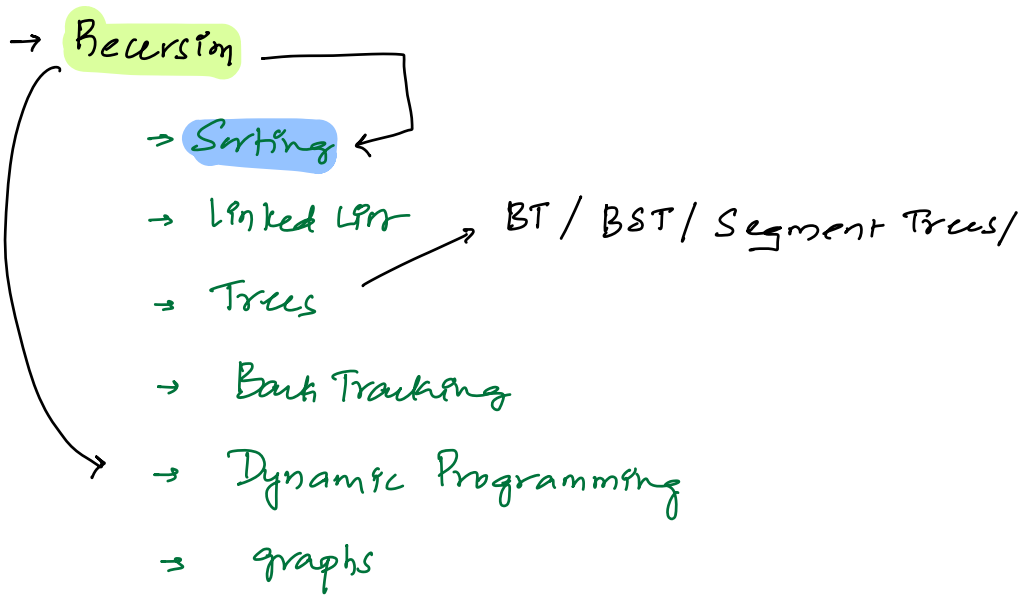

Recursion
Sorting
Linked Lists
Trees
BT / BST / Segment Trees /
Back Tracking
Dynamic Programming
graphs

Recursion: Function calling itself

→ Solving a problem using same instance of same problem

smaller input size

↳ subproblem

→ Solving a problem using its subproblem.

$Sum(N) = 1 + 2 + 3 + \dots + N-1 + N$

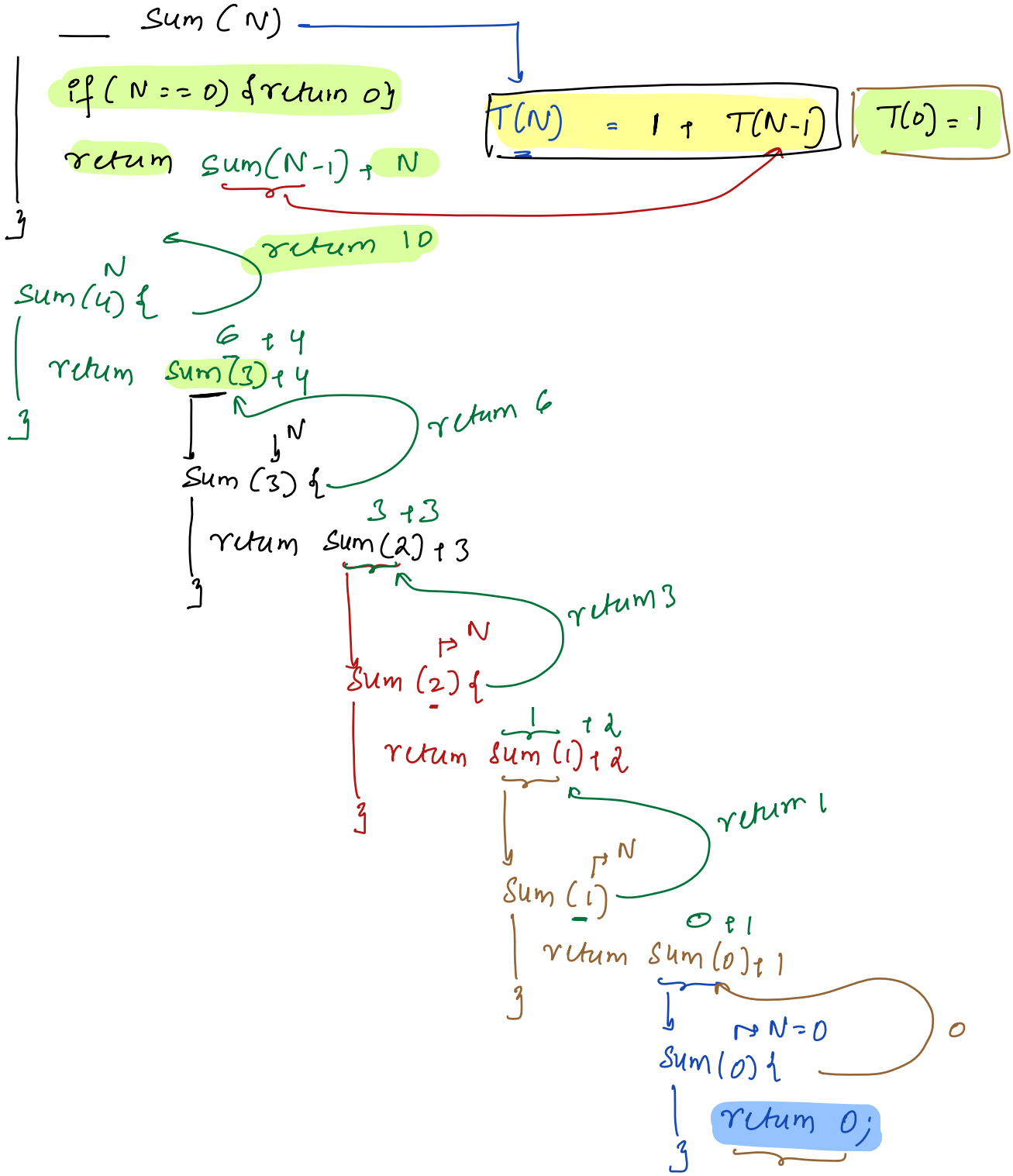
$Sum(N) = Sum(N-1) + N$

Smaller instance of same problem

3 Steps

1) Assumption: Decide what your function does
2) Main logic: Solving assumption using subproblems
3) Base Condition: When to stop.

Ass: Calculate & return Sum of N Natural Numbers



Ass: Calculate & Return N!

```
fact(N)
{
  if(N==0) return 1
  return fact(N-1) * N
}
```

$N! = 1*2*3 \cdots N-1*N$

$= (N-1)! * N$

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

$T(0) = 1$

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

N:	0	1	2	3	4	5	6	7	8	9	..	N-2	N-1	N
fibonacci:	0	1	1	2	3	5	8	13	21	34				

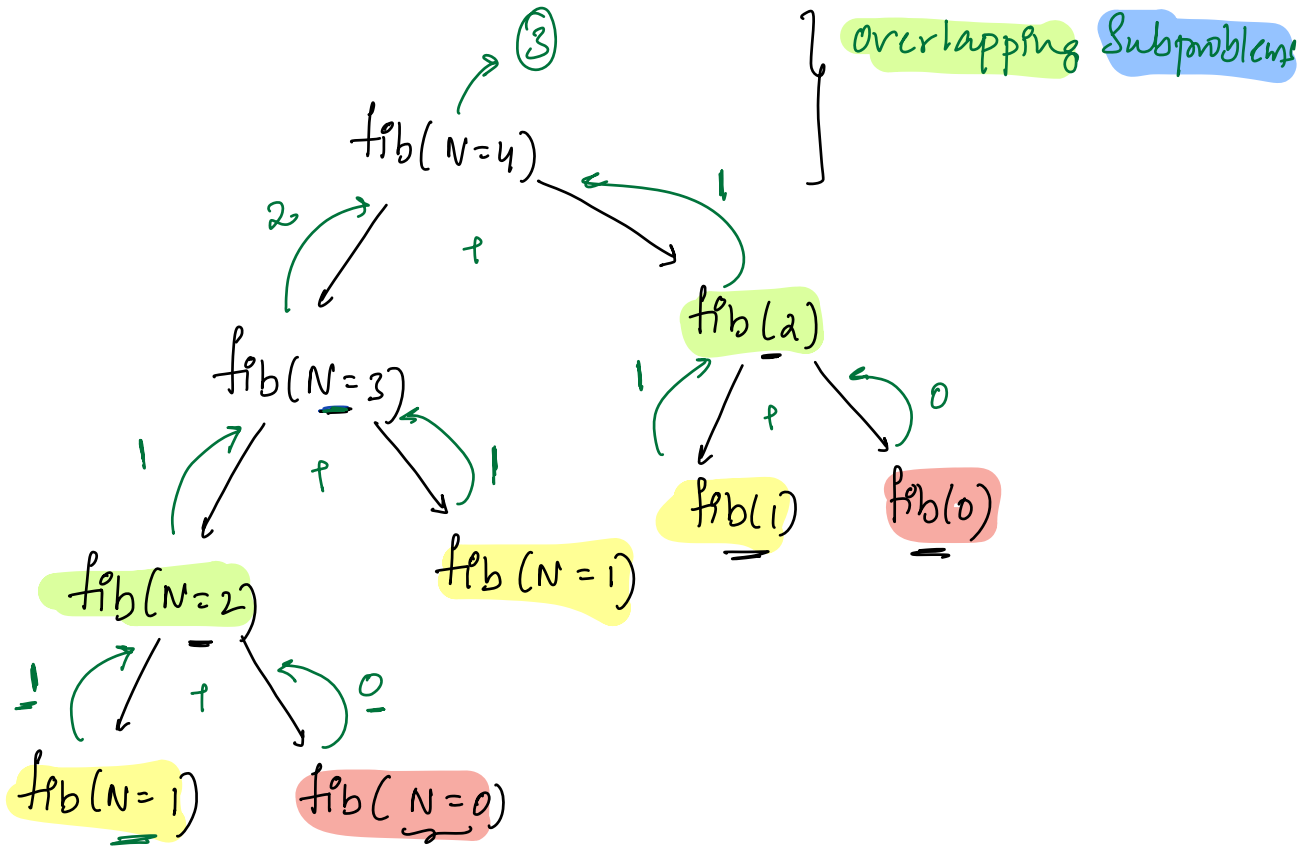
Ass: Calculate & return N^{th} fib number?

```
fib(N) {
  if(N<=1) return N;
  return fib(N-1) + fib(N-2)
}
```

$\Rightarrow$ N=0 : fib(-1) + fib(-2) ✗

N=1 : fib(0) + fib(-1) ✗

N=2 : fib(1) + fib(0)


3
fib(N=4)
overlapping
Subproblems
2
+
1
fib(N=3)
fib(2)
1
+
1
+
0
fib(N=2)
fib(N=1)
fib(1)
fib(0)
1
+
0
fib(N=1)
fib(N=0)

Ass: Calculate & return $a^n \% p$ ⇒ TC: O(N)

```
power(a, n, p)
{
    if(n==0) { return 1; }
    return (power(a, n-1, p) * a) % p;
}
```

Subproblems

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

$a^N \% p = (a^{N-1} \% p * a \% p) \% p$

$a^{10} = a * a^9$

$a^{10} = a^5 * a^5$

$a^{11} = a^5 * a^5 * a^1$

$a^{12} = a^6 * a^6$

$a^{15} = a^7 * a^7 * a$

$a^n \% p$

if (n is even) → power(a, n/2, p)

$((a^{n/2}) \% p * (a^{n/2}) \% p) \% p$

else

$(((a^{n/2} \% p * a^{n/2} \% p) \% p * a \% p) \% p$

// Constraints: [1, 10^9]

$p = 10^9$

$a \% p * b \% p$

$[0, p-1]$ $[0, p-1]$ $= p^2 \rightarrow 10^{18} \rightarrow$ long

↳ Cannot store in long.

$a \% p * b \% p * c \% p = p^3 = 10^{27} \rightarrow$ overflow

$[0, p-1]$ $[0, p-1]$ $[0, p-1]$

$(a \% p * b \% p) \% p * c \% p = p^2 = 10^{18}$

$a \% p \rightarrow 10^{18}$

$[0, p-1]$ $[0, p-1]$

// 2c

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

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

$T(0) = 1$

$T(1) = 1$

TC: $O(N)$

SC: → O(log N)

int / long

power(a, n, p) { → fast exponentiation.

```
if(n==0) return 1;
long y = power(a, n/2, p);
if(N%2==0){
    return (y*y)%p;
}
else
    return ((y*y)%p * a%p)%p;
}
```

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

$T(1) = 1$

$\mapsto$ a, n >= 0, p = 10^9

TC: ⇒

Code:

Recursive Relations:

power1: $T(N) = T(N-1) + 1 \quad T(0) = 1, \quad T(N) = O(N)$

power2: $T(N) = 2T(N/2) + 1 \quad T(1) = 1, \quad T(N) = O(N)$

power3: $T(N) = T(N/2) + 1 \quad T(1) = 1 \quad T(N) = O(\log N)$

// $T(N) = T(N-1) + 1$ → known Term

$T(0) = 1$

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

$= T(N-2) + 2$

$T(N-2) = T(N-3) + 1$

$= T(N-3) + 3$

$T(N-3) = T(N-4) + 1$

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

// generalized expression

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

// Substitute $k = N$ → SC: $O(k)$, $k = N$

$= T(0) + N$

SC: $O(N)$

$= 1 + N$

$T(N) = N + 1$

$T(N) = O(N)$ SC: $O(N)$

28)

$$T(N) = 2T(N/2) + 1, \quad T(0) = 1$$

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

$$= 2\left[2T(N/4) + 1\right] + 1$$

$$= 4T(N/4) + 3 = 2^2 T(N/2^2) + 2^2 - 1$$

$T(N/4) = 2T(N/8) + 1$

$$= 4\left[2T(N/8) + 1\right] + 3$$

$$= 8T(N/8) + 7 \Rightarrow 2^3 T(N/2^3) + 2^3 - 1$$

$T(N/8) = 2T(N/16) + 1$

$$= 8\left[2T(N/16) + 1\right] + 7$$

$$= 16T(N/16) + 15 \Rightarrow 2^4 T(N/2^4) + 2^4 - 1$$

// general expression

$T(0) = 1$, $T(1) = 1$

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

$N/2^k = 1$ } $2^k = N$

$N = 2^k$ }

// Apply $\log_2$ on both sides

$k = \log_2 N$

$2^{\log_2 N} = N$

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

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

$$= 2N - 1$$

$T(N) = O(N)$, SC: $O(k) = O(\log_2 N)$

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

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

$= T(N/4) + 2 \Rightarrow T(N/2^2) + 2$

$T(N/4) = T(N/8) + 1$

$= T(N/8) + 3 \Rightarrow T(N/2^3) + 3$

$T(N/8) = T(N/16) + 1$

$= T(N/16) + 4 \Rightarrow T(N/2^4) + 4$

// generalized expres

$T(N) = T(N/2^k) + k$ $\quad T(1) = 1$

$k = \log_2^N$

$T(N) = T(1) + \log_2^N$

$T(N) = O(\log_2^N)$ $\quad SC: O(\log_2^N)$

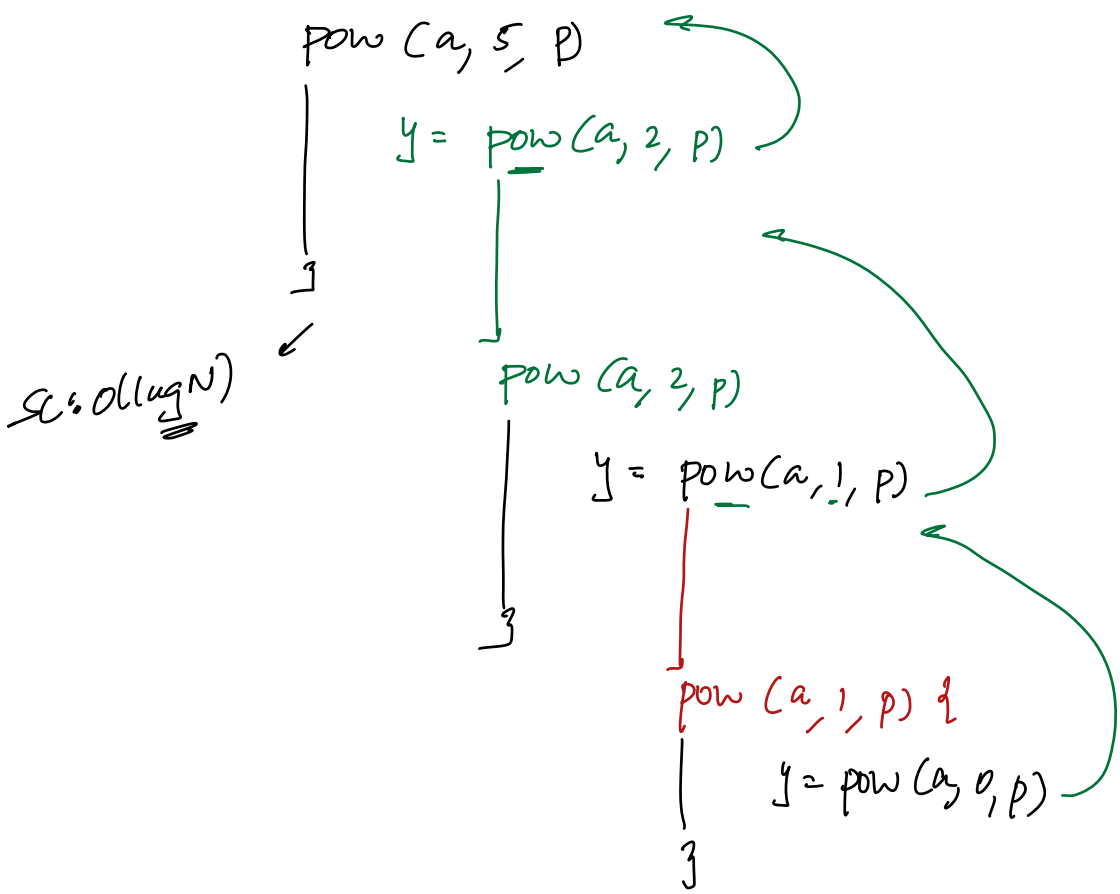
TODO:

- $T(N) = 2T(N-1) + O(1)$ ✓
- $T(N) = 2T(N/2) + O(N)$ ✓
- $T(N) = T(N-1) + T(N-2) + O(1)$ ✓

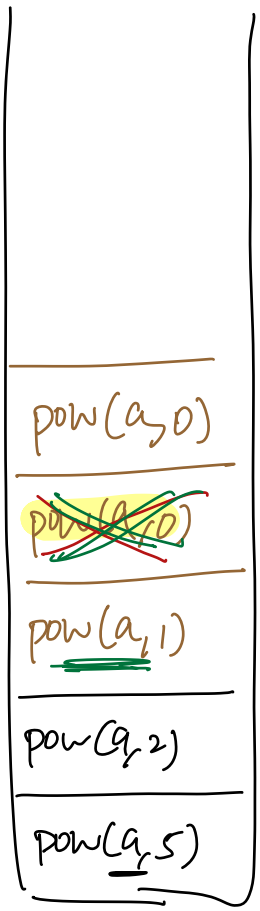
$T(1) = 1$

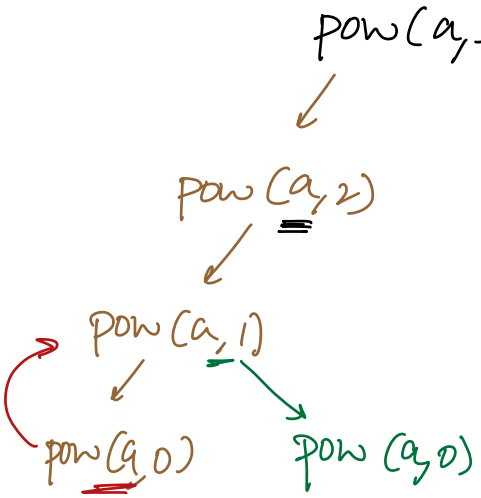
// Doubts:

1) Why is Space Complexity : $O(N)$?



2nd power





//

// Doubts → code(N) → { F(N) = F(N/2) + }

code(N/2)

We are trying func(N)