Page No.
Date

DSA

'lecture of 17th October'

Agenda :- (DSA Recursion)

→ Factorial : CW | done

→ Power linear : CW | done

→ Power Logarithmic : CW | done

→ Print Zig Zag → CW

→ Tower of Hanoi - CW

Page No. ____
Date ____

Q. Factorial

// use same high & low level strategy

$f(5) = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

HLT

1. Expectation: $f(5) = 5 \times 4 \times 3 \times 2 \times 1$
// ye हमको चाहिए because this what we want

2. Faith : $f(4) = 4 \times 3 \times 2 \times 1$
// we keep faith that this program will work for a value less than expectation so, in this case we take expectation as f(5) so, we assume everything works for f(4). This is our faith.

3. Expectation Weds Faith :-

f(5) निकालना है अगर f(4) is given to us so, we know from faith that f(4) = 4 × 3 × 2 × 1 so, this means that

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

Page No. _____
Date

```
import java.io.*;
import java.util.*;
public class Main {

    public static void main(String[] args) throws
        Exception {
        // write code here

    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int fn = factorial(n);

    System.out.println(fn);

    }

    // E => f(5) = 5.4.3.2.1

    // F => f(4) = 4.3.2.1

    // EWF => f(5) = 5.f(4)

    public static int factorial(int n) {

        if (n==0) {

            return 1;

        }
```

Page No. _____
Date

noob mistake!

(*) int fnm1 = factorial(n-1); // (n-1)·(n-2)··· (n-3)···3·2·1

int fn = n * fnm1; // n·(n-1)·(n-2)···3·2·1

return fn;

}

}

// NOOb वाला करेगा: वो सबसे पहली mistake ये करेगा कि वो usne i.e जो factorial(n-1); return करेगा usko recieve करना is very important.

// अगर function void नहीं है तो recieve करना is very important.

(*) Now, we are trying to get our Base value / Base case using low level thinking

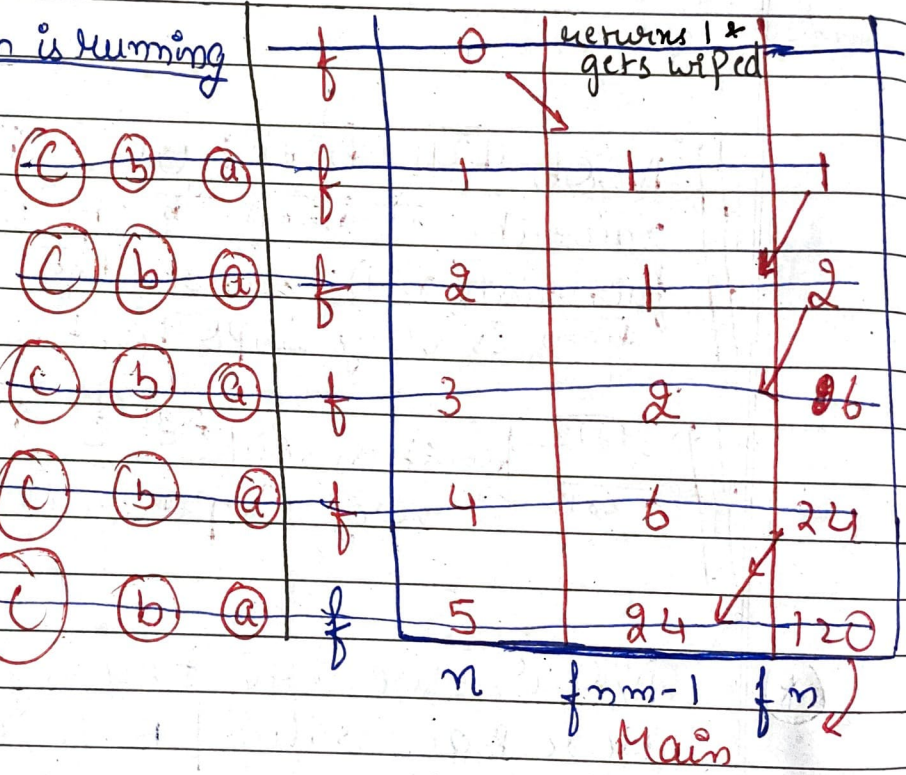
Page No.
Date

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

Stack

Line which is running

(*) ऊपर जाते Time ⓐ चला वापस आते Time ⓑ & ⓒ चलेंगे



(*) पहली baari Main से n=5 pass किया और सो, line no. ⓐ starts running सो, what it does is it calls n=4 सो, ऊपर call लगा गई

(*) अब code ऊपर जो चला है i.e it is के (n-1) i.e. f(4)

Page No.
Date

(*) अब f(4) की line @ चलेगी so, (n-1) ie f(3) is getting called now.

(*) अब जैसे ही f(3) की line @ चलेगी तो f(2) पास होगा i.e f(2) will get called in the stack.

(*) अब अगर उसकी line @ चली तो f(1) को call लगेगा, so f(1) is getting passed

(*) Ab फिर से line @ चली for f(1) so, f(0) i.e. (n-1) will be called

(*) for f(0) we know that factorial of (0) is 1. अब इसके ऊपर जाना नहीं

(*) Ab code में जाके हम लिखेंगे if (n==0){return 1;}

(*) अब जैसे ही f(0) returns 1. It gets wiped off from the stack & we go down to f(1)

(*) इसको जो return 1 है वो इसके नीचे वाला recieve करेगा. पता है कहाँ?

(*) अब f(1) की line (b) chalegi so, fnm1 gets 1 & then, fn = fnm1 * n so, f(n) for f(1) will also be 1*1 = 1.

Page No. _____
Date

(*) जब line (c) चलेगी तो fn return होगी and वो जो fn का value return करेगी that will be stored in fnm1 of the next. Example:- for f(1) line (c) : the value it returns will be recieved in fnm1 of f(2).

(*) now line for f(1) is filled and line (a), (b) & (c) have run & returned the value so, f(1) will come out of stack.

(*) अब f(2) के लिए चलेगी fn = 2 * 1 = 2 so, fn = 2. jab line no. (b) starts executing then for line (c) value of fn for f(2) is returned i.e. 2.

(*) This value gets stored in fnm1 for f(3). Then, fn for f(3) will be

n * fnm1 i.e. 3*2 = 6 when (c) executes f(4) recieves value 6 in fnm1

(*) now f(3) returns value 6 & is removed from stack. f(4) will have fnm1 value as 6. fn of f(4) will be 4 * fnm1 ⇒ 4 * 6 = 24

(*) This value is returned after (c) is executed & f(5) gets fnm1 value as 24.

Page No.
Date

(*) f(4) is removed from stack after returning n*fnm1
(*) In the end f(5) will have its fn value i.e. 5*fnm1 ⇒ 5*24 = 120

(*) The value 120 is sent to main & in the end even f(5) is removed from stack.

(*) Main function returns value of f(5) to be 120.

Page No.
Date

Q Power Linear

⇒ Given :-

→ a no. x

→ another no. n

⇒ Calculate :- x^n

example :- $2^5 = 32$

Lets Start high level Thinking

① Expectation : Power(2, 5) = 2·2·2·2·2

→ expectation is Power(2,5) will give 2^5 by multiplying 2 to itself 5 times

② Faith : Power (2, 4) = 2·2·2·2

→ Faith hai ki Power(2, 4) hme 2×2×2×2 = 16 value provide kre gi

③ * It is important to understand what is varying and what is constant i.e. here 2 is constant i.e. x is constant and n varies from 4 to 5 for faith n = 4 & for expectation n = 5

(3) Expectation weds Faith

(*) हमको 2×2×2×2×2 चाहिए

Power(2, 4) humko 2×2×2×2 देगा
so, we only need to multiply
2 once in Power(2, 4) to get Power(2,5)

Code

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(Strings[] args)
                throws Exception {
        // write code here

        Scanner scn = new Scanner(System.in);
        int x = scn.nextInt();
        int n = scn.nextInt();

        // noob mistake" call लगा देगा but recieve नहीं करेगा" या फिर " main में recieve करना भूल जाएगा" #DontBeANOOB
        int xpn = power(x, n);

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

// E ⇒ p(2,5) = 2·2·2·2·2

// F ⇒ p(2,4) = 2·2·2·2

// E w.r.t F ⇒ p(2,5) = p(2,4) * 2

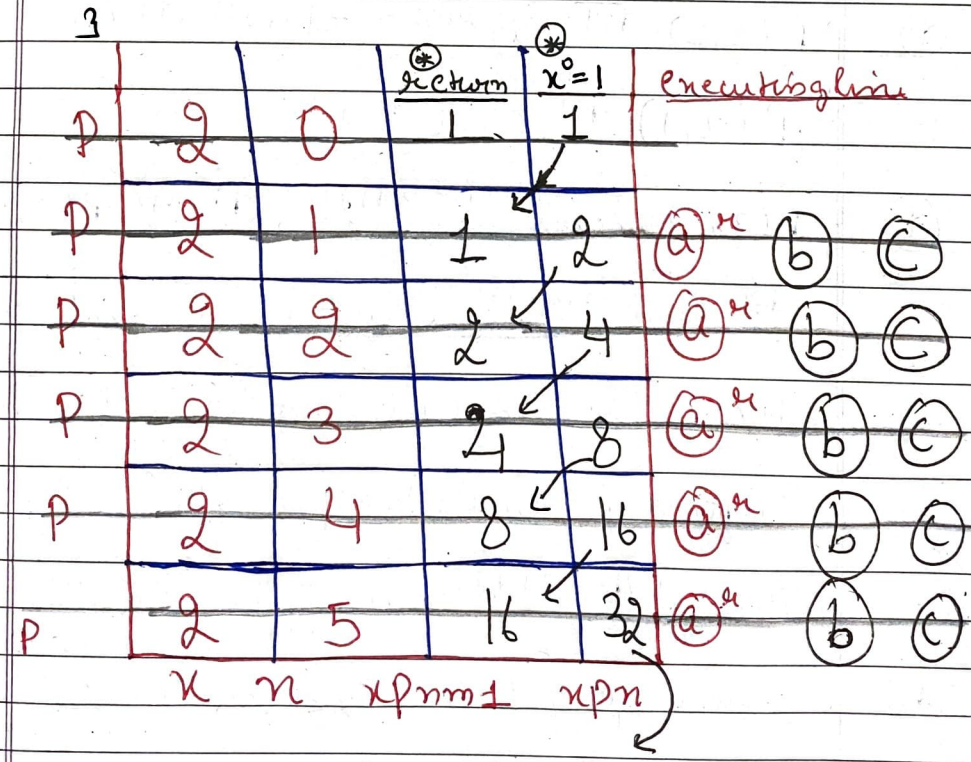
```
public static int power(int x, int n){
    if(n==0){                          // Base case
        return 1;
    }
    int xpnm1 = power(x, n-1); //x*x..n-1 Times
    int xpn = xpnm1 * x; //x*x..n times
    return xpn;
}
```

Page No.
Date

Low Level Thinking

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

base case



Page No. ______
Date

(*) x = 2, n = 5 main से call hua
अब (a) right for P(2,5) चलेगा
तो P(2,4) को call krega as per line (a)

(*) Again when (a) right executes for P(2,4) it will call function P(2,3)

(*) P(2,3) के लिए (a) right चलेगा तो वो P(2,2) को call करेगा

(*) P(2,2) का (a) right चलेगा तो (n-1) i.e P(2,1) को call करेगा

(*) P(2,1) level पे आ गए तो वो P(2,0) को call करेगा which is actually our base case because we know that anything raised to power '0' is 1.
Too अब इसको base case मानलेंगे

(*) likhenge if (n==0){ return 1; }

(*) now, work of P(2,0) is done it has

Page No. ____
Date

returned a value i.e. 1 which will be used by P(2, 1) hence after returning value P(2, 0) gets removed from stack.

(*) अब बात ये है कि return वाला 1 recieve कौन करेगा

so, we see on the left side

xpnm1 is there to recieve its value

hence, xpnm1 gets value '1'.

(*) line (b) चलेगी for P(2, 1) & it will multiply 1st value i.e x with 3rd value i.e xpnm1 and store that value in xpn.

(*) line (c) चलेगी for P(2, 1) and it will return value x * xpnm1 i.e 2 stored in xpn to P(2, 2)'s xpnm1 and P(2, 1) gets removed from stack.

(*) line (b) चलेगी for P(2, 2) because P(2, 2) has value of xpnm1 & x which will give a value of x * xpnm1 i.e. 2 * 2 = 4, this gets stored in xpn of P(2, 2)

(*) line (c) चलेगी for P(2, 2) तब वो return करेगा value 4 to P(2, 3) का xpnm1 and will get removed from stack.

Page No.
Date

(*) 3rd line (b) starts for P(2,3) which will calculate the value of xpn for P(2,3) i.e. x * xpnm1 = 2 * 4 = 8 and store this value in xpn.

(*) line (c) runs for P(2,3) which enables it to return value P(2,3) to xpnm1 of P(2,4) and P(2,3) gets removed from stack.

(*) line (b) runs for P(2,4) which allows it to calculate xpn = x * xpnm1. i.e. xpn = 2 * 8 = 16 which gets stored in xpn.

(*) line (c) executes for P(2,4) hence, it returns value of xpn = 16 to xpnm1 of P(2,5) and gets removed from stack.

(*) finally, P(2,5) has its value of xpnm1 = 16 and x = 2 so, xpn = 16 * 2 = 32 this value gets stored in xpn of P(2,5)

(*) As P(2,5) was called by main function so, on running line (c) P(2,5) returns value xpn = 32 to the main function and gets removed from stack.

Page No. ____
Date

Q Power Logarithmic

(*) अगर 2^5 निकालना था तो 6 space लिया इसने ऐसे ही अगर इसको 2^{10} के लिए चलाते तो 11 space लेगा and हर बार ek multiply कर रहा है e.g. 2^5 के लिए base case को exclude करके 5 बार चला so, do we have something better?

(*) मान लो 2^{18} निकालना है i.e. $2^{18} \Rightarrow 2 \cdot 2^{17}$ // linear
⇓
$2^9 \cdot 2^9$ // logarithmic

(*) Formula 1 ⇒ $x^n = x \cdot x^{n-1}$ (Linear)

(Logarithmic approach)

Formula 2 ⇒ $x^n = x^{n/2} \cdot x^{n/2} \cdot x$ [odd]

⇒ $x^n = x^{n/2} \cdot x^{n/2}$ [even]

(*) formula 1 में x^{16} के लिए 16 ht. का tower hoga but for formula 2 ⇒ 4 ht. का tower hi बनेगा

Page No.
Date

$x^n = x \cdot x^{n-1}$

$2^8 = 2^1 \cdot 2^7$
$2^7 = 2 \cdot 2^6$
$2^1 = 2 \cdot 2^5$
$2^5 = 2 \cdot 2^4$
$2^4 = 2 \cdot 2^3$
$2^3 = 2 \cdot 2^2$
$2^2 = 2 \cdot 2^1$
$2^1 = 2 \cdot 2^0$

Total 8 times

* no. of iteration = n

$x^n = x^{n/2} \cdot x^{n/2}$ (even)
$x^n = x^{n/2} \cdot x^{n/2} \cdot x$ (odd)

$2^8 = 2^4 \cdot 2^4$

$2^4 = 2^2 \cdot 2^2$

$2^2 = 2^1 \cdot 2^1$

$2^1 = 2^0 \cdot 2^0 \cdot 2$

Total 4 times

This is faster

* $(\log \frac{n}{2}) + 1$

i.e. for $n = 8$

no. of iterations = 4

Page No.
Date

Code :-

```
import java.io.*;
import java.util.*;
Public Static void main(String[] args)
                    throws Exception {
// write your code here
Scanner scn = new Scanner (System.in);
  int x= scn.nextInt();
  int n= scn.nextInt();
  int xpn = power (x, n);
  System.out.println(xpn);
}
Public static int power(int x, int n ) {
  if (n==0) {
    return 1;
  }
```

```
int xpb2 = power(x, n/2);        line (a)
int xpn = xpb2 * xpb2;           line (b)
if (n%2 == 1) {
    xpn = xpn * x;               line (c)
}
return xpn;                      line (d)
}
```

Low Level Thinking

(*) Why is this code superior?

⇒ Because the height of the Building/tower is less hence, it requires less space and time taken to get ans also decreases because of decrease in no. of iterations to get final ans. // better space & time complexity

	x	n	xpb2	xpn	Running line
P	2	0	return value 1		
P	2	1	1	X2	a ⓑ ⓒ ⓓ
P	2	2	2	4	a ⓑ ⓒ ⓓ
P	2	4	4	16	a ⓑ ⓒ ⓓ
P	2	8	16	256	a ⓑ ⓒ ⓓ

(*) जब main से call लगेगा तो P(2,8) के लिए line ⓐ चलेगा वो call करेगा P(2,4) को line ⓐ चलेगा फिर 3rd वापस P(2,2) को call लगेगा because line ⓐ चलेगा for P(2,1) को call लगेगा line ⓐ चलेगा for P(2,2) तो finally जब P(2,1) के लिए line ⓐ चलेगा तो P(2,0) को call करेगा और हमें पता है कि 2^0 or x^0 is 1 तो वो 1 return करेगा and this becomes our base case

(*) अब P(2,0) value 1 return करेगा xpb2 of P(2,1) को and will get removed from stack.

(*) अब line (b) चलेगी for P(2,1) so, we get xpn = xpb2 * xpb2 i.e. xpn = 1 but अब line (c) चलेगी अब value of xpn get update because n is odd hence, xpn of P(2,1) = 1*2 = 2

(*) अब line (d) चलेगी for P(2,1) and it will return value of xpn to xpb2 of P(2,2) and gets removed from stack.

(*) अब line (b) चलेगी for P(2,2) so, xpn = 2*2 = 4 and this value gets stored in xpn and line (c) executes but as n = even so, nothing happens. As line (d) executes P(2,2) returns value which

gets stored in xpb2 of P(2,4) and P(2,2) gets removed from stack.

(*) 3rd line (b) चलेगी for P(2,4) so, xpn = 4*4 i.e 16 and this value gets stored in xpn then, line (c) executes as n is even, nothing happens. then, line (d) executes so, P(2,4) returns the value of xpn = 16 to xpb2 of P(2,8) and itself gets removed from stack.

(*) finally, line (b) executes for P(2,8) as xpn = xpb2 * xpb2 so, xpn = 16x16 = 256 and this value gets stored in xpn of P(2,8) as line (c) executes nothing happens because n is even and line (c) changes for n = Odd (Only). In the end line (d) executes P(2,8) returns value 256 to main and gets removed from stack.