DAY 43 DSA

17 Oct-2021

To understand version, one must first understand Recursion
—Stephen Hawking

# Recursion cont......

## Agenda for Today

1. Factorial ⟶ CW
2. Power Linear ⟶ CW
3. Power Logarithmic ⟶ CW.
4. Print Zigzag ⟶ pending
5. Tower Of Hano ⟶ pending.

Ques. 1.

# FACTORIAL

A number n will be given to us. We need to find the factorial of n.

The factorial of a number is the product of all the integers from 1 to that number

For eg → factorial of n = 4, is 4*3*2*1

Very Imp. Point

| |
|---|
| You must know that the Factorial of both 0 and 1 is 1. |

$$\boxed{\begin{aligned} &\lfloor 0 = 1 \\ &\lfloor 1 = 1 \end{aligned}}$$

| |
|---|
| Factorial of negative integers do not exist. |

High Level Thinking → Expectation → Faith → Expectation meets Faith

| Expectation | Faith | Expectation meet faith |
|---|---|---|
| $f(5) = 5*4*3*2*1$ | $f(4) = 4*3*2*1$ | $f(5) = 5*f(4)$ |

1. Self Expectation
we expect that अगर input 5 है so $n = 5$
तो output होगा → $f(5) = 5 \times 4 \times 3 \times 2 \times 1$

2. Build Faith :- हमें ये faith रखना चाहिए कि अगर हमारा code $n = 5$ के लिए output देसकता है तो $n = 4$ के लिए भी Output जरूर देगा।
You just need to believe this.
Don't focus on how this will happen.
बस मानलो की factorial for $n = 4$ मिलजाएगा
$f(4) = 4 \times 3 \times 2 \times 1$.

3. Expectation meets Faith :-
For printing the desired output for $n=5$.
we could just print "5" before the output for $n=4$.

$$f(5) = 5 \times f(4)$$

We are done with High Level Thinking

→ Now, we need to do the $\boxed{\text{Low Level Thinking}}$ to find the base case.
↓
Dry run & Draw Stack

## Why is a base case Important?

If we don't have a base condition that check if the recursion function should be called or not then the program wouldn't know when to stop.

अगर कोई base condition नही होगी तो program कभी रुकेगा ही नही तो program infinitely चलता रहेगा

हमे ये नही चाहिए
तो हमे एक base condition या base case जरूर रखनी होगी ताकि हमारे program को ये पता हो कि जब ऐसी कोई condition सच होगी या जब मेरे पास ऐसा कोई case होगा तो मुझे रुक जाना है।

तो अब हम Low Level Thinking करते हैं

तो हमे पता चलगया हमारा Base Case 0

Low Level Thinking

using the Stacking process

Using the Stacking Process ↓

मतलब abc तीनो line चली

Dry run & draw stack

| | n | fnm1 | fn | |
|---|---|---|---|---|
| f | 0 | | | |
| f | 1 | 1 | 1 | abc |
| f | 2 | 1 | 2 | abc |
| f | 3 | 2 | 6 | abc |
| f | 4 | 6 | 24 | abc |
| f | 5 | 24 | 120 | abc |

```
if(n==0){return 1;}
line a
int fnm1=factorial(n-1);
line b
int fn = n*fnm1;
line c
return fn;
```

जैसे ही line a, bc तीनो line चली func की तो ये function जितनी भी memory cover करते थे, वो सब wipe out होजाएगी bcz now these variable are no longer of use तो Java Garbage Collector garbage collect करेगा।

अगर function void ना हो (मतलब अगर return type void ना हो (कुछ और हो return type like int/string/char) etc)।

तो हमेशा याद रखे कि recursive call लगाते हुए result को receive करना कभी न भूले।

$\underbrace{\text{int fn} =}_{\text{ऐसे receive}}$ factorial(n-1);

करना भी जरूरी वर्ना सही ans print नहीं होगा।

| Page No. | | | |
|---|---|---|---|
| Date | | | |

# FACTORIAL <code>

```
import java.util.*;
import java.io.*;
public class Main
{
public static void main(String[] args)
{ Scanner scan = new Scanner(System.in);
int n = scn.nextInt();
int fn = factorial(n);
System.out.println(fn);
}
//Expectation → f(5) → 5*4*3*2*1
// Faith → f(4) = 4*3*2*1.
// Expectation meet faith = 5* f(4).

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

fnm1 ~~मतलब~~ factorial of n minus 1

```
int fnm1 = factorial(n-1); // n-1xn-2 —— 3X2X1.
int fn = n * fnm1    // n X n-1 X —— 3 X 2X1.
return fn;
}
}
```

Ques. 2

# PowerLinear

↳ We are given a number x and a no. n, you are required to calculate x raised to the power n ie x multiplied n times

eg → $x = 5$
$n = 4$

$Power(x, n) = 5 \times 5 \times 5 \times 5 = (5)^4$

↓ Generalizing it

$$\boxed{power(x,n) = x * x * x — x (n \text{ times}) = x^n}$$

→ High Level Thinking

| **Expectation** | **Faith** | **Expectation meets Faith** |
|---|---|---|
| Power(x, n). $= x*x*x$ — $x*x$ (n times) $= x^n$ | Power(x, n-1). $= x*x*x*x$ —— $x*x$( -1 times) $= x^{n-1}$ | Power (x, n). $= x*(x^{n-1})$ $= x *$ Power(x, n-1); |

High Level Thinking → Expectation, Faith, Expectation meets Faith } Done

# Low Level Thinking

↓

It's done to find the base case.

$x=2$
$n=5$

| | x | n | xpnm1 | xpn | |
|---|---|---|---|---|---|
| P | 2 | 0 | | | |
| P | 2 | 1 | 1 | = 2 | a b c |
| P | 2 | 2 | 2 | = 4 | a b c |
| P | 2 | 3 | 4 | = 8 | a b c |
| P | 2 | 4 | 8 | = 16 | a b c |
| P | 2 | 5 | 16 | = 32 | a b c |

ये Linear approach है। इसमें recursive calls के Tower की height n+1 होती है।

n+1 recursive calls हो रही हैं ∴ The Time Complexity is O(n+1).

```
public static int power(int x, int n)
{ if (n==0)
    { return 1; }
Line a→ int xpnm1 = power(x, n-1);
Line b→ int xpn = xpnm1 * x;
Line c→ return xpn;
}
```

**Time Complexity**
↓
O(n) because we are recursively calling a subproblem with n-1 from n, hence n+1 recursive call will be made (+1

**Space Complexity**
↓
O(1) → No data structure used ∴ no extra space used.

जैसे ही line a, b, c तीनों line चली func की तो ये function के सभी variables ने जितनी भी memory cover कररहे थे, वो सब वापस space empty करदेंगे, वो सब function wipe out होजाएंगे memory से because now these variables are no longer of use तो Java Garbage Collector इन्हें as Garbage collect करेगा।

# Power Linear <code>

Page No.
Date

```
import java.util.*;
import java.io.*;
public class Main
{
public static void main(String[] args)
{
    Scanner scan = new Scanner(System.in);
    int x = scan.nextInt();
    int n = scan.nextInt();
    xpn मतलब x raise to the power n
    int xpn = power(x, n);
System.out.println(xpn);
}
                    // Eexpectation → p(2,5) = 2·2·2·2·2
                    // Faith → p(2,4) = 2·2·2·2
                    // Eexpectation meets faith → p(2,4)·2
public static int power(int x, int n)
{
  if (n==0)
    {
     return 1;
    }
                  x to the power n minus 1
int xpnm1 - power(x, n-1);
int xpn = xpnm1 * x;
            x to the Power n
return xpn;
}
}
```

Ques. 3

# Power Logarithmic

→ We are given a number $x$ and a number $n$, you are required to calculate $x$ raised to the power $n$.

→ हमें $x^n$ as output provide करना है for input $x, n$.

→ The Input & Output of Power Linear & Power Logarithmic is same.

→ Then what is the difference between power linear & power logarithmic

| Power Linear | Power Logarithmic |
|---|---|
| For even & odd positive powers | for even +ve powers |
| → $x^n = x \cdot x^{n-1}$ | → $x^n = x^{n/2} * x^{n/2}$ |
| | for odd +ve powers |
| | → $x^n = \underbrace{x}_{} * x^{n/2} * x^{n/2}$ |
| | एक x extra multiply होता है odd Integer के case में। |
| eg → $x = 2$, $n = 8$ | |
| $2^8 = 2 \cdot 2^7 \rightarrow 256$ | $2^8 = 2^4 \cdot 2^4 \rightarrow 256$ |
| $2^7 = 2 \cdot 2^6 \rightarrow 128$ | $2^4 = 2^2 \cdot 2^2 \rightarrow 16$ |
| $2^6 = 2 \cdot 2^5 \rightarrow 64$ | $2^2 = 2^1 \cdot 2^1 \rightarrow 4$ |
| $2^5 = 2 \cdot 2^4 \rightarrow 32$ | $2^1 = 2^0 \cdot 2^0 \cdot 2^1 \rightarrow 2$ |
| $2^4 = 2 \cdot 2^3 \rightarrow 16$ | $\log_2 8$ recursive calls |
| $2^3 = 2 \cdot 2^2 \rightarrow 8$ | |
| $2^2 = 2 \cdot 2^1 \rightarrow 4$ | |
| $2^1 = 2 \cdot 2^0 \rightarrow 2$ | |
| $2^0 = 1 \rightarrow 1$ | |
| $n+1$ recursive call | |

| | $x$ | $n$ | xpb2 | $x^n$ |
|---|---|---|---|---|
| Pow | 2 | 0 | x | bcz n is odd |
| Pow | 2 | 1 | $(1)^2 = 1$ | = 2 |
| Pow | 2 | 2 | $(2)^2 = 4$ | = 4. |
| Pow | 2 | 4 | $(4)^2 = 16$ | = 16 |
| Pow | 2 | 8 | $(16)^2 =$ | 256. |

## Power Linear → Time Complexity → $O(n)$

Page No.
Date

→ Since, you are recursively calling for a subproblem with $n-1$ from $n$, hence $n+1$ recursive calls will be made (+1 when it hits the base case $n=0$). You can also count the number of recursive calls in the call stack. So, the time complexity turns out to be $O(n)$ (which is independent of the value of $x$.

## Power Logarithmic → Time Complexity → $O(\log_2 n)$

→ Since, you are recursively calling for a subproblem with $n/2$ from $n$, hence at max $\log_2(n)$ recursive calls will be made. You can also count the number of recursive calls in the call stack. So, the time complexity turns out to be $\log_2(n)$ (which is independent of value of $x$

Space Complexity $= O(1)$ (bcz no extra data structure used.

### High Level Thinking of Power Logarithmic

| Expectation | Faith | Expectation meets faith |
|---|---|---|
| If $n$ is even<br>$x^n = x^{n/2} * x^{n/2}$<br>If $n$ is odd<br>$x^n = x * x^{n/2} * x^{n/2}$ | $P(x, \frac{n}{2}) = x * x * x$ - - - - - - $x * x$ - $\{\frac{n}{2}$ times$\}$<br>$= x^{n/2}$ | If $n$ is even<br>$P(x,n) = P(x, \frac{n}{2}) * P(x, \frac{n}{2})$<br>If $n$ is odd<br>$P(x,n) = x * P(x, \frac{n}{2}) * P(x, \frac{n}{2})$ |

So, we are done with High Level Thinking

# Low Level Thinking

```
public static int power
    (int x, int n)
{ if (n==0) {return 1;}
lineA  int xpb2 = power(x, n/2);
lineB  int xpn = xpb2 * xpb2;
lineC  if (n%2==1)
         {xpn = xpn * x;}
       }
lineD  return xpn;
}
```

Base Case

| | $x$ | $n$ | xpb2 | $x^n$ |
|---|---|---|---|---|
| pow | 2 | 0 | | |
| pow | 2 | 1 | because n is odd. $(1)^2 = 1$ | $= 2$ |
| pow | 2 | 3 | $(2)^2 = 4$ | $= 8$ |
| pow | 2 | 7 | $(8)^2 = 64$ because n is odd. | $= 128$ |

abcd

abcd

abcd

→ Height = 4

$\therefore \log_2 7$

Do the Square of xpb2
& if n is odd then multiply by x.

## Pseudocode for Power Logarithmic

1) If n is 0, then return 1.
2. else
   a.) Get the value of Power$\left(x, \frac{n}{2}\right)$ in a Variable named "xpnb2".
   b) If n is odd, then return x * xpnb2 * xpnb2
   c) Else (n is even), return xpnb2 * xpnb2

# PowerLogarithmic <code>

Page No.
Date

```
import java.util.*;
import java.io.*;
public class Main
{
  public static void main (String[] args)
  {
    Scanner scn = new Scanner (System.in);
    int x = scan.nextInt();
    int n = scan.nextInt();
    int xpn = power(x, n);
```

↳ x to the power n raised.

```
    System.out.println(xpn);
  }
```

If n is even | odd.

//Expectation → $P(x^n) = P(x^{n/2}) * P(x^{n/2})$ | $P(x^n) = * P(x^{n/2}) * P(x^{n/2})$

//Faith → $P(x^{n/2}) = x * x$ —— $\frac{n}{2}$) times $= x^{\frac{n}{2}}$

//Expectation meets faith = If n is even. $\begin{cases} power(x, n) = power(x, \frac{n}{2}) * power(x, \frac{n}{2}) \end{cases}$

If n is odd. $\begin{cases} power(x, n) = x * power(x, \frac{n}{2}) * power(x, \frac{n}{2}) \end{cases}$

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

→ x to the power by 2

```
  int xpb2 = power(x, n/2);

  int xpn = xpb2 * xpb2;
  if(n%2==1){ xpn = xpn*x; }
  return xpn;
}}
```

- Tamanna Verma