$$

$$\textcircled{2} \quad \text{Assume} \quad \sum_{i=1}^k i = \frac{k(k+1)}{2}$$

$$\textcircled{3} \text{ We have to prove for } k+1 \text{ RMS should be } \frac{(k+1)}{2}$$

$$\sum_{i=1}^{k+1} i^2 = 1^2 + 2^2 + \dots + k^2 + (k+1)^2$$

$$= k(k+1) + (k+1) = (k+1)(k+2)$$

Since  $\|M_N - R_N\| \rightarrow 0$  as  $N \rightarrow \infty$  (our assumption holds true),

Since Lassertis given a  
by ENI  $\exists i \in n^{(n+1)}$

Recursion (Example of printing)  
Recursive function: expectation  
 and writing (idn)  $\rightarrow 1, 2, 3, \dots, n$

### Meeting Expectation with faith

```

void printc( char> f {  

    if (n == 0) return;  

    cout << printc(n-1);  

    cout << c[n];  

}  

int main() {  

    cout << printc(5);  

}

```

# Low-Level Thinking

function Call shark  
P1(4) [ ] n  
P2(5) [ ] n  
P3(6) [ ] n  
P4(7) [ ] n

The diagram illustrates a function call stack with four frames:

- f1(a)**: The top frame, containing the value **a**.
- f2(b)**: The second frame from the top, containing the value **b**.
- f3(c)**: The third frame from the top, containing the value **c**.
- f4(d)**: The bottom frame, containing the value **d**.

function call stack

## function call stack

The diagram illustrates the state of memory during a function call. A vertical blue line on the left represents the stack. To its right, the text "function Call stack" is written above a horizontal line. Below this line, the word "Program" is written above another horizontal line, which is labeled "Terminates".

## one Print Decreasing

Example
Sample Input
5
Sample Output
5 4 3 2 1

① Expectation:

$\text{printDec}(n) : n, n-1, n-2 \dots 1$

② faith:

$\text{printDec}(n-1) : n-1, n-2, n-3 \dots 1$

③ Meet expectation with faith:

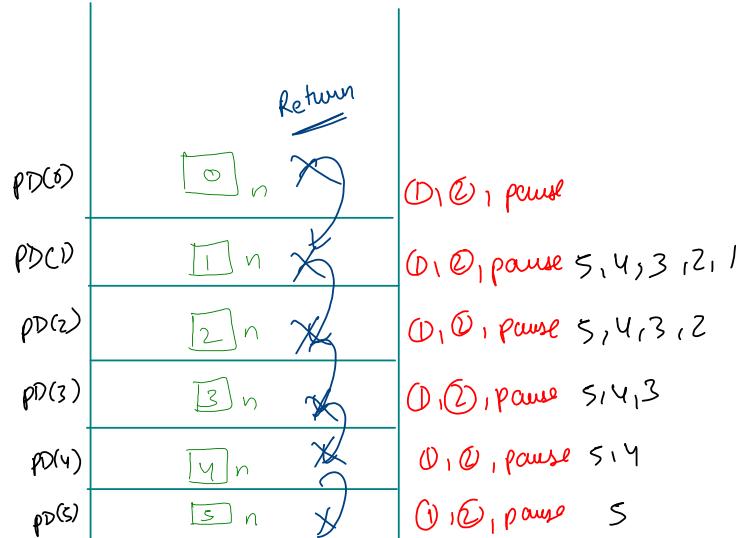
$\text{Syso}(n);$   
 $\text{printDec}(n-1);$

acc to MLT

```
void printDec(int n)
{
    ① Syso(n);
    ② printDec(n-1);
}
```

Head Rec

work is done before  
calling functions



## Ques) Print Decreasing Increasing

**Example**  
 Sample Input  
 3  
 Sample Output  
 3  
 2  
 1  
 1  
 2  
 3

① *Expectation:*

$pDecInc(n) \rightarrow$

n
n-1
n-2
:
1

② *faith:*

$3 \rightarrow 3, 2, 1, 1, 2, 3$   
 $2 \rightarrow 2, 1, 1, 1, 2$   
 $pDecInc(n-1) \rightarrow n-1, n-2 \dots 1, 1, 2, \dots$

n
n-1
n-2
:
1
2
:
n-1
n

③ Meet Expectation with faith:

Print n  
 call (n-1)  
 Print n

```
public static void pdi(int n){  

    if(n == 0) return;  

    System.out.println(n); > Pre  

    pdi(n-1);  

    System.out.println(n); > Post  

}
```

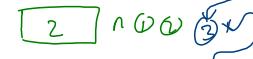
$pDI(0)$



$pDI(1)$



$pDI(2)$



$pDI(3)$



$pDI(4)$



4, 3, 2, 1  
 4, 3, 2  
 4, 3  
 4,

4, 3, 2, 1, 1  
 4, 3, 2, 1, 1, 2  
 4, 3, 2, 1, 1, 2, 3  
 4, 3, 2, 1, 1, 2, 3, 4

main()

Ans) factorial

**Example**  
Sample Input  
5  
  
Sample Output  
120

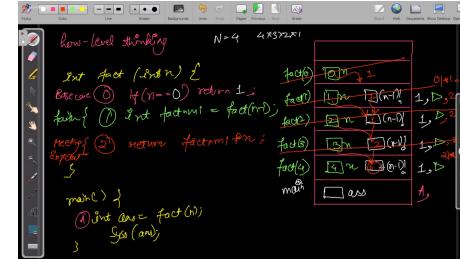
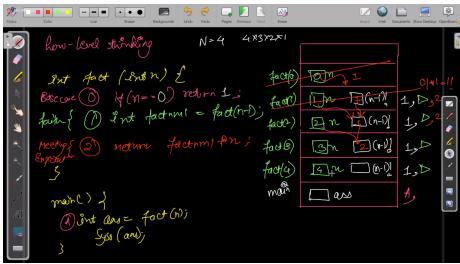
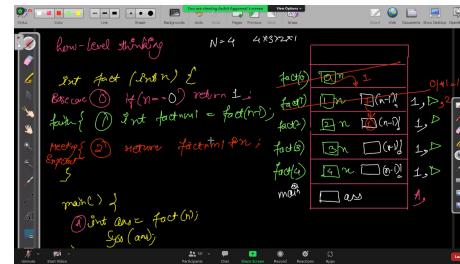
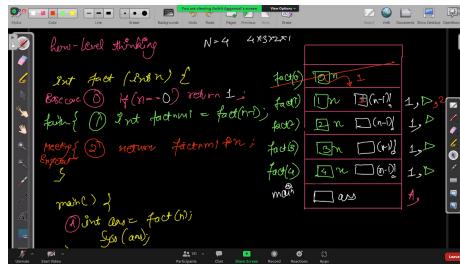
```
public static int factorial(int n){  
    if(n == 0) {  
        return 1;  
    }  
    return n*factorial(n-1);  
}
```

Expectation:  $f_{\text{act}}(n) \rightarrow n \times n! \times n^2 \times n^3 \dots$   
 faith:  $f_{\text{act}}(n \rightarrow) \rightarrow n - 1 \times n^2 \times n^3 \dots$   
 Meet Expectation with faith:

High  
Level  
Thinking

Meet expectation with faith.

$$\text{fact}(n) = n * \text{fact}(n-1)$$



Ques) Power Linear  $\Rightarrow O(n)$

$a^b \rightarrow a * a * a \dots b \text{ times}$

```
public static int power(int x, int n){  
    if(n == 0) return 1;  
    return x*power(x,n-1);  
}
```

① Expectation:  $\text{power}(x, n) \rightarrow x * x * \dots n \text{ times}$

② faith:  $\text{power}(x, n-1) \rightarrow x * x * x \dots (n-1) \text{ times}$

③ Meeting Expectation with faith:  $x * \text{power}(x, n-1)$

How-level Thinking

```
int power(int x, int n) {  
    ① if(n == 0) return 1;  
    ② int pxnml = power(x, n-1);  
    ③ return pxnml * x;  
}
```

3      +

base case

$3^5 = 243$

## Ques) Power Logarithmic

Expectation:  $\text{power}(x, n) \rightarrow x * x * x \dots n \text{ times}$

Faith:  $x * x * x \dots n^{1/2} \text{ times}$

Mutating Expectation with faith:  $x^{n/2} * x^{n/2}$

```

if ( $n \cdot 10^2 == 0$ )
    return power( $n_1, n_2$ ) * power( $x, n/2$ )
else
    return  $x * power(n_1, n_2) * power(x, n/2)$ 

```

how-level Thinking

```

int power(Int x, Int n) {
    ① if ( $n == 0$ ) return 1;
    ② Int pXnby2 = power( $x, n/2$ );
    ③ if ( $n/2 == 1$ ) return  $x * pXnby2 * pXnby2$ ;
        else return  $pXnby2 * pXnby2$ ;
}

```

$n \rightarrow 2^x \Rightarrow x \text{ no of calls}$

$$\log_2 n \leq x$$

$O(\log_2 n)$

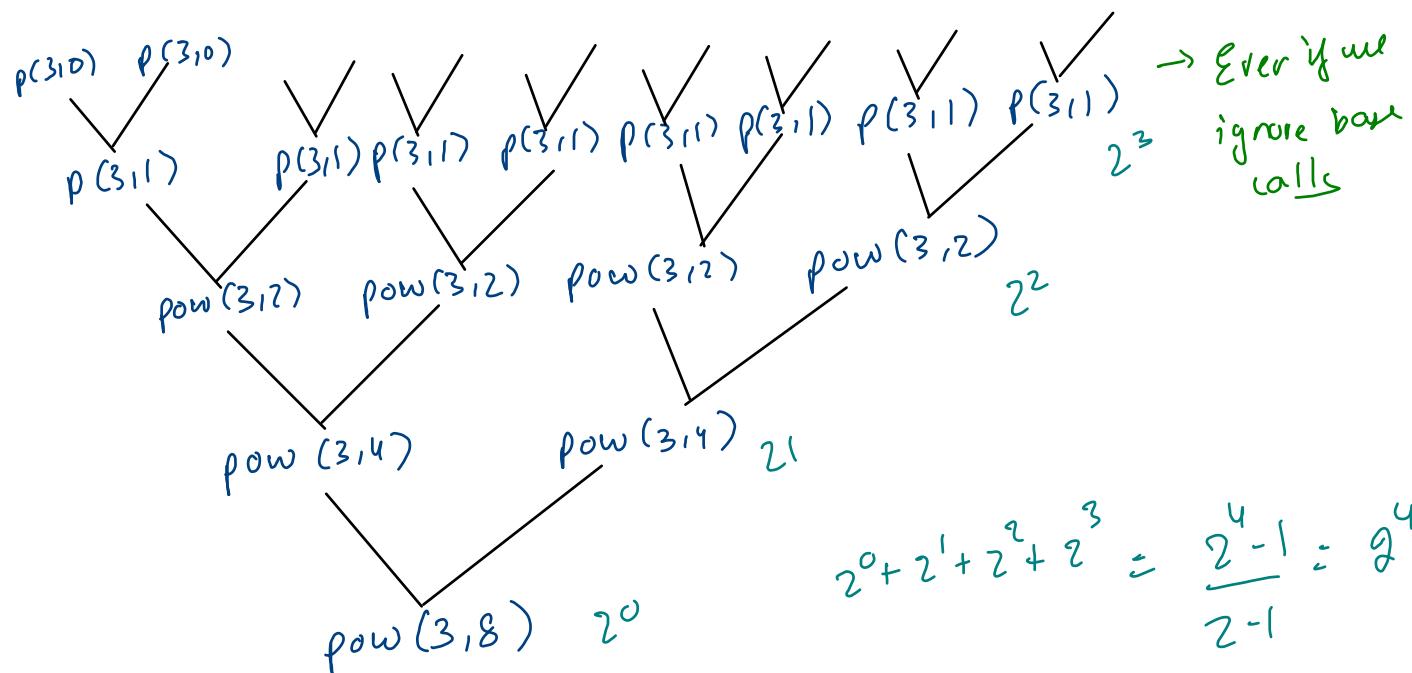
if we do

$$\text{int } xpn = \text{power}(x, n)_2 * \text{power}(x, m)_2$$

↑                   ↑  
2 calls

Ques what will be the complexity?

Recursion Tree



$$2^0 + 2^1 + 2^2 + 2^3 = \frac{2^4 - 1}{2 - 1} = 2^4$$

Again  $O(n)$  {or even worse}

$\approx \Theta(n)$

→ Even if we  
ignore base  
calls

The screenshot shows a LeetCode problem titled "Print Zig Zag". The problem asks for a zigzag pattern of numbers from 1 to n. The solution provided is as follows:

```
class Solution {
public:
    vector<int> printZigZag(int n) {
        vector<int> ans;
        int i = 1;
        while(i <= n) {
            if(i % 2 == 1) {
                ans.push_back(i);
                i++;
            } else {
                ans.push_back(i);
                i--;
            }
        }
        return ans;
    }
};
```

The output for n=3 is shown as [3, 2, 1]. The output for n=4 is shown as [4, 3, 2, 1]. The output for n=5 is shown as [5, 4, 3, 2, 1]. The output for n=6 is shown as [6, 5, 4, 3, 2, 1]. The output for n=7 is shown as [7, 6, 5, 4, 3, 2, 1]. The output for n=8 is shown as [8, 7, 6, 5, 4, 3, 2, 1]. The output for n=9 is shown as [9, 8, 7, 6, 5, 4, 3, 2, 1]. The output for n=10 is shown as [10, 9, 8, 7, 6, 5, 4, 3, 2, 1].

### ③ Meet expectation with faith

+ 2 times print is used.

```
public static void pzz(int n){  
    if(n == 0) return;  
  
    System.out.print(n + " ");  
    pzz(n-1);  
    System.out.print(n + " ");  
    pzz(n-1);  
    System.out.print(n + " ");  
}
```

P22(1) P22(2) P22(3)  
P22(4) P22(5)  
P22(6) P22(7)  
P22(8)

```

if(n == 0) return;
System.out.print(n + " ");
pzz(n-1);
System.out.print(n + " ");
pzz(n-1);
System.out.print(n + " ");
pzz(n-1);
    
```

$n \rightarrow \text{state no}$

```
public static void pzz(int n){  
    if(n == 0) return;  
    1 System.out.print(n + " ");  
    2 pzz(n-1);  
    3 System.out.print(n + " ");  
    4 pzz(n-1);  
    5 System.out.print(n + " ");
```

$n \rightarrow$  state no.

$3 \rightarrow 1$     $3 \rightarrow 2$     $2 \rightarrow 1$     $2 \rightarrow 2$     $1 \rightarrow 1$     $1 \rightarrow 2$     $0 \rightarrow *$     $1 \rightarrow 3$     $1 \rightarrow 4$    or    $1 \rightarrow 5$     $2 \rightarrow 3$     $1 \rightarrow 1$     $1 \rightarrow 2$    or    $1 \rightarrow 3$     $1 \rightarrow 4$    or    $1 \rightarrow 5$     $2 \rightarrow 5$     $3 \rightarrow 3$   
 $\downarrow$              $\downarrow$   
 $3$              $2$              $1$              $1$              $1$              $2$              $1$              $1$              $1$              $2$              $1$              $1$              $1$              $2$              $3$

I'll work for 2nd half of the tree.

## Recursion (Time Complexity formula)

$$(\text{calls})^{\text{height}} + \{\text{preorder} + \text{inorder} + \text{postorder}\} * \text{height}$$

↑  
from any node

for 122

$$(2)^n + \{k+k+k\} * n$$

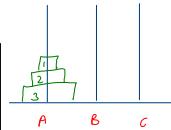
$$= 2^n + 3kn$$

$O(2^n)$

## Tower of Hanoi

1. There are 3 towers. Tower 1 has  $n$  disks, where  $n$  is a positive number. Tower 2 and 3 are empty.  
 2. The disks are increasingly placed in terms of size such that the smallest disk is on top and largest disk is at bottom.  
 3. You are required to:  
 3.1 Print the steps to move the disks.  
 3.2 Move Tower 1 to Tower 2 using Tower 3  
 3.3. following the rules:  
 3.3.1 move 1 disk at a time.  
 3.3.2 never place a smaller disk under a larger disk.  
 3.3.3 you can only move a disk at the top.

Note - The online judge can't force you to write this function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

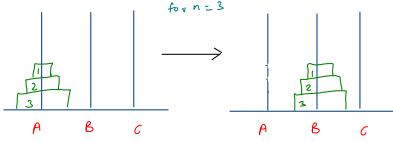


- constraints:**  
 ↳ only one disk can be removed at a time.  
 ↳ larger disk can not be placed on top of smaller disk.

① Expectation:

$\text{toh}(n, \text{A}, \text{B}, \text{C})$

for  $n=3$



→

Final state: Tower B has 3 disks (3, 2, 1).

② Faith:

$\text{toh}(n-1, \text{A}, \text{B}, \text{C})$



→

Final state: Tower B has 2 disks (2, 1).

③ Mutating Expectation from Faith

$\text{toh}(\text{C}, \text{n}-1, \text{A}, \text{B})$   
 src dest aux



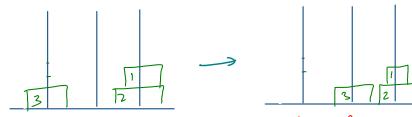
→

Final state: Tower C has 2 disks (2, 1).

$n^{\text{th}} [\text{A} \rightarrow \text{B}]$

```
public static void toh(int n, int t1id, int t2id, int t3id){
    if(n == 1) {
        System.out.println(n + "[" + t1id + " -> " + t2id + "]");
        return;
    }

    toh(n-1, t1id, t3id, t2id);
    System.out.println(n + "[" + t1id + " -> " + t2id + "]");
    toh(n-1, t3id, t2id, t1id);
}
```



→

Final state: Tower B has 3 disks (3, 2, 1).

$\text{toh}(\text{n}-1, \text{C}, \text{B}, \text{A})$   
 src dest aux



→

Final state: Tower A has 3 disks (3, 2, 1).

## Tower of Hanoi Dry Run

```
public static void toh(int n, int t1id, int t2id, int t3id){
    if(n == 0) {
        return;
    }
    toh(n-1,t1id,t3id,t2id);
    System.out.println(n + " " + t1id + " -> " + t2id + "]");
    toh(n-1,t3id,t2id,t1id);
}
```

0		
0 B A C	A	
1 B C A	A	
0 A C B	A B	
0 A C B	A B	
1 A B C	A C A B	
2 A C B	A C B A	
3 A B C	A B C	

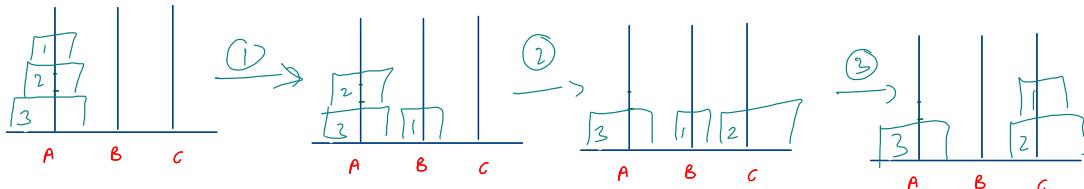
1 → 2 3

1 → 2 3

1 → 2 3

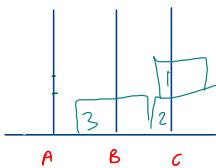
first faith is complete

- ① 1 from A to B
- ② 2 from A to C
- ③ 1 from B to C
- ④ 3 from A to B
- ⑤ 1 from C to A
- ⑥ 2 from C to B
- ⑦ 1 from A to B



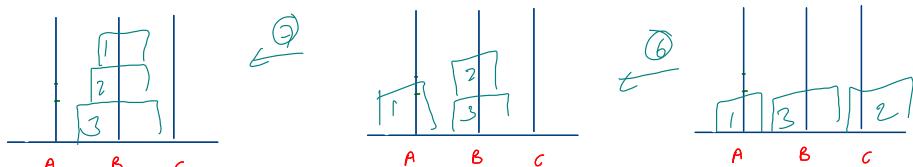
↓ ④

→ own work



↓ ⑤

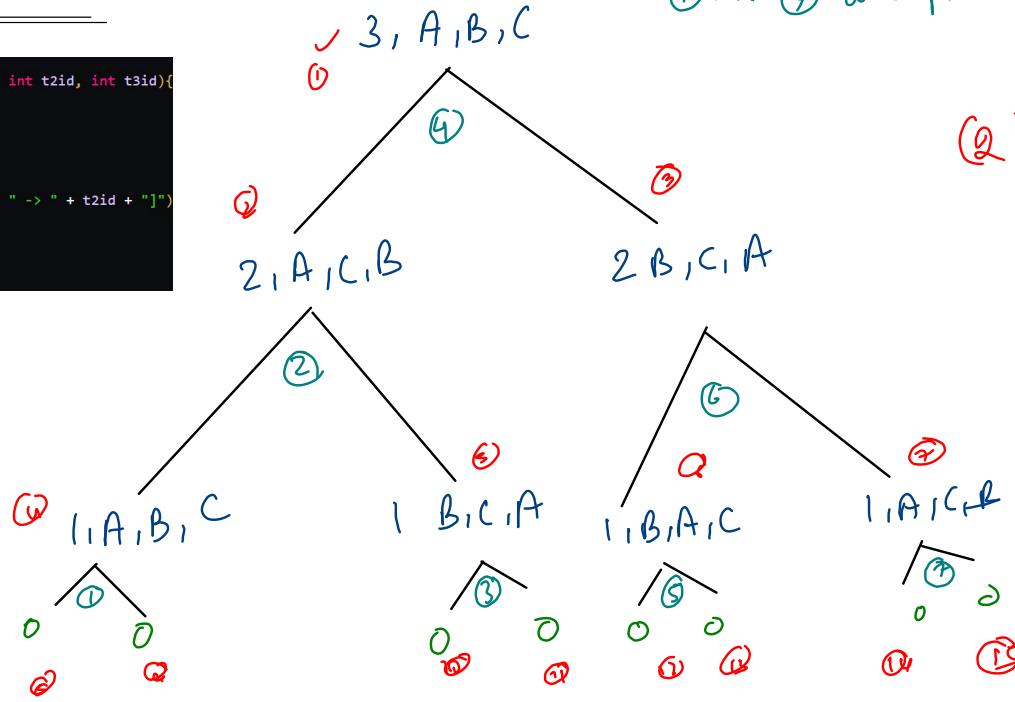
second faith



## Tower of Hanoi Recursion Tree

```
public static void toh(int n, int t1id, int t2id, int t3id){
    if(n == 0) {
        return;
    }

    toh(n-1,t3id,t2id,t1id);
    System.out.println(n + "[" + t1id + " -> " + t2id + "]");
    toh(n-1,t2id,t1id,t3id);
}
```



①... ⑦ are print statements

$$(2)^{n+1} + (O+k+O)n$$

$$2^{n+1} + kn$$

$$\approx O(2^n)$$