High Achievers

Nov23_PSP_17Jan
VIDYA CHAITANYA
Vijay V A
Kevin Theodore E
Mayur Hadawale
Sai Sharath
Prabhakar
Pratham Singh
Suraj Devraye
Shaurya Srivastava
Harshil Dabhoya
Pranadarth S
Varun
Yash Malviya
Panduranga
Pushkar Deshpande
SIJU SAMSON

Nov23_PSP_17Jan
RAMESH R S
Vigneshwaran K
Manjunatha I
M.veronica
Rajeev Singh Khati
Tushar Desarda
ALLEN GEOSHAN M
Shashi
Mateen
Rahul Kotha
Sarat Patel
Manikandan M
sowmya
Poornima Patil
Akanksha Narayan Shinde
Pravin Raj

Why recursion ?

a) Merge Sort / Quick sort

b) Binary Tree / Binary Search Tree / Tries

c) Dynamic Programming

d) Back tracking

e) Graphs

What is recursion ?

Function calling itself

Solving problems using smaller instances of same problem

example

Calculate sum of n natural numbers

$$sum(n) = 1 + 2 + 3 + \dots + n-1 + n$$

$$= sum(n-1) + n$$

How to write recursive code ?

Assumption : Decide what your function does

Main logic : Solve assumption using sub problem

Base Condition : Input for stopping

Function call tracing

Keeping track of all function call sequence, arguments, return values to understand flow of program.

example 1

```
int add (int x, int y)
    return x + y;

int mul (int x, int y)
    return x * y;

int sub (int x, int y)
    return x - y;

int main () {
    int x = 10;
    int y = 20;
    print (sub (mul (add (x, y), 30), 75));
    return 0;
}
```

Function call tracing

sub (mul (add (x, y) , 30) , 75) returns 825 (↑ printed)
(mul (add (x, y) , 30) = 900)

↳ mul (add (x, y) , 30
(add (x, y) = 30)

↳ add (x, y)
(x = 10, y = 20)

We see one function call is calling another, so we need to store them to keep track of them.

Data structure involved in function call

add (x, y) return 10+20 : 30 ✗ remove him from stor
mul (add (x, y) , 30) return 30 * 30 ✗ remove him
sub (mul (add (x, y) , 30) , 75) return 825

Once execution is done please remove

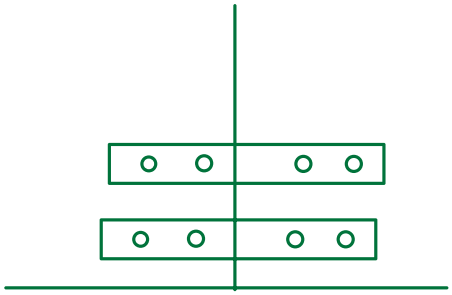
Observation 1:

New function calls are added to top always

Observation 2

You can only remove from top.

Insert at top and delete from top



Stack, you only have access to top.

Factorial of N

If n = 5 find the factorial of N

$n! = 1 * 2 * 3 * \cdots * n = 1 * 2 * 3 * 4 * 5 = 120$

Recursive Code

what should my function do.

Assumption → Given n, Calculate & return $n!$

Main logic

$n! = 1 * 2 * 3 * \cdots * (n-1) * n$

$n! = (n-1)! * n$

Solve assmp. using sub problem

fact(n) = fact(n-1) * n

Base Condition

when to stop

$n! = (n-1)! * n \quad , \quad n = 1$

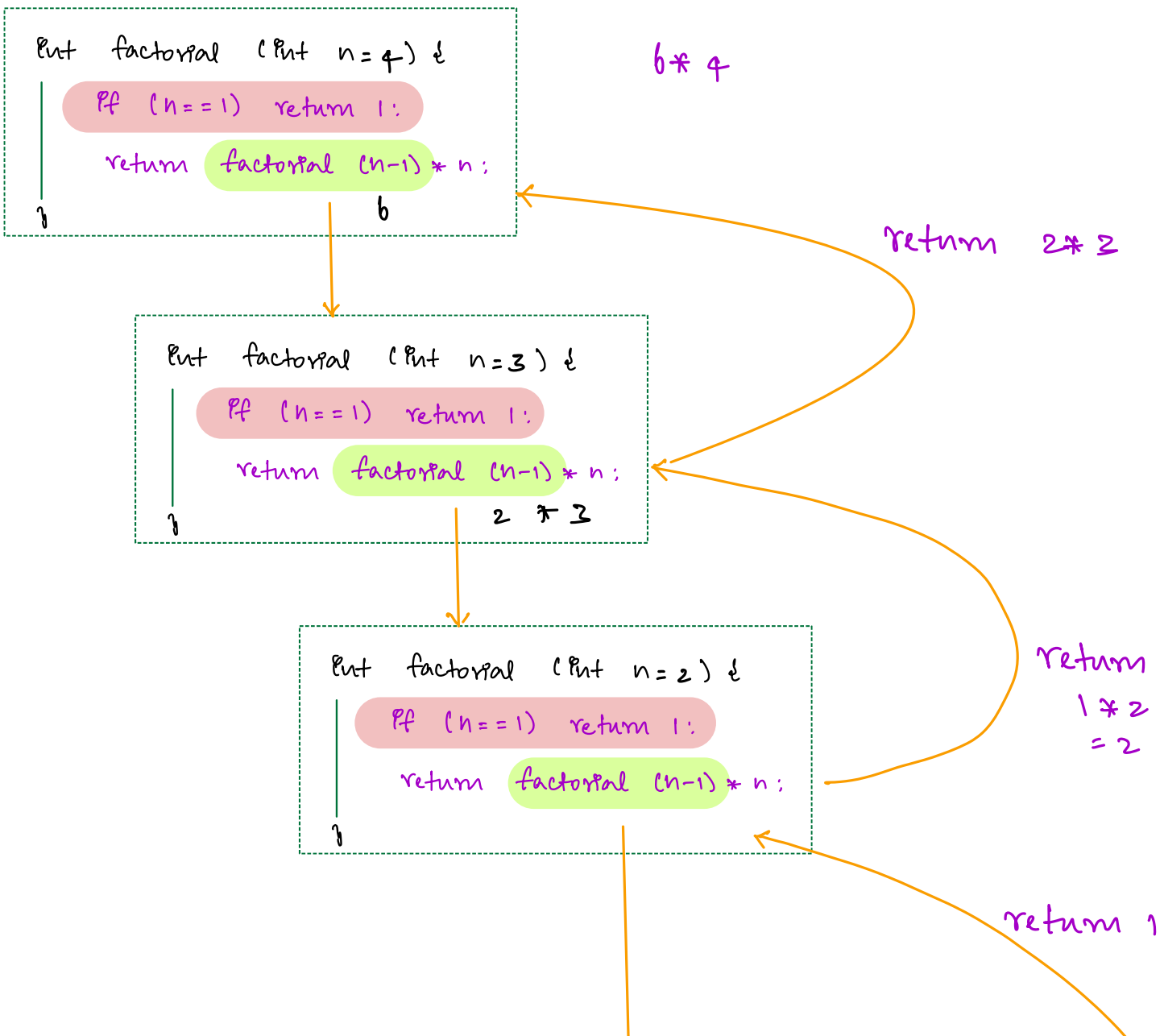
$n == 1 \qquad 1! = 1$

n==1; return 1

pseudo code

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

Tracing (N=4) $4! = 24$



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

Question 2

Given n, calculate and return Fib(n)

Fibonacci Series

n	0	1	2	3	4	5	6
ans	0	1	1	2	3	5	8

Sum of 2 preceding numbers in fib series

Quiz 2

If n = 7, what is the n^{th} number in the fib series

= f(6) + f(5) = 13

Recursion Code

Assumption : Given n, calculate & return fib(n)

Main logic :

Sub problem (smaller instance)

$$fib(n) = fib(n-1) + fib(n-2)$$

Base logic :

f(1) = f(0) + f(-1) does not exist

f(0) = f(-1) + f(-2) does not exist

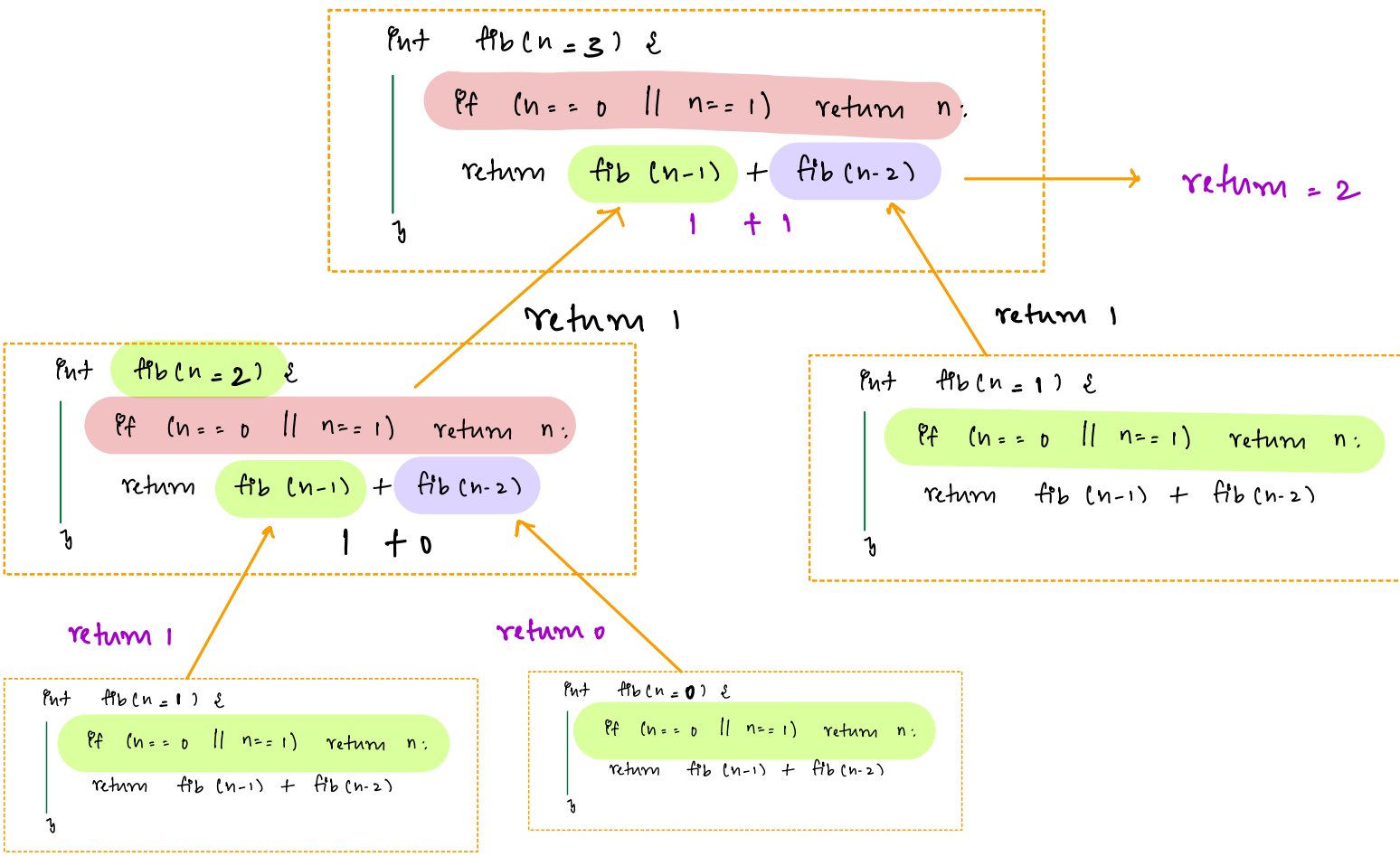
If n==1, return 1

n==0, return 0

Code :

```
int fib(n) {
    if (n==0 || n==1) return n;
    return fib(n-1) + fib(n-2);
}
```

Tracing for n = 3



Question 3

Given two Integers a, n find a^n using recursion

Input : a = 2 n = 3

output: $= 2^3 = 2*2*2 = 8$

approach 1

$a^n = \underbrace{a * a * a * a * \cdots * a}_{n \text{ times}}$

$a^n = a^{n-1} * a$ → smaller Instance of same prblm

pow (a, n) = pow (a, n-1) * a

Recursive Code 1

```
Int  pow ( a , n) {
     If (n == 0)  return 1 ;   // Base Condition
     return  pow ( a , n-1) * a ;  // Sub problem
}
```

Approach 2

Observation If $a=2$, $n=4$ when n is even

$$2^4 = 2^2 * 2^2$$

If $a=2$, $n=5$

$$2^5 = 2^2 * 2^2 * 2$$

we can also divide pow(a,n) as

If n is even

$$pow(a,n) = pow(a, n/2) * pow(a, n/2)$$

If n is odd

$$pow(a,n) = pow(a, n/2) * pow(a, n/2) * a$$

Recursive logic

assumption Given a, n calculate & return pow(a,n) a^n

Main logic:

if n is even

$$pow(a,n) = pow(a, n/2) * pow(a, n/2)$$

else

$$pow(a,n) = pow(a, n/2) * pow(a, n/2) * a$$

Base Condition :

$n == 0$; return 1

Recursive Code :

```
int pow(a, n) {
    if (n == 0) return 1;
    if (n % 2 == 0)
        return pow(a, n/2) * pow(a, n/2)
    else
        return a * pow(a, n/2) * pow(a, n/2)
}
```

Approach 3

In the above code we are repetedly calling pow(a, n/2) multiple times if n is odd or even we can optimize that.

Recursive Code :

```
int fastpow(a, n) {
    if (n == 0) return 1;
    p = fast pow(a, n/2)
    if (n % 2 == 0) return p * p
    else return p * p * a;
```

}y

Code tracing a = 2 n = 9

ans = 512 = 2^9

Put fastpow (a = 2 , n = 9) {

```
if ( n == 0 ) return 1;
p = fast pow (a, n/2)
if ( n-1. 2 == 0 ) return p * p
else return p * p * a;
```

}y

16 * 16 * 2

return 16

Put fastpow (a = 2 , n = 4) {

```
if ( n == 0 ) return 1;
p = fast pow (a, n/2)
if ( n-1. 2 == 0 ) return p * p
else return p * p * a;
```

}y

return 4

Put fastpow (a = 2 , n = 2) {

```
if ( n == 0 ) return 1;
p = fast pow (a, n/2)
if ( n-1. 2 == 0 ) return p * p
else return p * p * a;
```

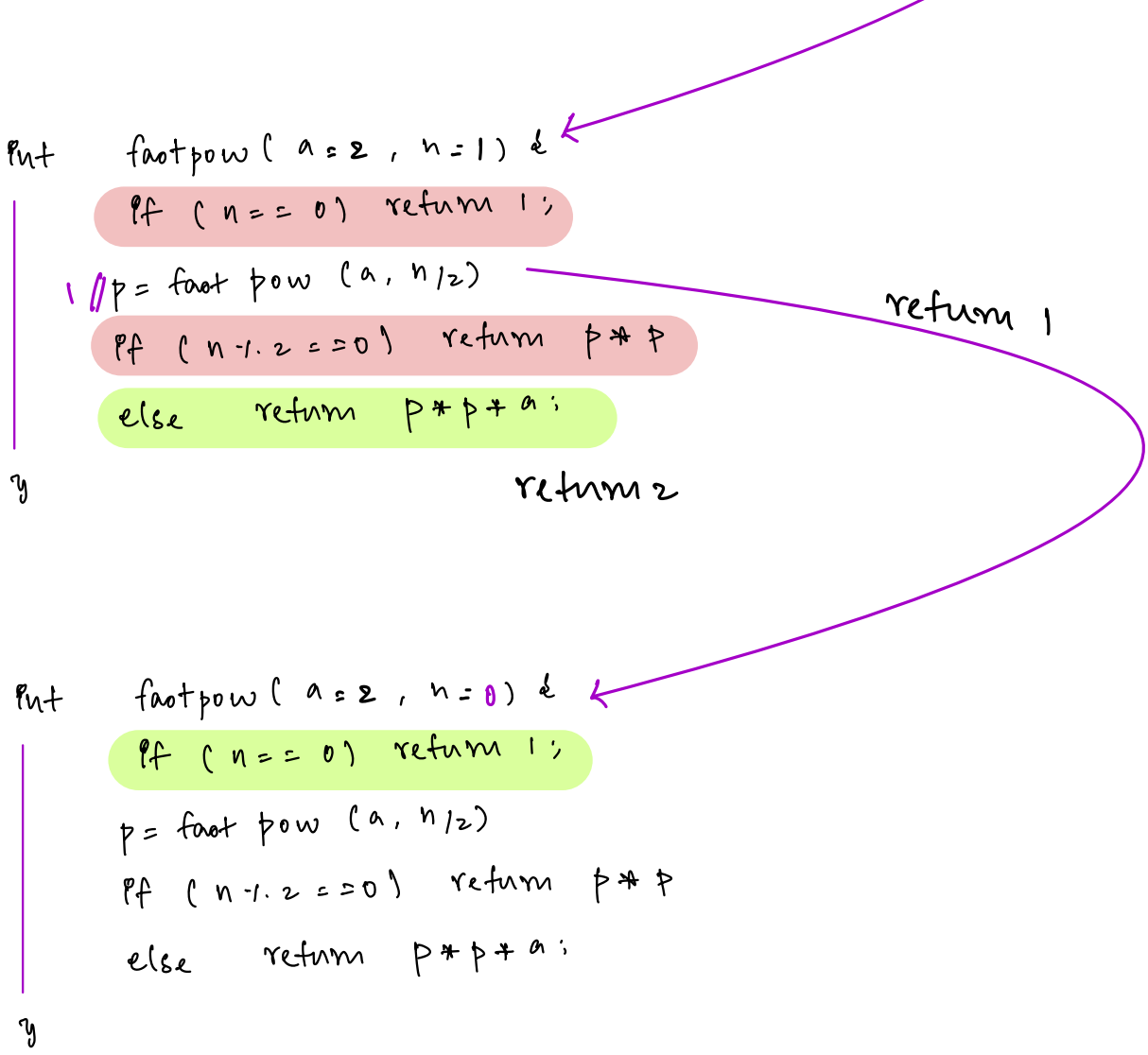
}y

return

2 * 2

= 4

return 2


int fastpow (a = 2 , n = 1) {
if (n == 0) return 1 ;
p = fast pow (a , n/2)
if (n % 2 == 0) return p * p
else return p * p * a ;
}
return 1
return 2
int fastpow (a = 2 , n = 0) {
if (n == 0) return 1 ;
p = fast pow (a , n/2)
if (n % 2 == 0) return p * p
else return p * p * a ;
}

Time Complexity Calculation

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

Time taken to compute

Step 1: Get the recurr. relationship

$T(n) = T(n-1) + 1$

Step 2: Get the T.C for base condition

$T(1) = 1$

Step 3: Generalize for K

$T(n) = T(n-1) + 1$ ——— (1)

$T(n-1) = T(n-2) + 1$ ——— (2)

Substitute 2 in 1

$T(n) = T(n-2) + 1 + 1 = T(n-2) + 2$

$T(n) = T(n-K) + K$

Step 4: Substitute known in unknown

$T(1) = 1$ $\quad T(n) = T(n-K) + K$

$T(n-K)$ — unknown

$n - K = 1 \Rightarrow \boxed{K = n-1}$

Step 5: Substitute value of K

$$T(n) = T(n-(n-1)) + n - 1$$

$$T(n) = n \longrightarrow \text{Time Complexity} - O(n)$$

Time Complexity of pow(a, n)

```
int pow(a, n) {
    if (n==0) return 1;
    if (n%2==0) return pow(a, n/2) * pow(a, n/2);
    else return pow(a, n/2) * pow(a, n/2) * a
}
```

Step 1: Come up with Recurrence Relation

$$T(n) = 2*(T(n/2)) + 1$$

Step 2: Recurrence relation for base condition

$$T(1) = 1$$

Step 3: Generalize recurrence relationship:

$$T(n) = 2 * T(n/2) + 1 \quad \text{——— (1)}$$

$$T(n/2) = 2 * T(n/4) + 1 \quad \text{——— (2)}$$

Substitute 2 in 1

$$T(n) = 2 * (2 * T(n/4) + 1) + 1$$

$$= 4T(n/4) + 3 \quad \text{——— (3)}$$

$$T(n/4) = 2 * T(n/8) + 1 \quad \text{——— (4)}$$

Substitute 4 in 3

$$T(n) = 4 * (2 * T(n/8) + 1) + 3$$

$$= 8 * T(n/8) + 7$$

$$T(n) = 2^k * T(n/2^k) + 2^k - 1$$

Step 4: Substitute unknown value to known value

$$n/2^k = 1 \Rightarrow 2^k = n$$

$$k = \log n$$

Step 5: Substitute k value

$$T(n) = 2^{\log n} + T(n/2^{\log n}) + 2^{\log n} - 1$$

$$= n + T(1) + n - 1$$

$$T(n) = O(n)$$

Mateen

Calculate time complexity of fast pow

```
int pow(a, n) {
    if (n==0) return 1;
    p = pow(a, n/2)
    if (n%2==0) return p*p;
    else return p*p*a;
}
```

O(log n)

T.C = # function calls * T.C (Individual function)

$$\log(n) * O(1)$$

$$= \log(n)$$

Quiz 3:

How many recursive calls are in factorial ( )

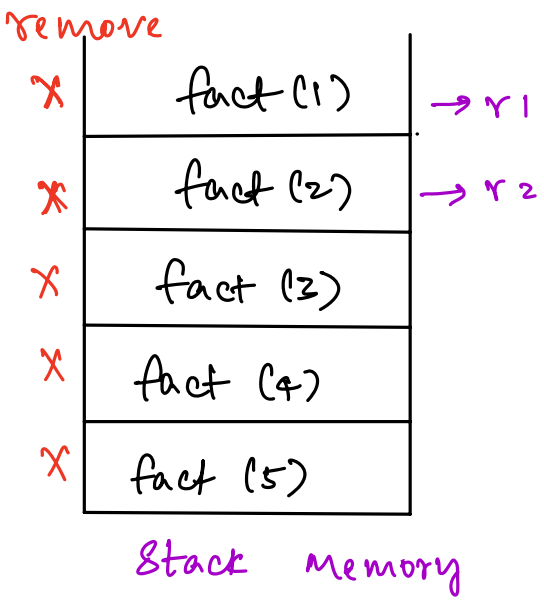
Quiz 4:

How many recursive calls are in Fastpow ( )

Space Complexity :

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

n = 5



∴ use the memory for execution

S.C = O(n)

Fast Pow Space Complexity

```
int pow (a, n) {
    if (n == 0) return 1;
    p = pow (a, n/2)
    if (n % 2 == 0) return p * p;
    else return p * p * a;
}
```

f (2, 0)
f (2, 1)
f (2, 2)
f (2, 4)
fp (2, 9)

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \dots \rightarrow 0$

progression of n

S.C = O (log n)

Doubts

```
int fastpow(a, n) {                    // a=3, n=5
    if (n == 0) return 1;              X
    p = fast pow (a, n/2)              //9
    if (n%2 == 0) return p*p
    else return p*p*a;
}
```

9 * 9 * 3 = 81 * 3

= 243

```
int fastpow(a, n) {                    // a=3, n=2
    if (n == 0) return 1;              X
    p = fast pow (a, n/2)              p=3
    if (n%2 == 0) return p*p           3*3 = 9
    else return p*p*a;
}
```

```
int fastpow(a, n) {                    // a=3  n=1
    if (n == 0) return 1;              X
    p = fast pow (a, n/2)              // p=1
    if (n%2 == 0) return p*p
    else return p*p*a;
}
```

1 * 1 * 3 // return 3

```
int    fastpow ( a, n) {                    // a=3    n=0
    if ( n == 0 )  return 1;
    p = fast pow (a, n/2)
    if ( n % 2 == 0 )  return  p * p
    else     return   p * p * a ;
}
```