Recursion - 2

- Few Problems
- Space & Time Complexity in Rec.

Warm up Q1 Rec Fn to multiply (a,b) without using '*'

$a \geq 0$
$b \geq 0$

```
int multiply (int a, int b){
    if(b==0)                    // Base
        return 0
    return a + multiply(a,b-1)  // Rec
}
```

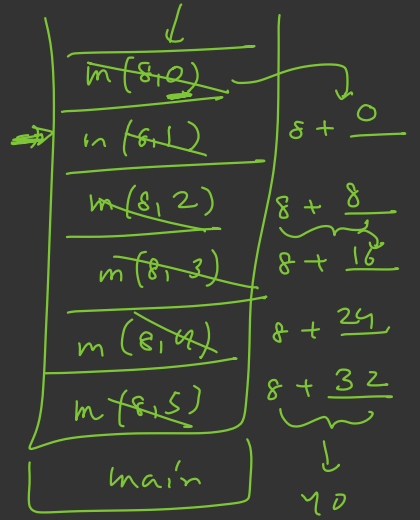
a = 8
b = 5

out = 40

8 x 5
= 8 + 8 x 4

$f(a,b) = a + f(a,b-1)$

```
int multiply (int a, int b){
    if (b==0)                          // Base
        return 0
    return a + multiply(a,b-1)         // Rec
}
```




```
if(a>b)
    multiply(a,b)      a = 8
                       b = 5
else
    multiply(b,a)
```

Space O(b)
Time O(b)

Time O(min(a,b))
Space O(min(a,b))

Power Fn

given two numbers a, n compute a^n

a = 3 n = 5

```
int f(a, n){
    if(n==0)
        return 1
    return a * f(a,n-1)
}
```

$3^5 = 3 * 3*3*3*3$

$= 3 * 3^4$

↑ Sub

$f(a,n) = a * f(a,n-1)$

$a^n = a * a^{n-1}$

$a^0 = 1$

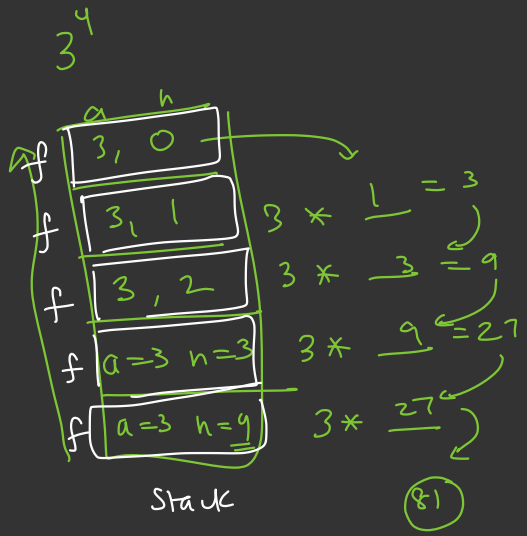
time : O(n)

Space : O(n)

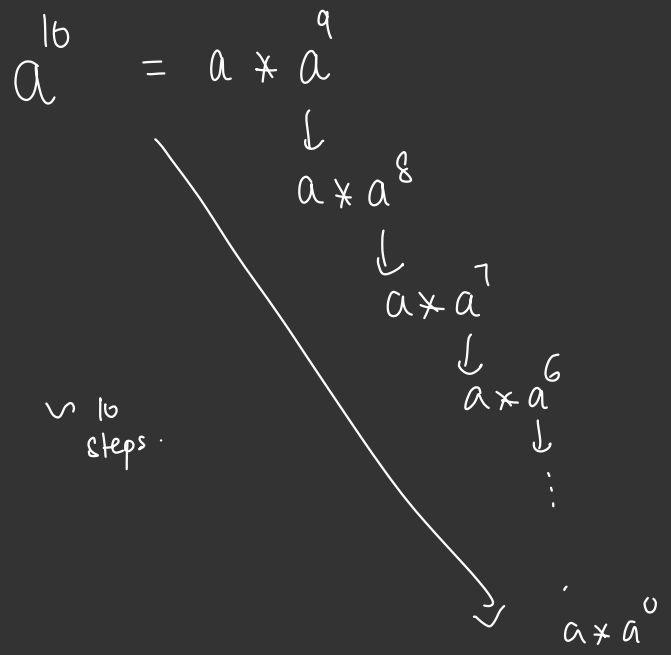
```
int f(a, n){
    if(n==0)
        return 1
    return a * f(a,n-1)
}
```

Space → O(n)

Time → O(n)



$3^4 = 81$


$a^{16} = a * a^{9}$
$a * a^{8}$
$a * a^{7}$
$a * a^{6}$
$a * a^{0}$
16 steps.

$a^{10} = (a^5)^2$

$\downarrow$

$a \cdot (a^2)^2$

$\downarrow$

$(a)^2$

3 steps

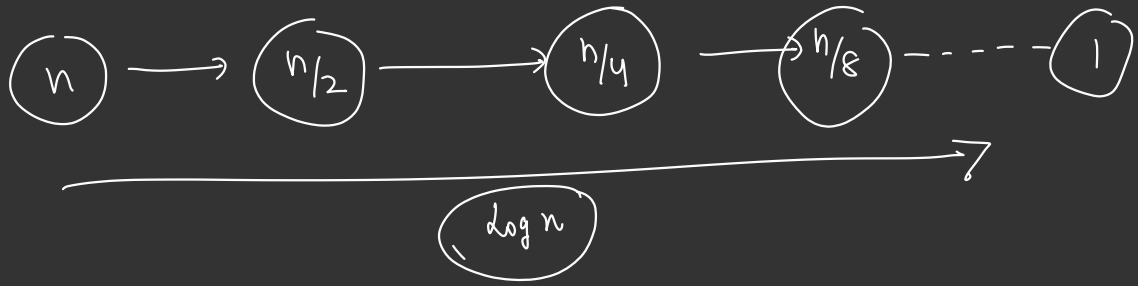
$a^5 \times a^5$

$\downarrow$

a^{10}

$$a^n = \left(a^{n/2}\right)^2 \quad \text{if } n \text{ is even}$$

$$a^n = a\left(a^{n/2}\right)^2 \quad n \text{ is odd}$$



Rec

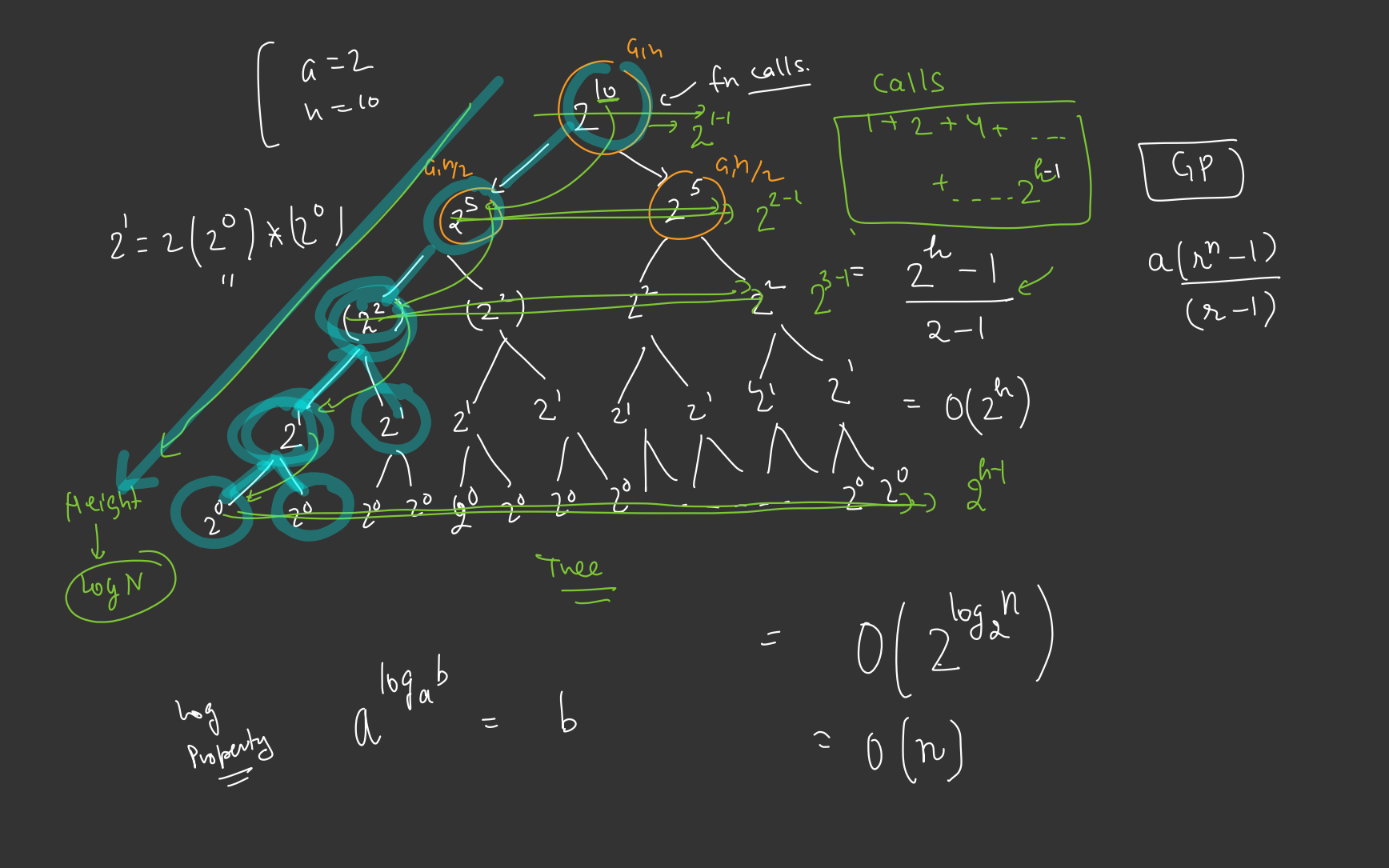
$$f(a,n) = \begin{cases} f(a,n/2)^2 & \text{if } n \text{ is even} \\ a\, f(a,n/2)^2 & \text{if } n \text{ is odd} \end{cases}$$

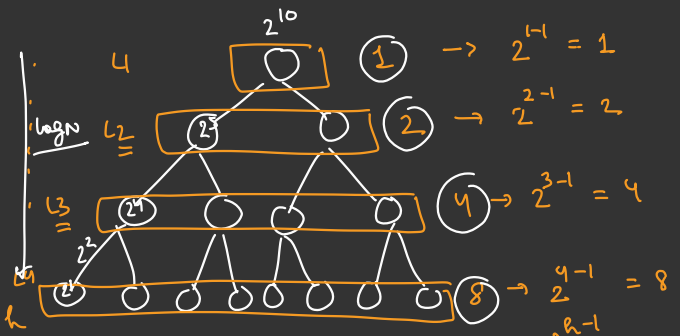
Base

$$f(a,0) = 1$$

```
int f(int a, int n){
    if(n==0)
        return 1
    if(n is even)
        return f(a,n/2) * f(a,n/2)
    else{
        return a * f(a,n/2) * f(a,n/2)
    }
}
```

Correct but inefficient


a = 2
n = 10
fn calls.
Calls
1 + 2 + 4 + ... + 2^(h-1)
GP
a(r^n - 1) / (r - 1)
2^1 = 2(2^0) * (2^0)
2^(h-1) = (2^h - 1) / (2 - 1)
= O(2^h)
Height
Log N
Tree
Log Property
a^(log_a b) = b
= O(2^(log_2 n))
= O(n)



$= 15$

$2^0 + 2^1 + 2^2 + \dots\dots 2^{h-1}$ ← Sum of Series (GP)

$\rightarrow \frac{a(r^n - 1)}{r - 1}$

$\frac{1 \cdot (2^h - 1)}{2 - 1} = 2^h - 1$

$= O(2^h)$

$= O(2^{\log_2 N})$

Time $= O(N)$

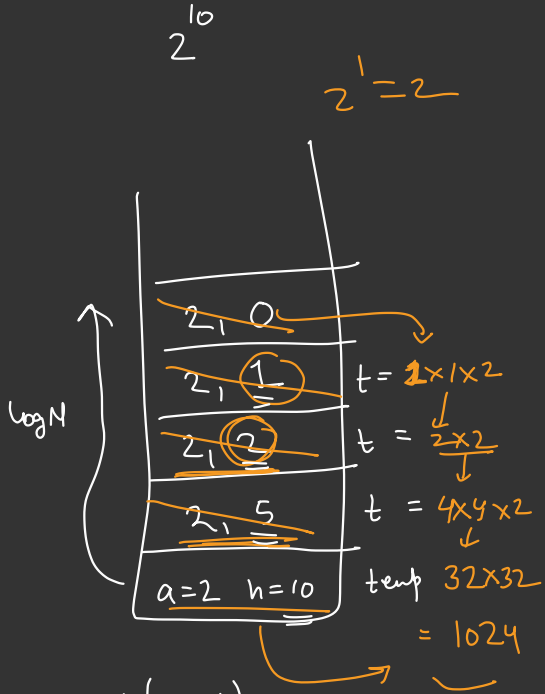
Space $= O(\log N)$

Rec

$$f(a,n) = \begin{cases} f(a,n/2)^2 & \text{if } n \text{ is even} \\ a\, f(a,n/2)^2 & \text{if } n \text{ is odd} \end{cases}$$

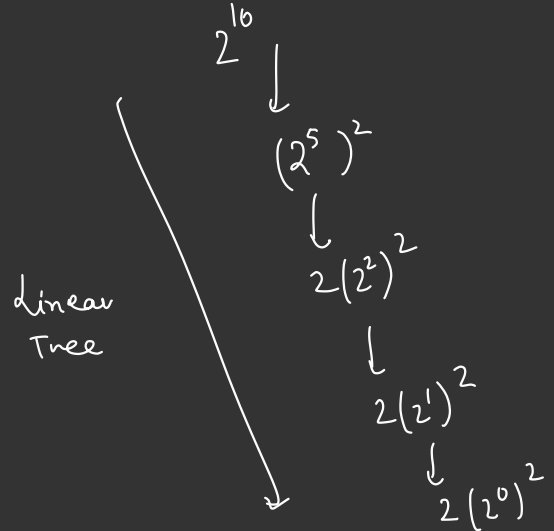
```
int f(a,n) {
    if(n==0)
        return 1
    temp = f(a,n/2)
    temp = temp * temp
    if(n is odd)
        return a * temp
    return temp
}
```

$2^{10} = (2^5)^2$, $2^5 = 32$



O(log N)

Time & Space



a^n = a x a x a ---- a

ans = 1

loop { for (i=1 ——— n) {
ans = ans * a
}

Time : O(n)

Space : O(1)

4 ways ① Iteration - 1 way $O(N), O(1)$ — Bitmasking $O(\log N), O(1)$

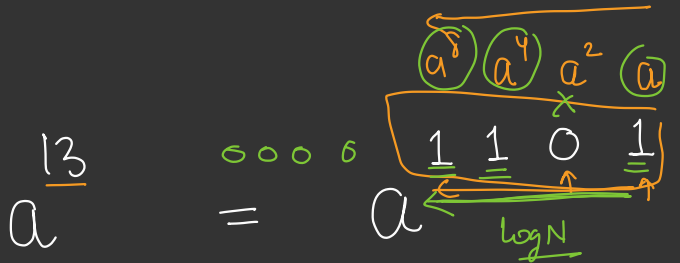
② Rec - 3 ways $O(N)$ $O(N)$ $O(\log N)$ Time, $O(\log N)$ Space

10:10 Break

Rec

$$f(a,n) = \begin{cases} f(a,n/2)^2 & \text{if } n \text{ is even} \\ a\, f(a,n/2)^2 & \text{if } n \text{ is odd} \end{cases}$$

$a^{13} = a^{8} \cdot a^{4} \cdot a^{1}$



$= a^{13}$

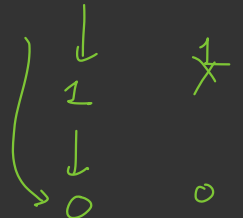
13 → 110*

6 → 110

3 → 1*

logN

1 1 0* → $n = n >> 1$ → $n/2$

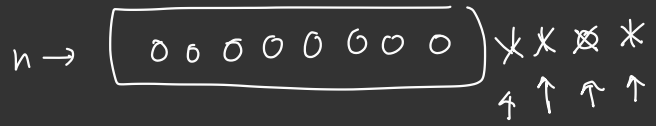


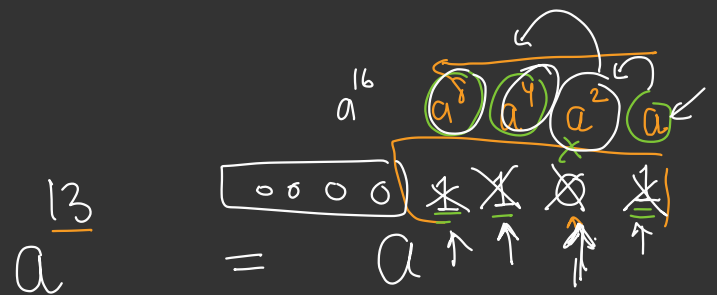
a^n = a

atmax 32 bits

0 0 0 0 1 1 0 1 0 0 1

Log N will be less than ≤32





ans = 1 * a * a^4 * a^8

= a^{13} Yes :)

Input: a, n Out: a^n

a = a
n = 13

```
ans = 1
while(n > 0){
    lastBit = (n & 1)
    if(lastBit == 1){
        ans = ans * a
    }
    a = a * a
    n = n >> 1
}
return ans
```

TC: O(log N)
SC: O(1)

a = 3 n = 5

$a = 3 \to 9 \to 81 \to 81^2$

ans = 1 * 3 * 81

= 243

1 * 3 * 81 * 81 * 81 * 81

3^5

5 → 2 → 1 → 0

$3^5 = 3^1 \, 3^4$

Introduce m = 3^5

fast Modulo exponentation / Binary

Warm Up Practice

given a number, find sum of digits using rec

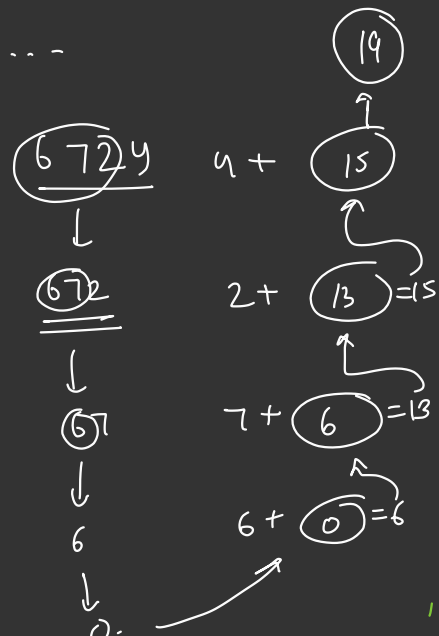
$N \rightarrow \frac{N}{10} \rightarrow \frac{N}{100} \cdots$

$N = 6724$

```
int Sum (int N){
    if(N==0)
        return 0
    return N%10 + Sum(N/10)
}
```

TC → $O(\log_{10} N)$

SC → $O(\log_{10} N)$



Last question. $a^n \% m$ $[m \leq 10^9]$

Modulo Exponentiation

```
int powerMod(int a, int n, int m){
    if(n==0)
        return 1
    long t = powerMod(a, n/2, m)
    t = (long(t) * t) % m
    if(n is odd){
        return (a * t) % m
    }
    return t.
}
```

Doubts

Longest Subarray zero Sum