Recursion :

→ How to write recursive code
→ Working
→ TC/SC

Why ?

→ Merge sort
→ Binary Trees
→ Dynamic Programming
→ Backtracking

Recursion : { function calling itself }

Solve a problem using smaller version of the same problem.
↳ sub problem

Sum(N) = $1+2+3+4+\cdots+N$

$1+2+3+4+\cdots+N-1 + N$

sum(N-1) + N

Recursion Code

Magic steps

1) **Assumption**: Decide what your function does, and assume it does exactly that
2) **Main Logic**: Solving Assumption with subproblem
3) **Base Condition**: When should code stop.

Tracing sub(mul(add(10,20),30),75)

```
int sub(x,y){
{   x-y
}
int add(x,y){
|   x+y
}
int mul(x,y){
|   x*y
}
```

add(10,20) = 30

↑ 30

mul(add(10,20),30) = 900

↑

sub(mul(add(10,20),30),75)

= 825

int fact(N){

Calculate factorial of N

1×2×3×4×.....×N

Assumption: return the factorial of N

1×2×3×4---×N-1×N

Base case: if(n==1)

return 1

Main Logic: ans = fact(N-1)×N

return ans

}

f(N) = 1×2×3×.....N-1 ×N

f(n-1)

f(0)

↑

f(1)

↑

f(2)

↑

f(3)

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

1	2	3	4	5	6	7	8	9	10	11	12
fib: 1	1	2	3	5	8	13					

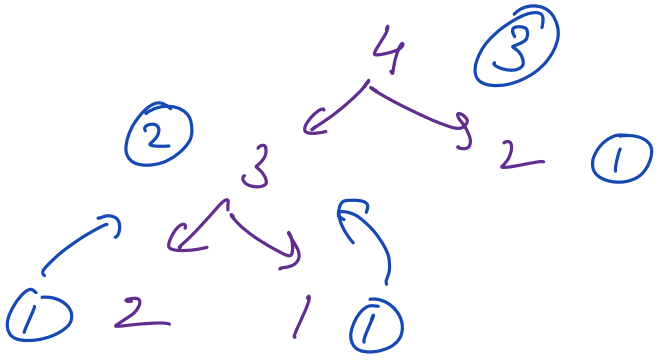
int fib(N) { N^th fibonacci numbers

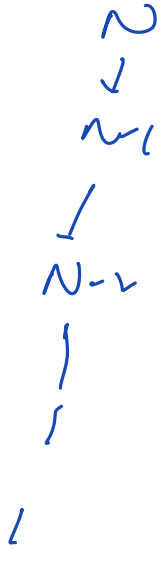
Assumption: return n^{th} fibonacci num

Base Case:
```
if(n==1 || n==2)
    return 1
```

Main logic
```
ans = fib(n-1) + fib(n-2)
return ans
```

}





SC: O(n)

Example :

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

int mul(x, y){
    return x*y
}

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

```
main(){
    int x=10, y=20, z=30
    int a= add(x,y)      a=
    int m= mul(a,z)      m=
    int s= sub(m,75)     s=
    print(s)
}
```

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

sub(mul(add(x,y), z), 75)

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

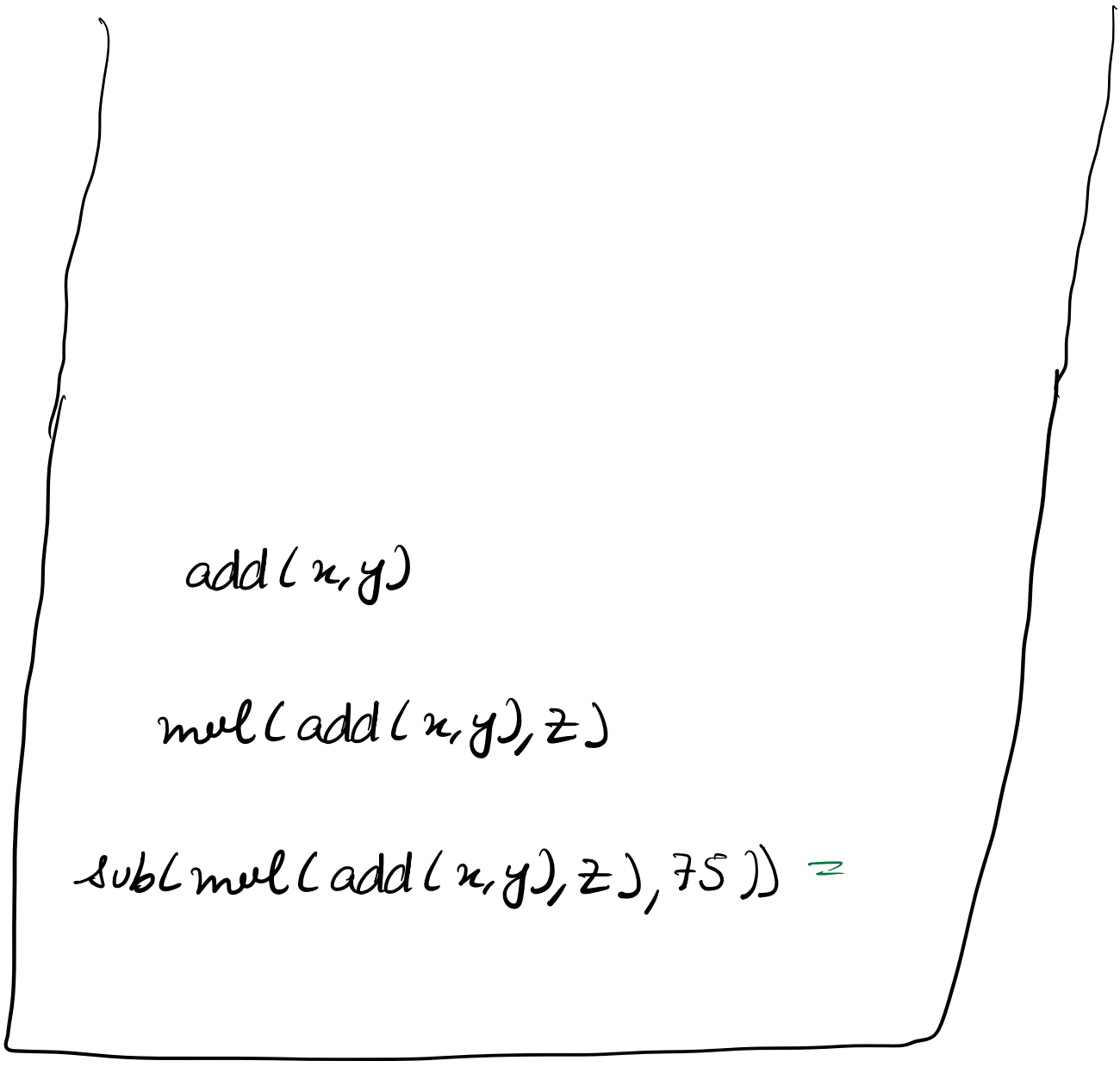
y = 75

mul(add(x,y), z)

add(x,y)

10 20

Internal working (Call Stack)



Works like an Idli cooker

Stack

```
int sum(N) {
  if (N==1) return 1
  return  N + sum(N-1)
}

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

TC

sum(5)

sum(5)
5 + sum(4)

Recursion Tree

sum(4)
4 + sum(3)

sum(3)
3 + sum(2)

sum(2)
2 + sum(1)

Q1 pow(a, n) a^n Dont worry about overflow

↳ n >= 0

$a^n = a \times a \times a \times a \times \dots \times a$

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

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

```
int pow (int a, int n) {
```

Assumption: returns value a^n

Base case: if (n==0) return 1

O(n)

return a * pow(a, n-1)

}

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

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

$a^{10} = a^2 \times a^8$

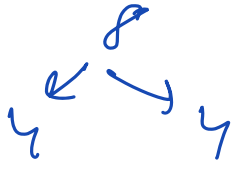
$a^{10} = a^3 \times a^7$

$a^{16} = a^8 \times a^8$

$a^n \quad a^{n/2} \quad a^{n/2}$

$a^{23} = a \times a^{11} \times a^{11}$

$a \times a^{n/2} \times a^{n/2}$



$a^{10} = a^9 \times a^1$

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

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

$a^{14} = a^7 \times a^7$

$a^{19} = a \times a^9 \times a^9$

$a^{16} = a^8 \times a^8$

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

  else { // a^n = a * a^n/2 * a^n/2
    return a * pow(a,n/2) * pow(a,n/2)
  }
}
```

$$T(N) = T(N/2) + T(N/2) + 1$$

$$T(N) = 2T(N/2) + 1$$

$$T(N/2) = 2T(N/4) + 1$$

$$T(N) = 2(2T(N/4) + 1) + 1$$

$$= 4T(N/4) + 2 + 1 \qquad T(1) = 1$$

$$= 4T(N/4) + 3$$

$$T(N) = 8T(N/8) + 7$$

$$= 16T(N/16) + 15$$

$$= kT(N/k) + k - 1$$

Put $k = N$

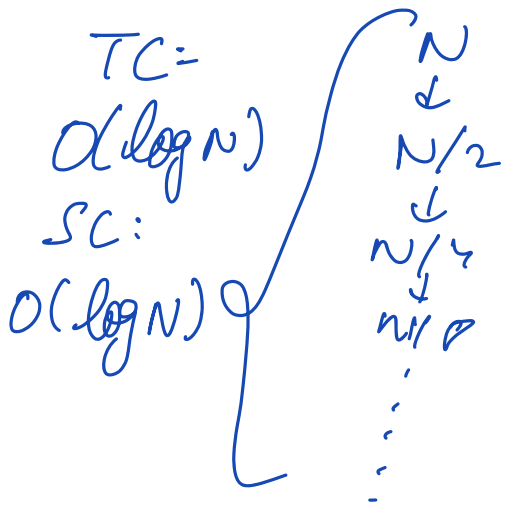
$$= NT(1) + N - 1$$

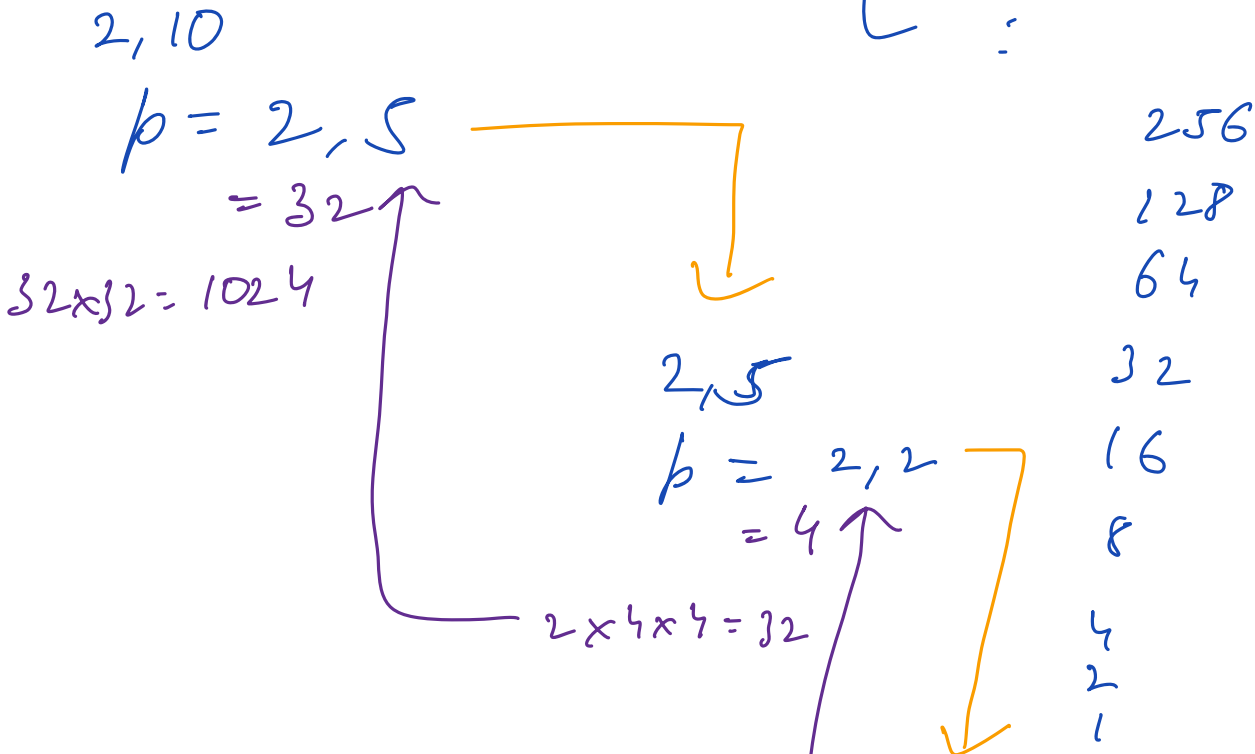
$$= N + N - 1 = 2N - 1 \qquad O(N)$$

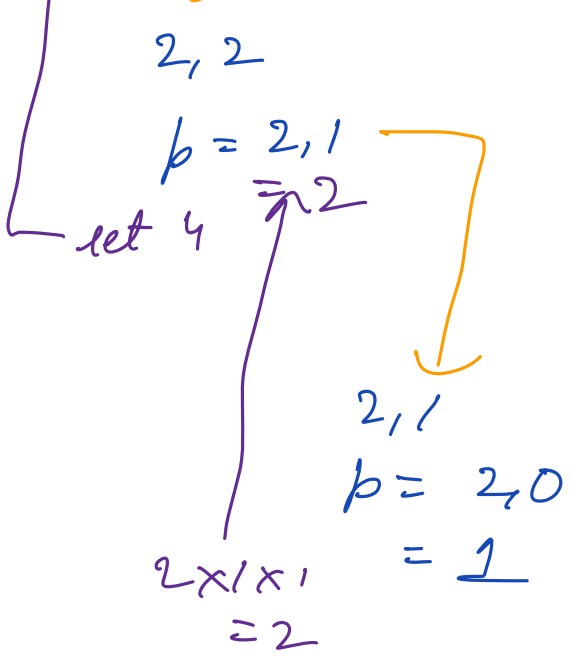
```
int pow(a, n) {                    // Binary Exponentiation
  if (n==0) return 1

  int p = pow(a, n/2)              // p = a^(n/2)
  if (n%2 ==0)
      return p*p
  else
      return a*p*p
}

main() {
  print (pow(2, 10))
}
```

break back at 10:10

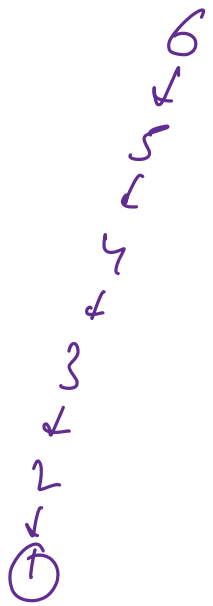
TC: O(log N)
SC: O(log N)

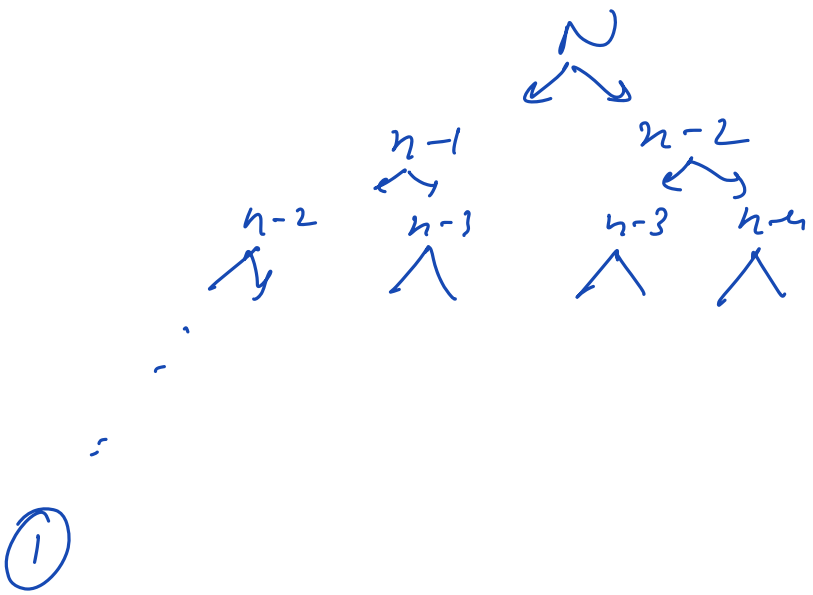







```
int pow (a, n) {
  if (n==0) return 1
  if (a==1) return 1

  int p = pow (a, n/2)
  if (n%2 ==0)
      return pxp
  else
      return pxpxa
}
```





$$1 + 2 + 4 + 8 + \cdots 2^N = 2^{N+1} - 1$$

$$= 2\,2^N - 1$$

$$= O(2^N)$$

int fact(N) {

Calculate factorial of N

$1 \times 2 \times 3 \times 4 \times \ldots \times N$

Assumption: return the factorial of N

$1 \times 2 \times 3 \times 4 \ldots \times N-1 \times N$

Base case: if (n == 1)

return 1

Main Logic: ans = fact(N-1) × N

return ans

}

$f(N) = 1 \times 2 \times 3 \times \ldots N-1 \times N$

$f(n-1)$

f(0)
↑
f(1)
↑
f(2)
↑
f(3)

Recursion tree

N
↓
N-1
↓
N-2
⋮
2
↓
1

TC: O(n)

fact (N) takes T(N)

fact (N-1) takes T(N-1)

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

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

$$T(N) = T(N-2) + 2$$

$$T(N) = T(N-3) + 3$$

$$T(N) = T(N-4) + 4$$

$$\vdots$$

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

$$1 + n-1$$

$$T(N) = n$$

2, 5

↓

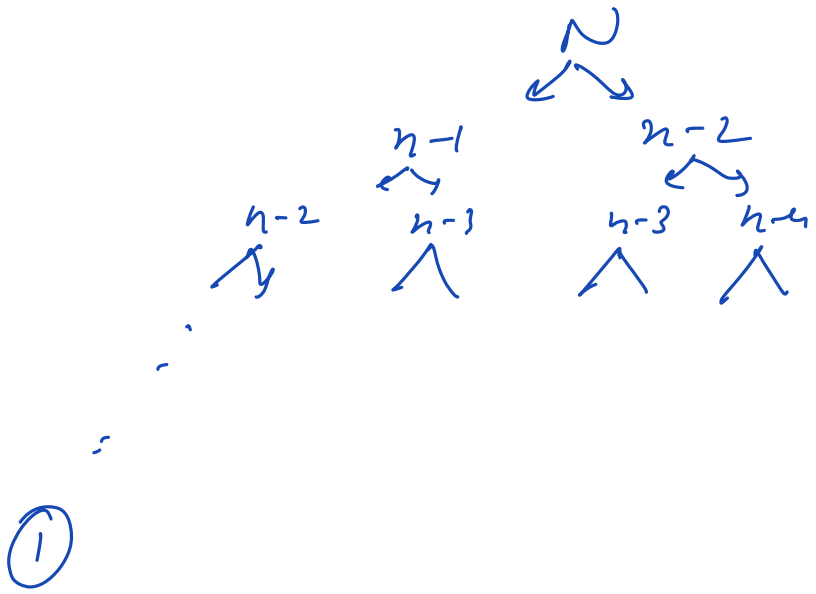
2, 2

↓

2, 1

↓

2, 0



1

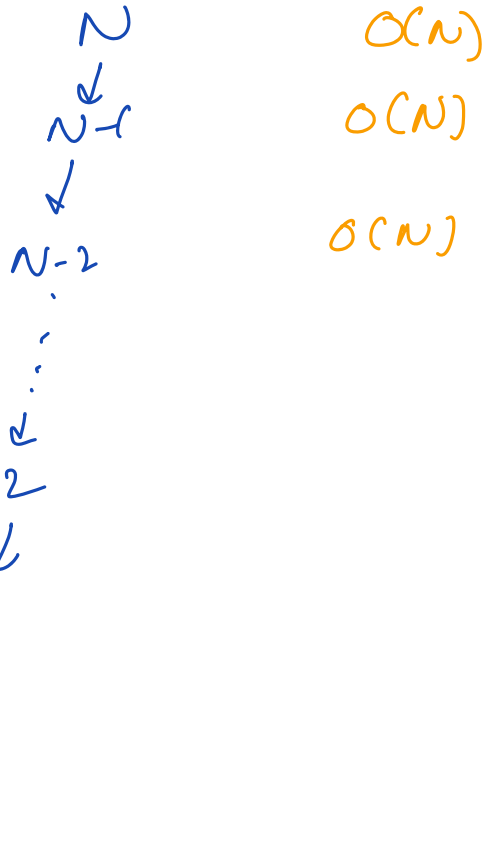
Space complexity using recursion tree

1) Height of the tree

Fib

TC: 2^n

SC: O(n)



print python

cout C++

Sys.out.println Java

console.log JS

∅ 1

Double +1

10110 010001

1 0 0 1 0 0 0 1

10 & 4