DAY 42 DSA

16 Oct, 2021

Introduction to Recursion.....

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

* What is Recursion ?

↳ A function is said to be recursive, if it calls itself. जैसे function को recursive बोलते हैं जो खुद ही को call करते हैं।

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

Math a) writing / defining a func in Math

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

↓ Math में अगर कोई function ऐसे लिखा गया है तो उसको Programming/Java में कैसे लिखेंगे

b) Use of function

$f(x) = x^2$

→ $f(2) = (2)^2 = 4$

Java a) writing / defining a func in Java

```
public static void Fun(int x)
{
    return (x * x);
}
```

Use of function in JAVA

→ int i = Fun(2);
= 4

Page No.
Date

तो Programming language में Algebra को represent करने वाला tool जो होता है, उसे **FUNCTION** बोलते हैं।

Maths में एक और concept होता था Principle of Mathematical Induction

PMI $\boxed{\sum_{i=1}^{n} i = \frac{n(n+1)}{2}}$ → ये 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 $n=1$.

→ तो $n=K$ के लिए true है, ये assume करके $n=K+1$ prove करदेंगे, और $n=1$ के लिए prove करदिया

→ तो eventually ये formula सारे natural numbers के लिए hold करता था।

↳ क्योंकि जब $n=1$ के लिए करदिया prove तो
1 को use करके 2 के लिए prove करलो
2 को use करके 3 के लिए prove करलो
! ऐसे करते करते
$K-1$ को use करके K के लिए prove करलो
K को use करके $K+1$ के लिए prove करलो
तो सारे natural no. के लिए true hold करजाएगा।

Eg.1 PMI

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

इस formula को true prove करके दिखाओ।

Step 1) Assume the formula is true for n=k

अब अगर हमने इसे true मान लिया,

we can say that

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

ये सच है

हमने ऐसा assume किया है.

मतलब

$$1+2+3+4+5 \text{ ---- } K = \frac{K(K+1)}{2}$$

ये हमने सच माना है.

Step 2) To prove for $n = k+1$

अब हमें ये prove करना है कि ये formula सच है $n = K+1$ के लिए

Page No.
Date

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

$$\sum_{i=1}^{K+1} i = \frac{(K+1)(K+2)}{2}$$ → ये अब हमें prove करना है

मतलब

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

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

ये हमें Prove करना है।

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

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

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

LHS RHS

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

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

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

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

∴ 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 proved formula for $n=k+1$ stage.

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

LHS RHS.

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

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

$$1 = 1$$

∴ LHS = RHS

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

Page No.
Date

- 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 तो
2 के लिए Prove करदेगे (K true है तो K+1 के लिए भी true होगा)
3 & so on
4
5

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, Lets Solve

Recursive functions PMI based होते है

$f(x) = x^2 = x*x$ → Square

$f(x) = x!$ → $x \cdot f(x-1)$ → factorial

बचपन मे हमने किए है Recursive func, जो func खुद की terms मे ही express हो जाए, उन्हे कहते है Recursive

Ques 1

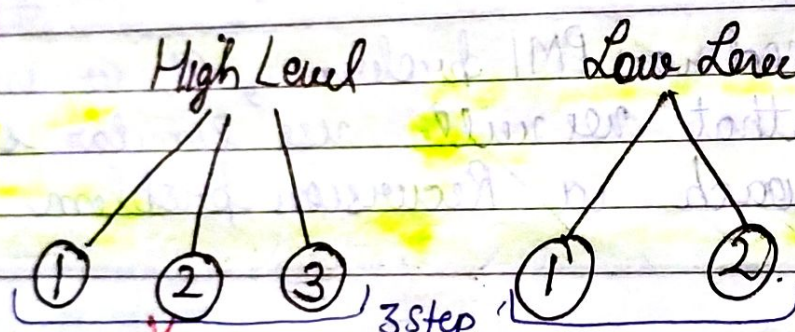
PRINT DECREASING

↳ If we pass input as 5, then we need to print (5 4 3 2 1) in different lines.

↳ जो print decreasing function होगा, हम उसे एक n variable pass करेंगे और इस function का काम होगा कि ये हमे n से 1 तक की counting return करे / या print करके दे।

Recursion और PMI का thought process same है तो यहाँ पर भी PMI वाला logic लगेगा।

हमें 2 level पर सोचना होता है:
- High Level
- Low Level



PMI और Recursion Abstraction पर चलता है।

↓

हम assume कर लेते हैं कि सामने वाला अपना काम कर के देगा।

X को एक काम दिया है, X अपना काम कर देगा तो मैं भी अपना काम कर दूँगा।

ये कुछ

PMI में Faith/Trust से Expectation meet करते हैं।

* **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 में faith से Expectation meet करती हैं।

अभी तक हमने ये बातें जानली हैं:-

1.) Math में भी functions होते थे, Java में भी function होते हैं।

2) PMI faith से expectation meet करती है। Workon Magic of Abstraction
   ↳ faith → Expectation

3) Math f(x) → Java function
   जैसे Math में function होते थे, वैसे ही Java में भी function है। Programming Language में Algebra को represent करने वाला tool जैसे Function

4) Recursion में faith से expectation meet करती है।
   faith → Expectation

Print Decreasing

Pd(n) = n, n-1, n-2, n-3 ——— 3,2,1

P(d) = 5 4 3 2 1

हमें 2 type की Thinking रखनी होगी

1. High Level Thinking
2. Low Level Thinking → Dry Run & Debug, Stack Frame

Faith | Expectation | Meet Expectation via Faith

HIGH LEVEL THINKING

(a) Faith → Assume की Pd(n-1) हमें मिलजाएगा

↳ Have faith that this function already works.

→ Pd(4) = 4 3 2 1.

↳ मानलो ये function अपना काम करके देगा और हमें 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 आजाएगी। ऐसा मानना मुश्किल लगेगा कि अभी बिना कुछ किए ये func चलजाएगा और हमे Pd(4) देगा

मतलब ये Abstraction का Magic समझना जरूरी है।

↳ हमारा काम होजाएगा, कैसे होगा → इतना ज्यादा जानने की जरूरत नहीं है हमें

eg → break car के लगजाएगे, कैसे लगेंगे? → क्या करना है जानकर?

↳ ये पता जरूर होना चाहिए कि कौनसी बात की detail जानना जरूरी है, और कौनसी बात की detail जानना जरूरी नहीं है।

(b) Expectation → Set Expectation

हमारी Expectation Pd से ये है कि अगर input 5 दिया तो वो हमे 5 4 3 2 1 देगा।

Pd(5) = 5 4 3 2 1

→ Expectation is same as question

(c) Meet Expectation via Faith → अपने सपनो को विश्वास के साथ पुरा करो।

Expectation को Faith की terms में express करो

Pd(5) को Pd(4) की terms में express करो।

Pd(5) = 5 Pd(4)

→ हम रखेंगे और Pd(4) अपने आप 4 3 2 1 हो जाएगा क्योंकि Pd(4) को call ये भी कर हो जा सकता है। करदी।

first print n & then call pd(n-1)

HIGH LEVEL THINKING

Page No.
Date

Only For $n=5$

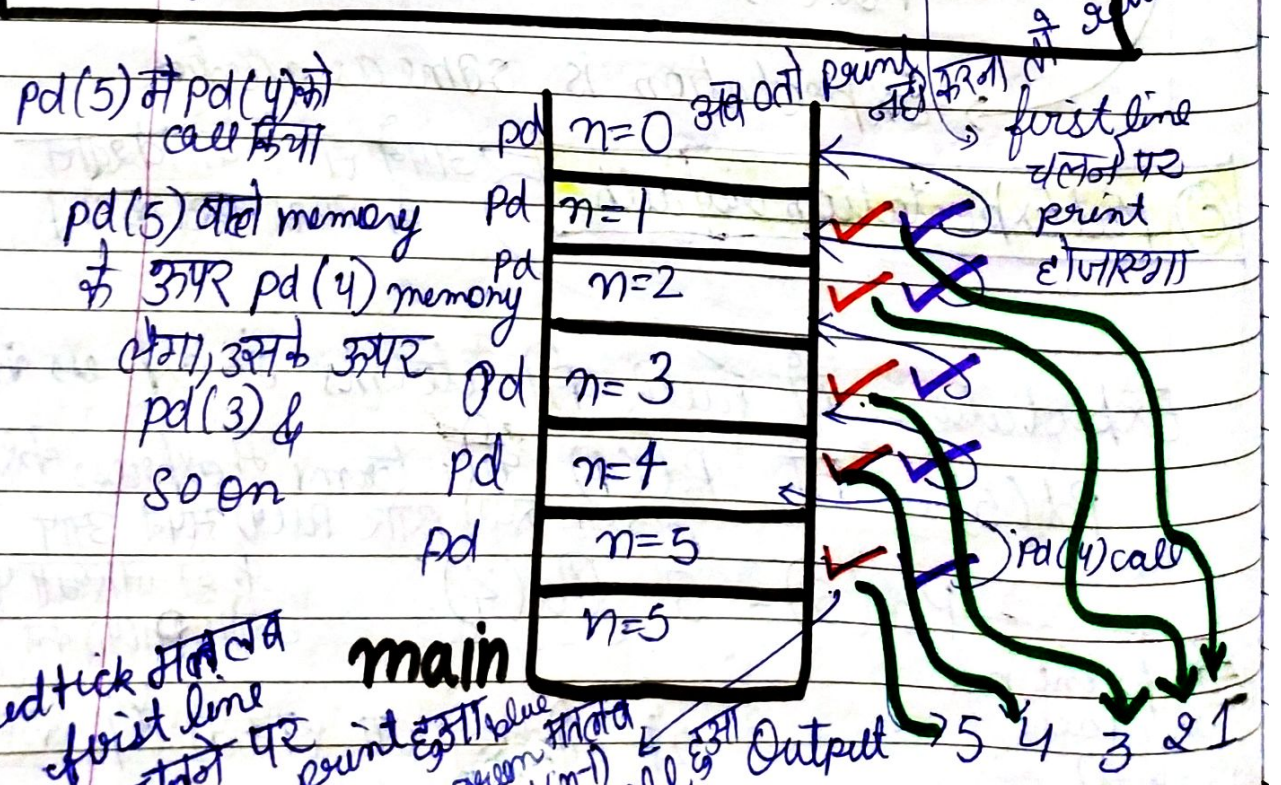
// Expectation ⇒ pd(5) = 5 4 3 2 1
// Faith ⇒ pd(4) = 4 3 2 1
// Meet Expectation via faith ⇒ pd(5) = 5 pd(4)

Generic (for all n)

// Expectation ⇒ pd(n) = (n)(n-2)(n-3) — 3 2 1
// Faith ⇒ pd(n-1) = (n-1)(n-2) — 3 2 1
// Meet Expectation via faith ⇒
→ (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);
}
```



LOWLEVEL THINKING

Page No.
Date

Depth में समझना की क्या चल रहा है Stack में

↳ तो Low Level Thinking का काम था base case figure out करना

↳ तो base case यहाँ इस ques में n=0 है जैसे ही n=0 हो जाए तो return करो, वर्ना 0 print होती रहेगी और Pd(n-1) को call लगती रहेगी।

Print Decreasing

```
import java.util.*;
import java.io.*;
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;
  }
  System.out.println(n);
  printDecreasing(n);
  }
}
```

→ Base Case

Print Decreasing (0) gets wiped out of stack. And the control comes back to printDecreasing(1)

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 dry 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.

```
Print to 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.

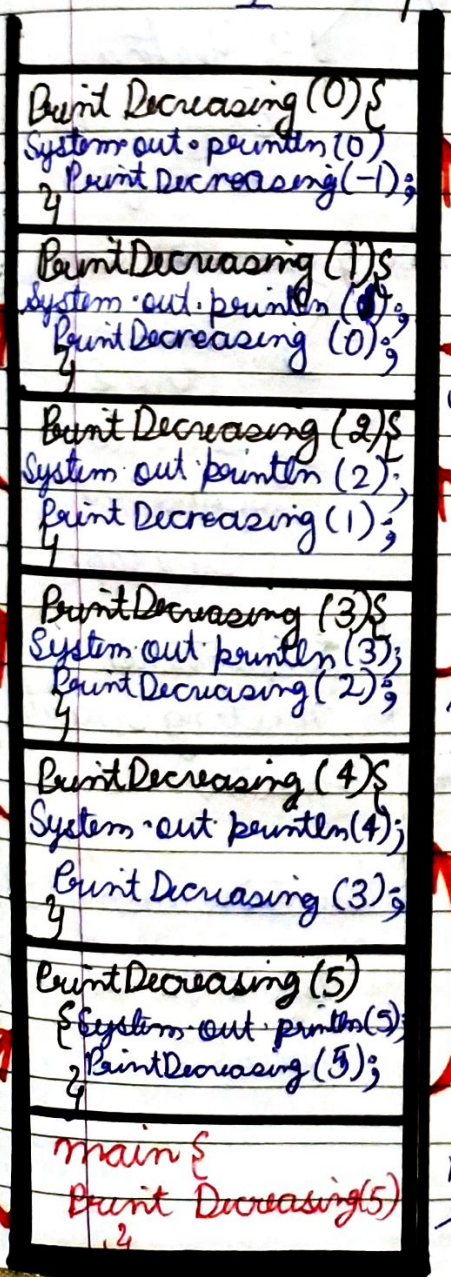
However if Recursion call stack is taken into account, then the space complexity is O(n) as there are n recursive calls

STACK FRAME OF PRINT DECREASING

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

→ Base Case.

Expectation:	Faith:	Expectation meeting Faith:
printDecreasing(5):	printDecreasing(4)	printDecreasing(5):
5	: 4	Print 5
4	3	printDecreasing(4);
3	2	
2	1.	
1		



Initially, the first call is received from main, and all the statements of main before calling printDecreasing(5) gets executed. When printDecreasing(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 executed, which further calls printDecreasing(4) which is now pushed on the top of the Stack.

When printDecreasing(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 controls comes to the second statement, which print(4). This statement get executed, which further calls printDecreasing(3) which is now pushed on the top of the stack. Similar steps repeat when printDecreasing(3), printDecreasing(2) & printDecreasing(1) are pushed into Stack one after other & printing "3", "2", "1". When the printDecreasing(1) calls printDecreasing(0) & the condition check n==0, become true and the return statement executes, returning the controls to printDecreasing(1). printDecreasing(0) gets wiped out of the Stack and the control comes back to printDecreasing(1).

Ques 2

PRINT INCREASING.

→ Input = 5,

Output → 1
2
3
4
5

Approach :-

Recursion
- High Level Thinking
  - 1. Expectation
  - 2. Faith
  - 3. Meeting Expectation with faith
- Low Level Thinking → Diagram & draw the stack frame to find the base case

Expectation	Faith	Expectation meeting faith
Print Increasing(5): 1 2 3 4 5	Print Increasing(4) : 1 2 3 4	Print Increasing (5) : Print Increasing(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) for sure

3) To meet this expectation with faith; what can we do?

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

Pseudo code

```
Print Increasing (5)
{ Print Increasing (4)
  Print (5)
}
```

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

```
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);
}
```

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 execute. So now control come back to Print Increasing (1), here the control come to statement 3, which print 1. Now control return 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 return to Print Increasing (5), next statement get executed, printing the number 5. At last control return 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();
        printDecreasing (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 → O(n)
Space Complexity → O(1)

Same as of Print Decreasing.

Recs -3

PRINT INCREASING DECREASING

Page No.
Date

↳ Actually Question is DECREASING INCREASING, but name given on website is wrong

This question is a combination of previous 2 questions i.e print Increasing & print Decreasing

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

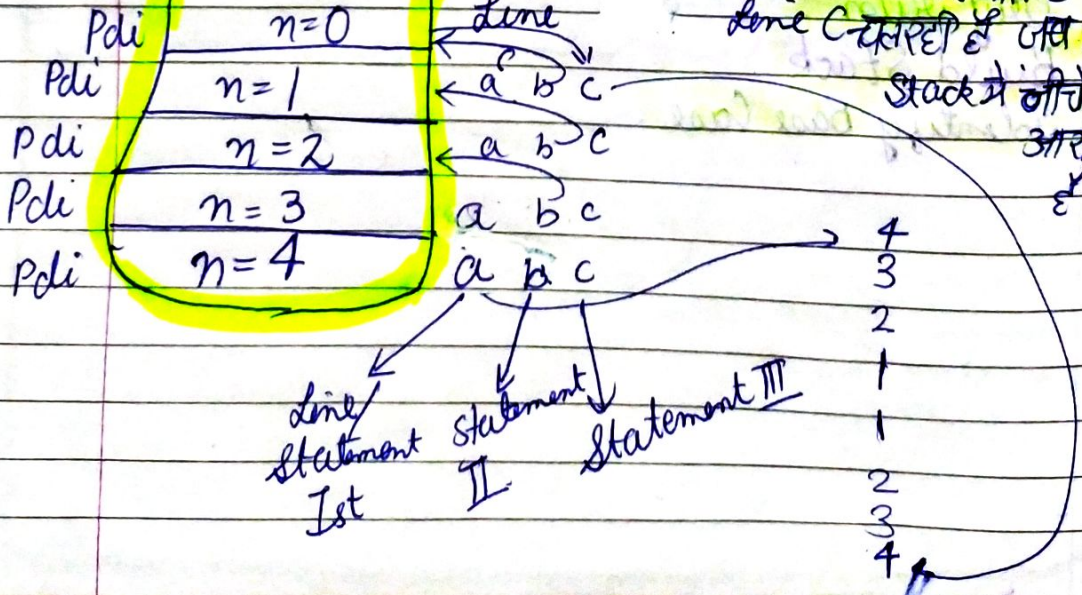
Let n=4

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

Generalizing

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

जब हम Recursion में deep dive करते हैं, उसे बोलते हैं pre area. जब recursion से बाहर आते हैं, उसे बोलते हैं Post Area



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

Page No.
Date

→ public class Main {

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

Print Increasing Decreasing

↳ Time Complexity
↳ The time complexity is linear because recursion call is made.

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)

Tamanna Verma