

DAY
42
DSA

Introduction to Recursion

16 Oct, 2021

→ Recursion is a wonderful programming tool.
It provides a simple, powerful way of approaching a variety of problems.

* What is Revision?

→ A function is said to be recursive, if it calls itself. जो सूची ही की call करते हैं।

Before jumping into the depth of Recursion,
lets try to see some connection between
Math & Programming

Math a) rewriting / defining

$$f(x) = (3x)^2$$

Math में अपर कोड
function द्वारा लिखा
रखा है हमें भी अपको
ng/Java में कैसे
लिखेंगे

b) Use of function

$$\rightarrow f(2) = (2)^2 - 4$$

Java. a) Writing / defining
a function

public static void Fun(int x)

9

Use of function in JAVA

```
→ int i = Fun(2);  
      = 4
```

(i) Programming language में Algebra को represent करने वाला tool जी होता है, उसे वोल्टी है।

Maths में इस और concept होता था Principle of Mathematical Induction

PMI

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

→ 1) formula prove करके दिखाओ।

Step 1) Let's assume that formula is true for $n=k$.

Step 2) Prove that formula holds true for $n=k+1$

Step 3) Prove that formula holds true for $n=1$.

→ (i) $n=k$ के लिए true है, तो assume करके $n=k+1$ prove करेंगे, यहाँ $n=1$ के लिए prove करेंगे।

→ तो eventually all formula for natural numbers के लिए hold करता था।

↪ क्योंकि जब $n=1$ के लिए करेंगे prove, तो 1 को use करके 2 के लिए prove करता, 2 को use करके 3 के लिए prove करता।

! ऐसे करते करते

$k-1$ को use करके k के लिए prove करता

k को use करके $k+1$ के लिए prove करता।

तो सभी natural no. के लिए true hold करता।

Eg. PMT

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

इस formula की true prove करके प्रक्रमों।

Step 1)

Assume the formula is true for $n=k$
अब अगर हमने इसी true माना दिया,
we can say that

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

यह सच है

हमने क्या
assume किया

assume किया है

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

यह हमने सच माना है.

Step 2)

To prove for $n = k+1$,

अब हमें यह prove करना है कि यह formula

सच है $n = k+1$ के लिए

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

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

परें अब हमें
prove करना है

मतलब

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

$$1+2+3+\dots-k+(k+1) = \frac{(k+1)(k+2)}{2}$$

ये हमें prove करना है।

$$(1+2+3+\dots-k) = \frac{k(k+1)}{2}$$

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

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

LHS

RHS

Page No.	
Date	

$$\begin{aligned} & \sum_{i=1}^{i=k} i + (k+1) \\ & = \frac{k(k+1)}{2} + k+1 \end{aligned}$$

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

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

$$\begin{aligned} & = k(k+1) + 2(k+1) \\ & = \frac{(k+1)(k+2)}{2} = \frac{(k+1)(k+2)}{2} \end{aligned}$$

∴ LHS = RHS

Hence Proved that $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$

By using formula from $n=k$ stage, we proceed formula for $n=k+1$ stage.

Step 3) Prove that the formula holds true for $n=1$.

LHS

RHS.

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

$$1 = \frac{2}{2}$$

$$1 = 1$$

∴ LHS = RHS

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

- ∴ Step 1) To Assume that the formula is true for
 $n=k$. &
 ∴ Step 2) then using Step 1) Assumption
 Prove that the formula is true for $n=k+1$.
 ∴ Step 3) Prove that the formula is true for $n=1$.

Hence, all the above steps are done

Hence, we can say that formula holds true for all natural numbers.

→ Here n is all the natural numbers

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

1. के लिए True है \Rightarrow
 2. के लिए प्रमाण करेंगे (K+1)
 3. & so on \Rightarrow तो $K+1$
 के लिए भी जब

We just solved a PMI problem, It is important to note that we will use similar steps to approach a Recursion problem.

We will discuss these steps through one problem

↳ "Print Decreasing"

So, Let's Solve

Recursive functions PMI based एवं

$$f(x) = x^2 - x*x \rightarrow \text{Square}$$

$$f(x = x!) \rightarrow x \cdot f(x-1) \rightarrow \text{factorial}$$

जटिल मैट्रिक्स

कठिन है Recursion

संख्या की Items

जो एक express Statement,

जोकि है Recursion

PRINT DECREASING

→ If we pass input as 5, then we need to print 5 4 3 & 1 in different lines.

→ To print decreasing function होगी, इस

उसे एक n Variable Pass करेगी इस function को. काम होगा कि हम n से 1 तक की counting return कर / या print करके दे।

Recursion और PMI का thought process same है

तो यहाँ पर भी PMI का logic मिलता है

इस 2 level पर सोचना होगा है

High Level

Low Level



इसे assume कर लें तो X की सभी

पाली अपना काम कर देगा

ये क्यद्य

X की एक काम दिया है,

जो मैं अपना काम कर देगा

जो मैं अपना काम कर देगा

PMI की Faith/Trust से Expectation meet होती है।

PMI



Faith

Expectation

Step 1) Assume
that function
is true for $n=k$.

Step 2) Prove that

function is
true for $n=k+1$.

Recursion



Faith

Expectation

Recursion की faith से Expectation meet होती है।

अब तक हमने क्या किया पाना है :-

- 1.) Math में की functions होती हैं, Java में की functions होती हैं।
- 2.) PMI की faith से expectation meet होती है। *Workshop / Magic of Abstraction*

3.) $f(x)$ → Java function *Programming Languages की Algebra की represent करती tool का Function*
(यहाँ Math में function होती है, ऐसे ही Java में की function है।)

- 4.) Recursion की faith से expectation meet होती है।
 faith → Expectation

Point Decreasing

$$Pd(n) = n, n-1, n-2, n-3 \longrightarrow 3, 2, 1$$

$P(d) = 5 \ 4 \ 3 \ 2 \ 1$ इसकी होमर्ट
में 2 type की Thinking

1. High Level Thinking

2. Low Level Thinking

HIGH LEVEL THINKING

① Faith

Faith

Expectation

Meet Expectation via Faith.

Dry Run & Draw

Stack Frame

Assume कि $Pd(n-1)$ में Pd(n-2)

↪ Have faith that this function already works.

$$\longrightarrow Pd(4) = 4 \ 3 \ 2 \ 1$$

(मानला कि function अपना काम करेगा और इसे $Pd(4)$ की value देगा।)

→ Recursion में सबसे मुश्किल है to do the

Suspension of Disbelief

मतलब ये मानना बहोत मुश्किल है कि $Pd(4)$ की value हमारे पास

आज/दूरी /ऐसा मानना मुश्किल जगेगा

कि अभी बिना किसी किसी func पर जाएगा और हमें $Pd(4)$ का

महालव थे Abstraction का Magic समझना ज़रूरी है।

→ हमारा काम हो जाएगा,
कैसे होगा → इतना पाठ्य जानने की
परवर्त नहीं है हमें
eg - break car के लगाजाएँ, कैसे
होंगे? → क्या करना है जानकारी?
ये पता जरूर दोनों चाहिए कि कौनसी
वात की detail जानना ज़रूरी है,
और कौनसी वात की detail जानना
ज़रूरी नहीं है।

⑥ Expectation

→ Set Expectation

उमरी Expectation Pd से थे हैं कि 3DR input
5 दिया तो वो हमें 5 4 3 2 1 देगा।

$$Pd(5) = 5 4 3 2 1$$

→ Expectation is same as question

⑦ Meet Expectation via Fault

→ अपने सपनों को विश्वास
के साथ पुरा करो।

Expectation की Fault की terms में express करो

Pd(5) की Pd(4) की terms में express करो।
→ 5 हम सुनेंगे और Pd(4) अपने आप

first print n
then call
pd(n-1)

$$Pd(5) = 5 \underbrace{Pd(4)}_{\text{करवा जाएगा 4,3,2,1}}$$

→ अगले बार यह करवा देंगे।

HIGH LEVEL THINKING

Page No.

Date

Only for $n=5$

// Expectation $\Rightarrow pd(5) = 54321$

// Faith $\Rightarrow pd(4) = 4321$

// Meet Expectation via faith $\Rightarrow pd(5) = 5pd(4)$

Generic (for all n)

// Expectation $\Rightarrow pd(n) = (n)(n-1)(n-2)\dots(3)(2)(1)$

// Faith $\Rightarrow pd(n-1) = (n-1)(n-2)\dots(3)(2)$

// Meet Expectation via faith \Rightarrow

$\rightarrow (n) pd(n-1)$

This is the basic code without any base case

public static void printDecreasing (int n)

{ System.out.println (n);

printDecreasing (n-1);

}

$pd(5) \rightarrow pd(4)$
call

$pd(5)$ at main

\rightarrow $pd(4)$ memory

$pd(4)$, $pd(3)$ \rightarrow $pd(2)$

$pd(3)$ &

soon

pd $n=0$ \rightarrow pd $n=1$

pd $n=1$

pd $n=2$

pd $n=3$

pd $n=4$

pd $n=5$

main

first line

at $n=0$

print

at $n=1$

print

at $n=2$

print

at $n=3$

print

at $n=4$

print

at $n=5$

Red tick at $n=0$

first line

at $n=1$

print

at $n=2$

print

at $n=3$

print

at $n=4$

print

at $n=5$

print

green tick at $n=0$

green tick at $n=1$

green tick at $n=2$

green tick at $n=3$

green tick at $n=4$

green tick at $n=5$

green tick at $n=6$

green tick at $n=7$

green tick at $n=8$

green tick at $n=9$

green tick at $n=10$

green tick at $n=11$

green tick at $n=12$

green tick at $n=13$

green tick at $n=14$

green tick at $n=15$

green tick at $n=16$

green tick at $n=17$

green tick at $n=18$

green tick at $n=19$

green tick at $n=20$

green tick at $n=21$

green tick at $n=22$

green tick at $n=23$

green tick at $n=24$

green tick at $n=25$

green tick at $n=26$

green tick at $n=27$

green tick at $n=28$

green tick at $n=29$

green tick at $n=30$

green tick at $n=31$

green tick at $n=32$

green tick at $n=33$

green tick at $n=34$

green tick at $n=35$

green tick at $n=36$

green tick at $n=37$

green tick at $n=38$

green tick at $n=39$

green tick at $n=40$

green tick at $n=41$

green tick at $n=42$

green tick at $n=43$

green tick at $n=44$

green tick at $n=45$

green tick at $n=46$

green tick at $n=47$

green tick at $n=48$

green tick at $n=49$

green tick at $n=50$

green tick at $n=51$

green tick at $n=52$

green tick at $n=53$

green tick at $n=54$

green tick at $n=55$

green tick at $n=56$

green tick at $n=57$

green tick at $n=58$

green tick at $n=59$

green tick at $n=60$

green tick at $n=61$

green tick at $n=62$

green tick at $n=63$

green tick at $n=64$

green tick at $n=65$

green tick at $n=66$

green tick at $n=67$

green tick at $n=68$

green tick at $n=69$

green tick at $n=70$

green tick at $n=71$

green tick at $n=72$

green tick at $n=73$

green tick at $n=74$

green tick at $n=75$

green tick at $n=76$

green tick at $n=77$

green tick at $n=78$

green tick at $n=79$

green tick at $n=80$

green tick at $n=81$

green tick at $n=82$

green tick at $n=83$

green tick at $n=84$

green tick at $n=85$

green tick at $n=86$

green tick at $n=87$

green tick at $n=88$

green tick at $n=89$

green tick at $n=90$

green tick at $n=91$

green tick at $n=92$

green tick at $n=93$

green tick at $n=94$

green tick at $n=95$

green tick at $n=96$

green tick at $n=97$

green tick at $n=98$

green tick at $n=99$

green tick at $n=100$

green tick at $n=101$

green tick at $n=102$

green tick at $n=103$

green tick at $n=104$

green tick at $n=105$

green tick at $n=106$

green tick at $n=107$

green tick at $n=108$

green tick at $n=109$

green tick at $n=110$

green tick at $n=111$

green tick at $n=112$

green tick at $n=113$

green tick at $n=114$

green tick at $n=115$

green tick at $n=116$

green tick at $n=117$

green tick at $n=118$

green tick at $n=119$

green tick at $n=120$

green tick at $n=121$

green tick at $n=122$

green tick at $n=123$

green tick at $n=124$

green tick at $n=125$

green tick at $n=126$

green tick at $n=127$

green tick at $n=128$

green tick at $n=129$

green tick at $n=130$

green tick at $n=131$

green tick at $n=132$

green tick at $n=133$

green tick at $n=134$

green tick at $n=135$

green tick at $n=136$

green tick at $n=137$

green tick at $n=138$

green tick at $n=139$

green tick at $n=140$

green tick at $n=141$

green tick at $n=142$

green tick at $n=143$

green tick at $n=144$

green tick at $n=145$

green tick at $n=146$

green tick at $n=147$

green tick at $n=148$

green tick at $n=149$

green tick at $n=150$

green tick at $n=151$

green tick at $n=152$

green tick at $n=153$

green tick at $n=154$

green tick at $n=155$

green tick at $n=156$

green tick at $n=157$

green tick at $n=158$

green tick at $n=159$

green tick at $n=160$

green tick at $n=161$

green tick at $n=162$

green tick at $n=163$

green tick at $n=164$

green tick at $n=165$

green tick at $n=166$

green tick at $n=167$

green tick at $n=168$

green tick at $n=169$

green tick at $n=170$

green tick at $n=171$

green tick at $n=172$

green tick at $n=173$

green tick at $n=174$

green tick at $n=175$

green tick at $n=176$

green tick at $n=177$

green tick at $n=178$

green tick at $n=179$

green tick at $n=180$

green tick at $n=181$

green tick at $n=182$

green tick at $n=183$

green tick at $n=184$

green tick at $n=185$

green tick at $n=186$

green tick at $n=187$

green tick at $n=188$

green tick at $n=189$

green tick at $n=190$

LOW LEVEL THINKING

Page No. _____
Date _____

Depth में समझना की क्या चल रहा है
Stack में

→ (i) Low Level Thinking का काम यह
base case figure out करो

(ii) base case पर इस ques में $n=0$ है
जोसे ही $n=0$ हो जाते outcome करो,
वही 0 point हो जाएगी 3/12
Pd(-1) को call करो

printDecreasing

```
→ import java.util.*;
→ import java.*;
→ public class Main {
    → public static void main (String [] args) {
        → Scanner scn = new Scanner (System.in);
        → int n = scn.nextInt ();
        → printDecreasing (n);
    }
    → //Expectation → pd(n) → n n-1 n-2 ... 3 2 1
    → //Faith → pd(n-1) → ... n-1 n-2 ... 3 2 1
    → //Meet Expectation via faith → n · pd(n-1)
    → public static void printDecreasing (int n) {
        → if (n == 0)
            → { return; }
```

→ Base Case

System.out.println (n);

printDecreasing (n);

}

3

Point Decreasing
(0) gets wiped
out of
stack. And
the control comes
back to printDecreasing()

HIGH LEVEL THINKING HAS 3 PARTS

1. Expectation

2. Faith

3. How to meet Expectation with Faith?

With the help of High Level Thinking,
we can figure out our basic code.

Then coming to LOW LEVEL THINKING

, we need to very run our code and
and try to look at the stack frame
after each recursive call. This helps us
to identify our BASE CASE

And, with both high & low level thinking
, we get our code finally.

Point Decreasing

↳ Time Complexity :- $O(n)$

As n calls are made and work is done corresponding to these n calls . Therefore the time complexity becomes $O(n)$

↳ Space Complexity :- $O(1)$

Since, there is no extra space is used , therefore , space complexity is constant .

However if Recursion call stack is taken into account , then the space complexity is $O(n)$ as there are n recursive calls

STACK FRAME OF PRINT DECREASING

```
public static void printDecreasing (int n)
{
    if (n == 0) { return; }
    System.out.println(n);
    printDecreasing (n-1);
}
```

Expectation :

```
Print Decreasing (5):
5
4
3
2
1
```

Faith :

```
Print Decreasing(4):
4
3
2
1
```

Expectation meeting
Faith:

```
Print Decreasing(5):
Print 5
Print Decreasing(4),
```

Initially, the first call is received from main, and all the statements of main before calling Print Decreasing (5) gets executed. When Print Decreasing (5) is called, it is pushed to the top of stack. Then the first statement of this function get executed. Condition $n == 0$ is checked, which is false. So, now control comes to the second statements, which prints "5". Then the third statement is excuted, which further calls Print Decreasing (4) which is now pushed on the top of the stack. When Print Decreasing (4) is called, it is pushed to the top of the stack. Then the first statement of this function is executed, the condition $n == 0$ is checked, which is false, so now control comes to the second statement, which print (4). This statement get excuted, which further calls Print Decreasing (3) which is now pushed on the top of the stack. Similar steps repeat when Print Decreasing (3), Print Decreasing (2) & Print Decreasing (1) are pushed onto stack one after other & printing "3", "2", "1". When the Print Decreasing (1) calls Print Decreasing (0), the condition check $n == 0$ become true and the return statement executes, returning the controls to PrintDecreasing (1). PrintDecreasing (0) gets repid out of the stack and the control comes back to PrintDecreasing (1).

```
Print Decreasing (0):
System.out.println(0)
Print Decreasing (-1);
```

```
Print Decreasing (1):
System.out.println(1);
Print Decreasing (0);
```

```
Print Decreasing (2):
System.out.println(2);
Print Decreasing (1);
```

```
Print Decreasing (3):
System.out.println(3);
Print Decreasing (2);
```

```
Print Decreasing (4):
System.out.println(4);
Print Decreasing (3);
```

```
Print Decreasing (5):
System.out.println(5);
Print Decreasing (4);
```

```
main {
    Print Decreasing(5)
}
```

PRINT INCREASING.

Input = 5,
Output → 1

2
3
4
5.

Approach :-

Recursion

High Level Thinking

Low Level Thinking

1. Expectation

2. Faith

3. Meeting
Expectation
with faith

Draw
& draw
the Stock
frame to
find the
base case

Expectation	Faith	Expectation meeting faith
Print Increasing(5);	Print Increasing(4)	Print Increasing(5)
2	1	1
3	2	2
4	3	3
5	4	4
		Print 5

Expectation → Our expectation from Print Increasing (n) is that if we pass $n=5$; it will give us the desired output (1 2 3 4 5)

Page No.	
Date	

- 2) Faith → Our faith from Print Increasing ($n-1$) is that it will print (1 2 3 4 9) for sure
- 3) To meet this expectation with faith; what can we do?

We can call PrintIncreasing (4), which will print (1, 2, 3, 4) & then our job will be to simply print 5.

Pseudocode

PrintIncreasing (5)

{
 PrintIncreasing (4)
 Print (5)}

Print Increasing (0){
 Print Increasing (1);
 System.out.println (0);

Print Increasing (1){
 Print Increasing (0);
 System.out.println (1);

Print Increasing (2){
 Print Increasing (1);
 System.out.println (2);

Print Increasing (3){
 Print Increasing (2);
 System.out.println (3);

Print Increasing (4){
 Print Increasing (3);
 System.out.println (4);

Print Increasing (5){
 Print Increasing (4);
 System.out.println (5);

main {
 Print Increasing (5);

public static void PrintIncreasing (int n)
{ if ($n == 0$)
 {return}; // Base Case
 PrintIncreasing ($n - 1$); // Faith
 System.out.println (n); // Self work

Initially the first call is received from main, and all the statements of main before calling Print Increasing (5) are executed. When Print Increasing (5) is called, it is pushed to top of stack. Then $n == 0$ condition is checked which is false, so now control goes to 2nd statement which further calls Print Increasing (4) which is now pushed to stack top. Similarly, this process continues till Print Increasing (0). Now, when condition $n == 0$ is checked, it is true. Now, return statement executes so now control comes back to Print Increasing (1), here the control comes to statement 3, which prints 1. Now control returns to Print Increasing (2), next statement here prints 2. Similar way control comes back to Print Increasing (3) & the next statements get executed, printing number 3. Now control returns to Print Increasing (4) & next statement get executed, printing the number 4. Finally control returns to Print Increasing (5), next statement get executed, printing the number 5. At last control returns to main.

```
→ import java.io.*;
→ import java.util.*;
→ public class Main {
    public static void main (String args) {
        Scanner scan = new Scanner (System.in);
        int n = scan.nextInt();
        printIncreasing (n);
    }
}

→ public static void printIncreasing (int n) {
    if (n == 0) // Base Case
        return;
    printIncreasing (n - 1); // Recursive Call
    System.out.println (n); // Self work.
}
```

Print Increasing

Time Complexity $\rightarrow O(n)$

Space Complexity $\rightarrow O(1)$

Same as of
Print Decreasing

PRINT INCREASING DECREASING

→ Actually Question is DECREASING, INCREASING, but name question
 This question is a combination of previous 2 questions i.e print Increasing & on website is wrong
 print Decreasing

We are required to print the count in decreasing order ($n, n-1, n-2 \dots 2, 1$) & then in increasing order ($1, 2, 3 \dots n$)

We are going to cover High Level Thinking & Low Level Thinking to understand this problem.

Steps of High Level Thinking (HLT):-

1. Set Expectation
2. Build Faith
3. Meet Expectation with Faith

Steps of Low Level Thinking (LLT):-

1. Dry run
2. Build Stack
3. Identify Base Case

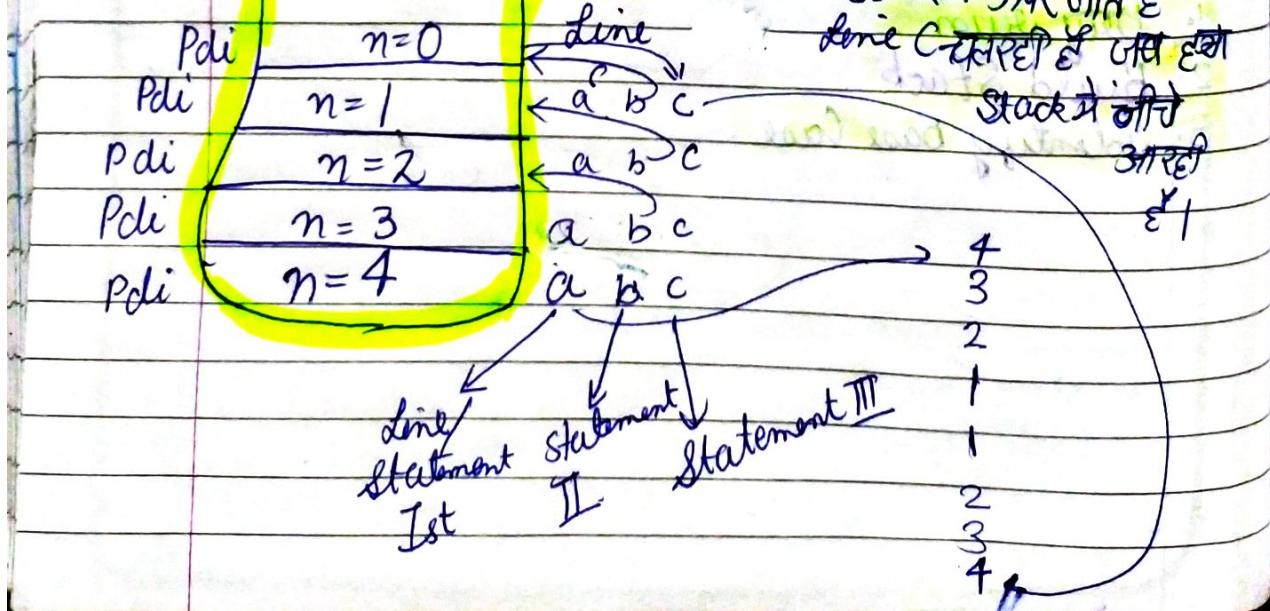
Let $n=4$

Expectation	Faith	Expectation meets Faith
$pdi(4) =$	$pdi(3) =$	$pdi(4) =$
4	3	4
3	2	$pdi(3) =$
2	1	4
1	1	4
1	2	$pdi(3) =$
2	3	4
3		
4		

Generalizing

Expectation	Faith	Expectation meets Faith
$pdi(n) =$	$pdi(n-1) =$	$pdi(n) =$
n	$n-1$	n
$n-1$	$n-2$	$pdi(n-1)$
$n-2$	1	n
1	$n-3$	
$n-1$	$n-2$	
n	$n-1$	

WTF recursion है deep
द्वारा दिये गए, उसे लोड करेंगे pre area.
WTF recursion से छाइर आते हैं, उसे लोड करेंगे Root Area



→ import java.io.*;
 → import java.util.*;

→ public class Main {

{ public static void main (String args)
 { Scanner scan = new Scanner (System.in);
 int n = scan.nextInt();
 pdi (n);

}

public static void pdi (int n)

{ if (n == 0)

→ Base case

{ return;

}

System.out.println (n); // pre area
 pdi (n-1);

System.out.println (n); // post area

Y Y

Print Increasing Decreasing

↳ Time Complexity

↳ The time complexity is linear
 because it makes one recursive call
 at a time.

Space Complexity

↳ No extra space required.

∴ Space Complexity is constant.

If we include the space used in Recursive stack, then
 the space Complexity is $O(n)$

Jamuna Verma