What is Recursion?

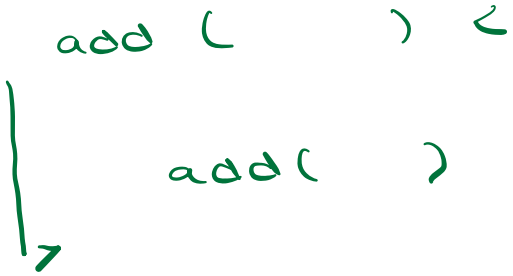
How to write recursive code?

Function call Tracing

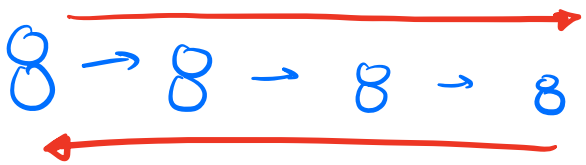
3 problems

TC and SC

Recursion → function calling itself



1. Digger doll → smaller doll
2. similar
3. end doll



Recursion → solving larger problems using smaller similar subproblem

$$sum(N) = \underbrace{1 + 2 + 3 + 4 + \ldots + N-1}_{} + N$$

$$sum(N) = sum(N-1) + N$$ → Recursive relation

Q. Write recursive fn for sum of N natural no.

1. Assumption : Decide what your fn does

2. Main Logic :

   Recursive relation to code /
   breaking big problem into smaller
   problem

3. Base condition

   terminating condition /
   smallest prob for which you already
   know answer

```
// given N, return sum of 1st N natural nos.
int sum (n) {

Base condn   if (n == 1)   return 1

             return   sum(n-1) + n

}
```

sum(4)
↓
sum(3) + 4
↓
sum(2) + 3
↓
sum(1) + 2
↓
1

Fn call Tracing

seq of fn calls that are made when a program is executed

↓

fn calls, arguments, return value

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

int mul(int x, int y, [illegible]) {
    return x * y [illegible];
}

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

void print(int x) {
    cout << x << endl;
}

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

main() {
x = 10, y = 20 ✓
print(sub(mul(add(x,y), 30, 75)))
}

print(sub(mul(add(x,y), 30, 75)))
↓ ↑ 825

sub(mul(add(x,y), 30), 75)
↓ ↑ 900

mul(add(x,y), 30)
↓ ↑ 30

add(x,y) {
return x+y
}

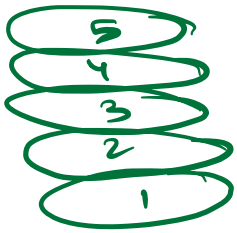
O/P
825

Fn call stack

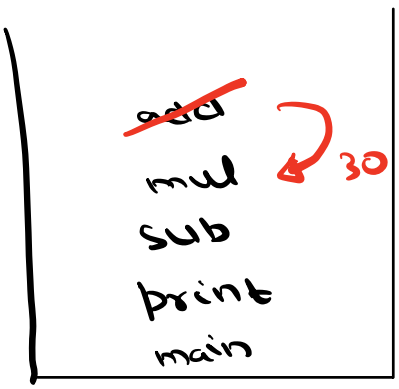
↓

LIFO

Last in First Out



stack
fn calls
arg
return values



Top of the stack
↓
fn is executed

1. Given a +ve integer N, find factorial of N

5! = 120

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$5! = 5 \times 4!$

① $fact(N) = N \times fact(N-1)$ recursive relation

$N! = N \times (N-1)!$

$0! = 1$

② if $n == 1$ return 1

Memory Limit Exceeded / stack overflow

```
// Given input N, return N!
int factorial (int N) {
    if (n == 0)
        return 1
    return factorial (N-1) x N
}
```

fact(0)
↓
fact(-1)
↓
fact(-2)
↓
-3
↓
⋮

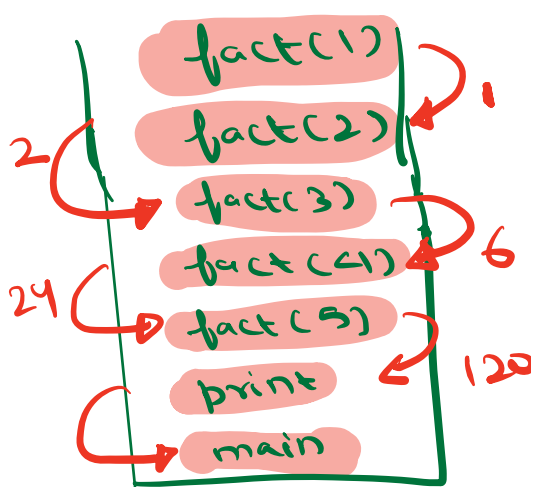
f(1)
↓
f(0) x 1
↓
f(-1)

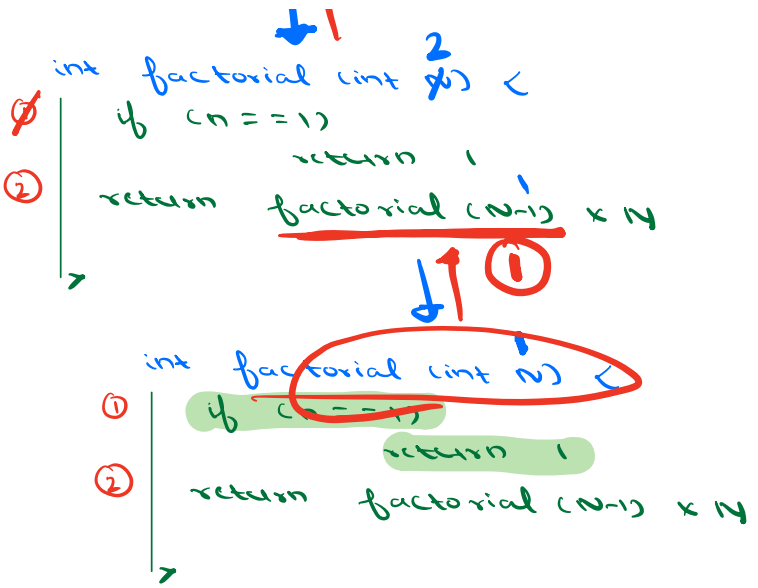
f(2) → 2
↓
1 f(1) x 2

```
main () {
    print ( factorial(5))
}
```

print (factorial (5)) ↓ ↑ 120

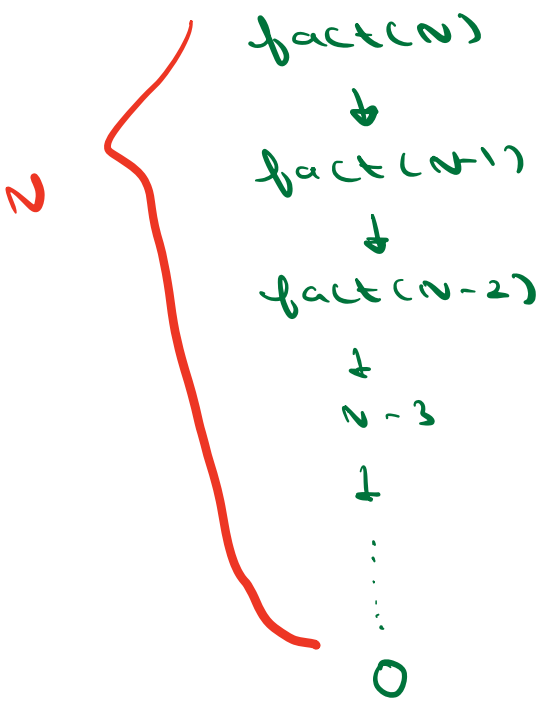
```
int factorial (int N) {                 5
    if (n == 1)
        return 1
    return factorial (N-1) x N          4
}
```
↓ ↑ 24
```
int factorial (int N) {                 4
    if (n == 1)
        return 1
    return factorial (N-1) x N          3
}
```
↓ ↑ 6
```
int factorial (int N) {                 3
    if (n == 1)
        return 1
    return factorial (N-1) x N          2
}
```
↓ ↑ 2





TC = No. of recursive calls x time taken of each recursive call

TC : N x O(1)
= O(N)



2. Fibonacci series

	0	1	2	3	4	5	6	7
→	0	1	1	2	3	5	8	13

Fib (6) = Fib (5) + Fib (4)

5 + 3

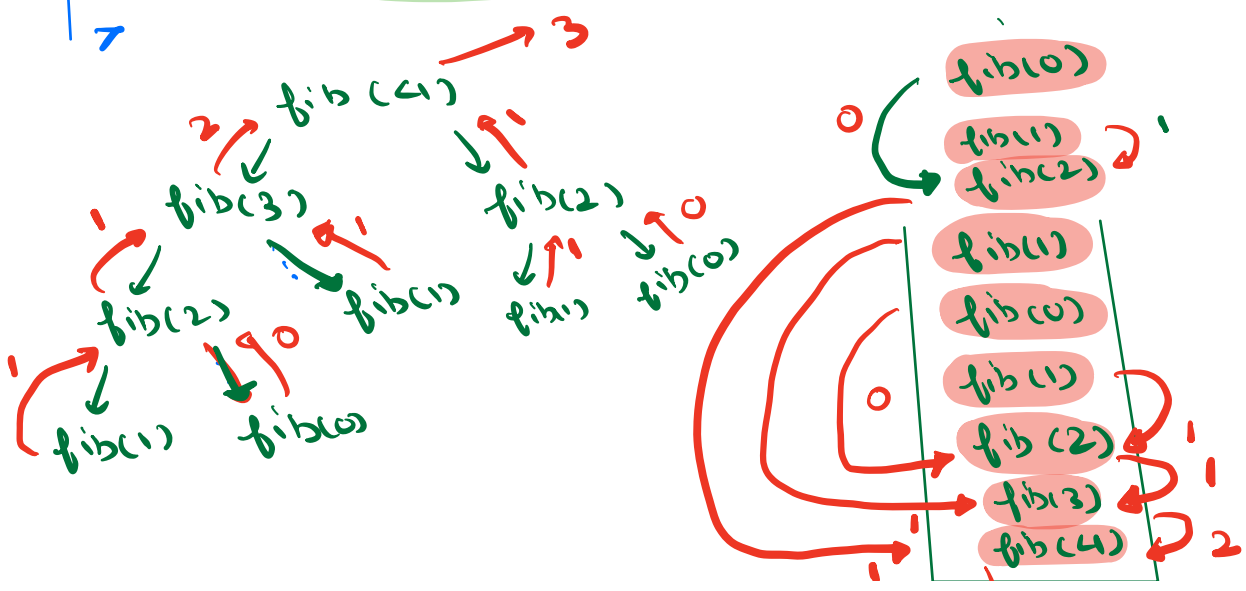
= 8

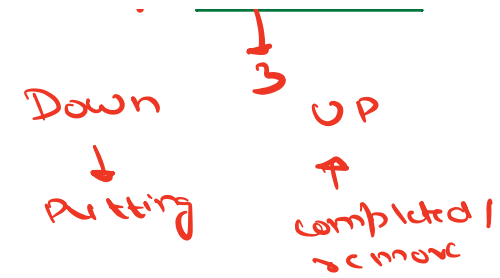
Fib(n) = Fib(n-1) + Fib(n-2)

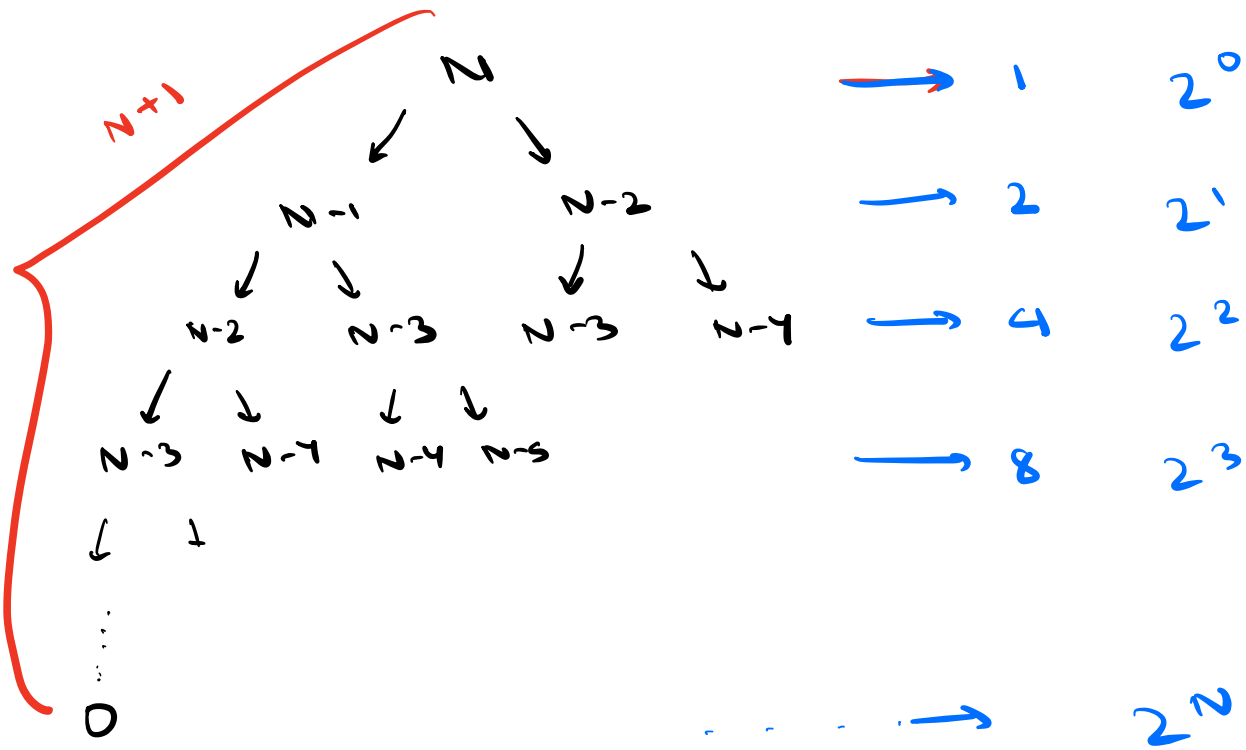
if (n == 0) return 0

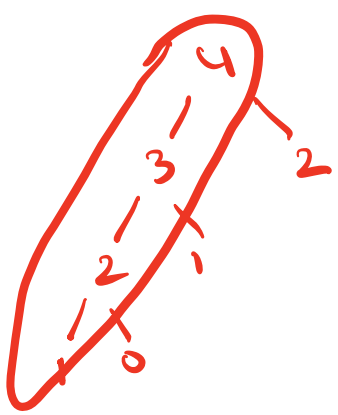
if (n == 1) return 1

```
// Given N, return Nth fibonacci no.
int fibonacci (int N) {

    if (n <= 1)   return n

    return fibonacci (N-1) + fibonacci (N-2)
}
```









Sum $= \frac{a(r^N - 1)}{r - 1}$

Total fn calls =

$2^0 + 2^1 + 2^2 + \ldots + 2^N$

$= \frac{1\,(2^{N+1} - 1)}{2 - 1}$

$= 2^{N+1}$

TC: $O(2^N)$

10:57

3. Find a^n using recursion.

$a = 2 \quad n = 3 \quad \text{ans} \rightarrow 8$

$a^5 = a \times a \times a \times a \times a$

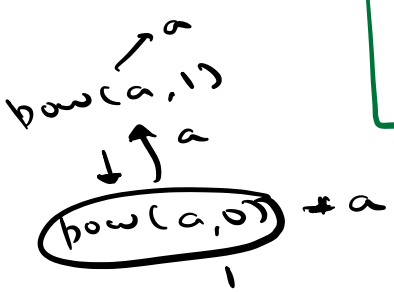
$a^5 = a^4 \times a$

$a^{10} = a^9 \times a$

$a^n = a^{n-1} \times a$

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

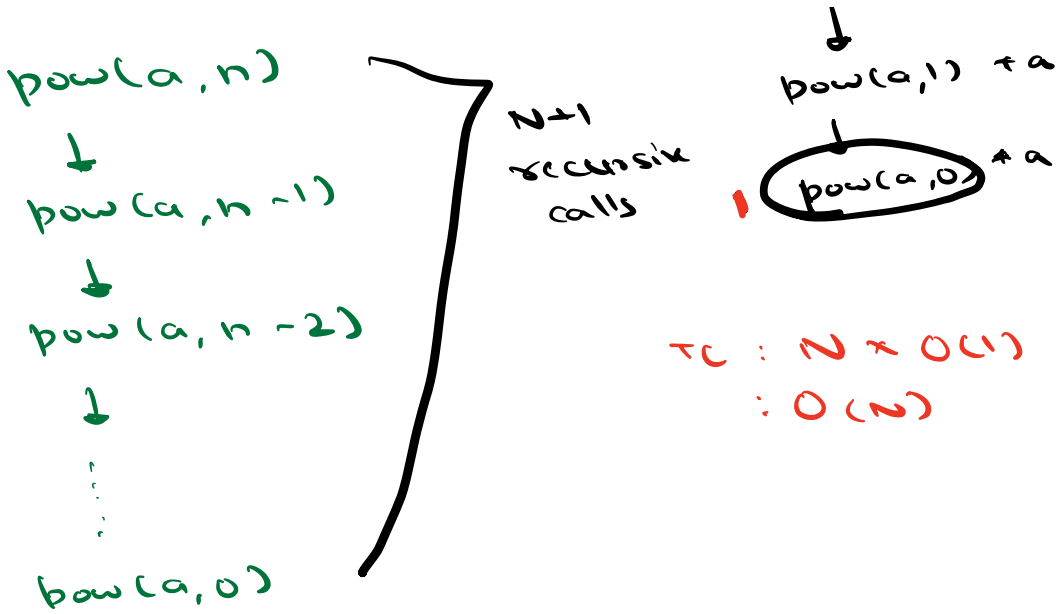
```
if (n==0)
    return 1
```



a^6
↓
$a^5 \times a$
↓
$a^4 \times a$
↓
$a^3 \times a$

```
// Given a,n ; return a^n
int pow (int a, int n) {
    if (n==0) return 1
    return pow(a,n-1) * a
}
```

pow(a,4)
↓
pow(a,3) * a
↓
pow(a,2) * a



TC : N x O(1)
: O(N)

$a^{10} = a^5 \times a^5$

$a^{11} = a^5 \times a^5 \times a$

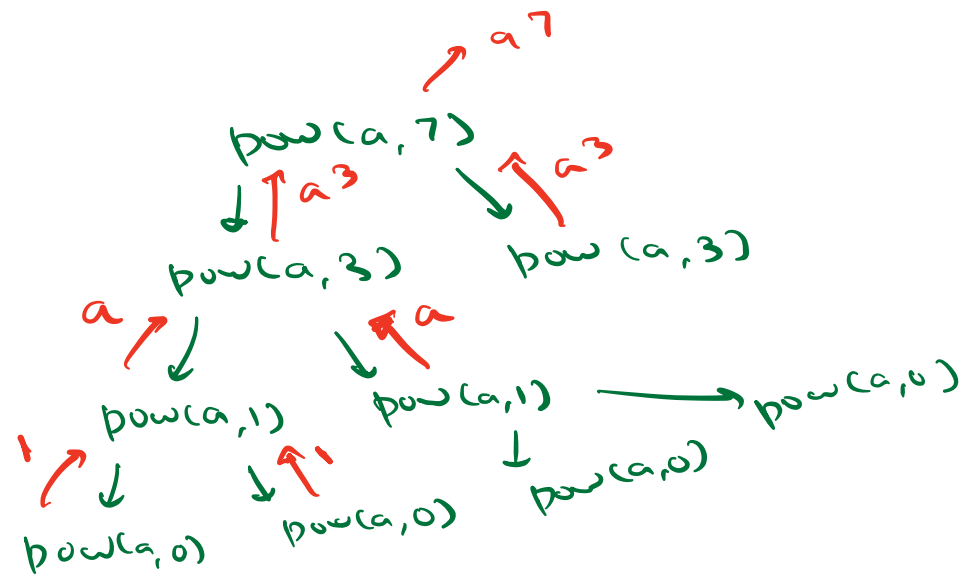
$a^6 = a^3 \times a^3$

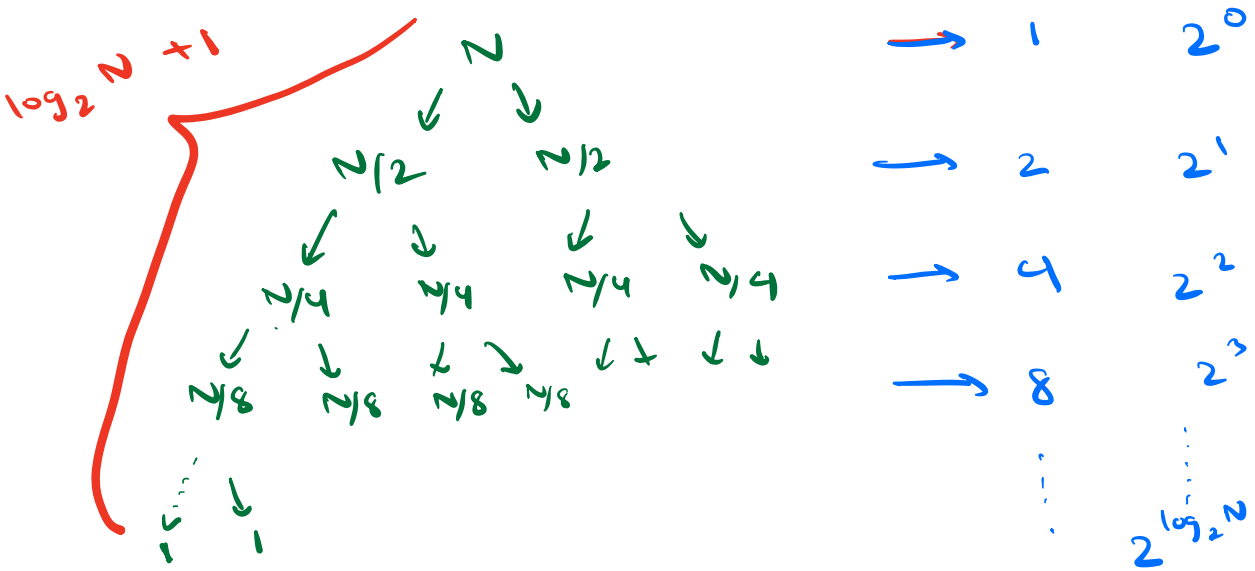
$$a^N = \begin{cases} a^{N/2} \times a^{N/2} & \text{if } N \text{ is even} \\ a^{N/2} \times a^{N/2} \times a & \text{if } N \text{ is odd} \end{cases}$$

$a^0 = 1$

if (N==0) return 1

```
int pow(int a, int 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
}
```





8 →(1) 4 →(2) 2 →(3) 1 →(4) 0

$N \to N/2 \to N/4 \to \dots 1 \to 0$

$\log_2 N \qquad +1$

No. of recursive calls $= 2^0 + 2^1 + 2^2 + \dots + 2^{\log_2 N}$

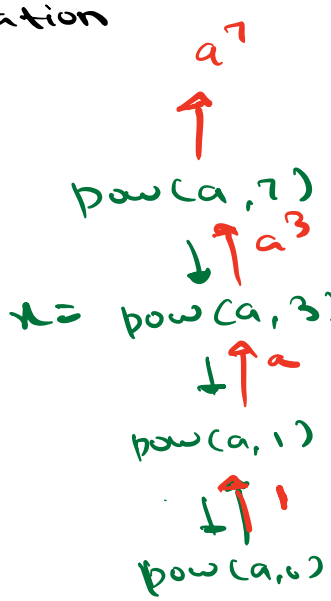
$$= \frac{2^0 (2^{\log_2 N + 1} - 1)}{2 - 1}$$

$$= 2^{\log_2 N} \cdot 2$$

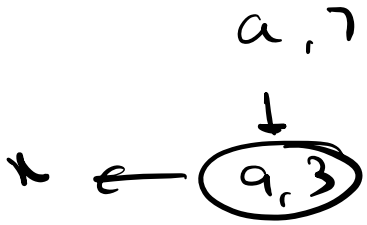
TC: $O(N)$

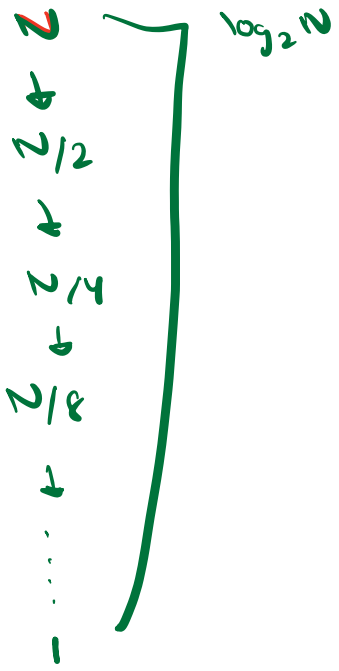
Fast Power / Fast Exponentiation

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

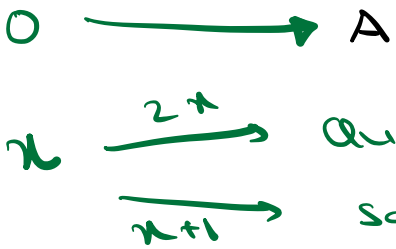


TC: $O(\log_2 N)$

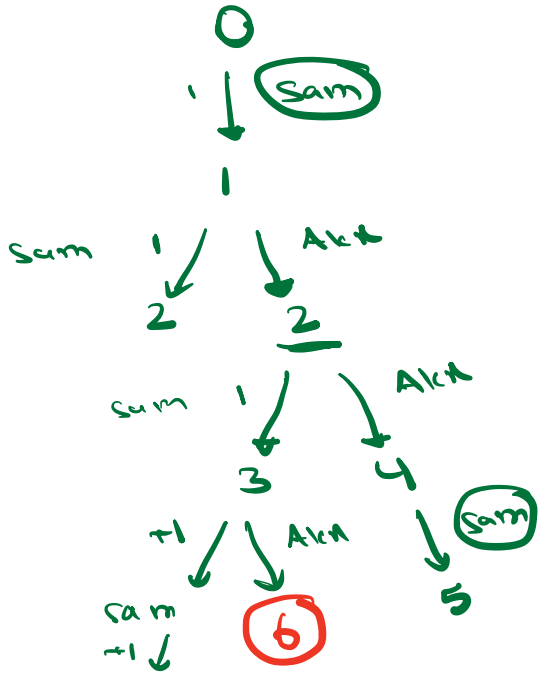

a, 7
9, 3


N
N/2
N/4
N/8
1
$\log_2 N$

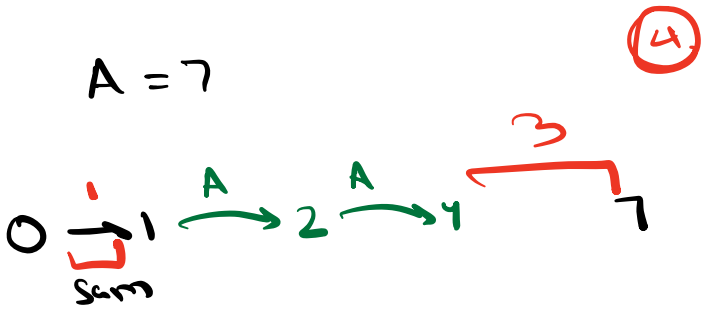
Doubts



A = 5

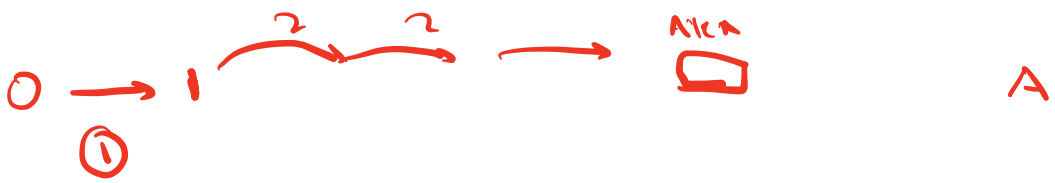


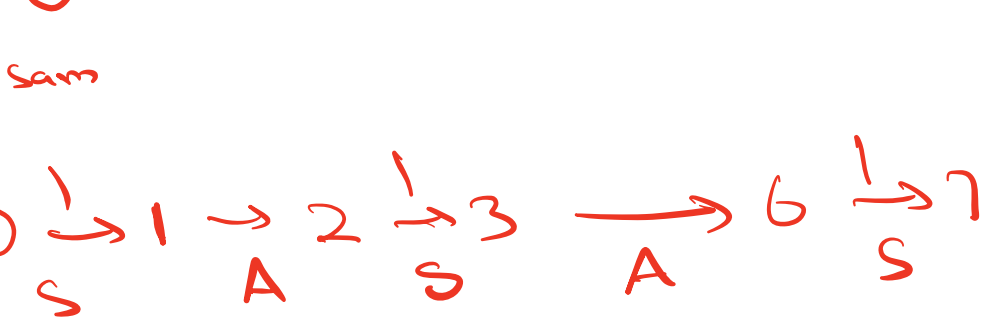
A = 7



Alex → 2x (x)

Sam → +1





ar of 1 and 0

Find cnt of subarrays where OR = 1

TC : O(N)