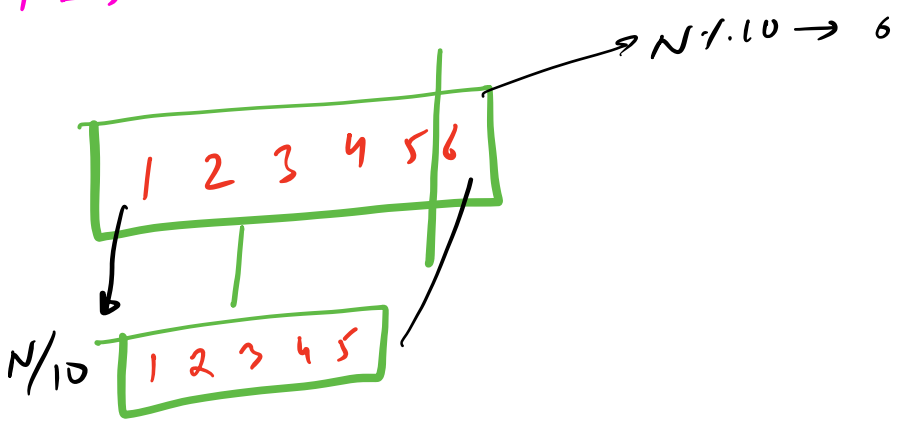
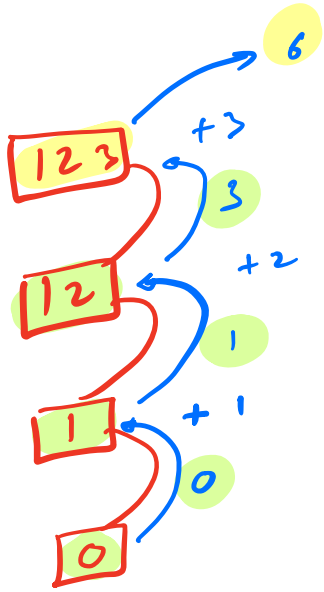
Q. Given N. find the sum of digits of N!

N: 1 2 3 4 5 6 → 21




```
int sod (N) {
// Ass: ret the sum of digits of N
   if (N == 0) ret 0;
   ret sod(N/10) + (N%.10);

}
```



Q Given a, N. Calculate a^N.

N ≥ 0

a: 2, N: 5 → 2^5 : 32

a: 3, N: 5 → 3^5 : 243

a: 7, N: 5 → 7^5 : 16807

a^N = a x a x a x ——— a

N times

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

RF

$pow(a, N) = pow(a, N-1) \times a$

```
int pow(a, N) {
// Ass: ret a^N
    if(N==0) ret 1;
    ret pow(a, N-1) x a;
}
```

if(N==1) ret a;

pow(2, 0)

↓

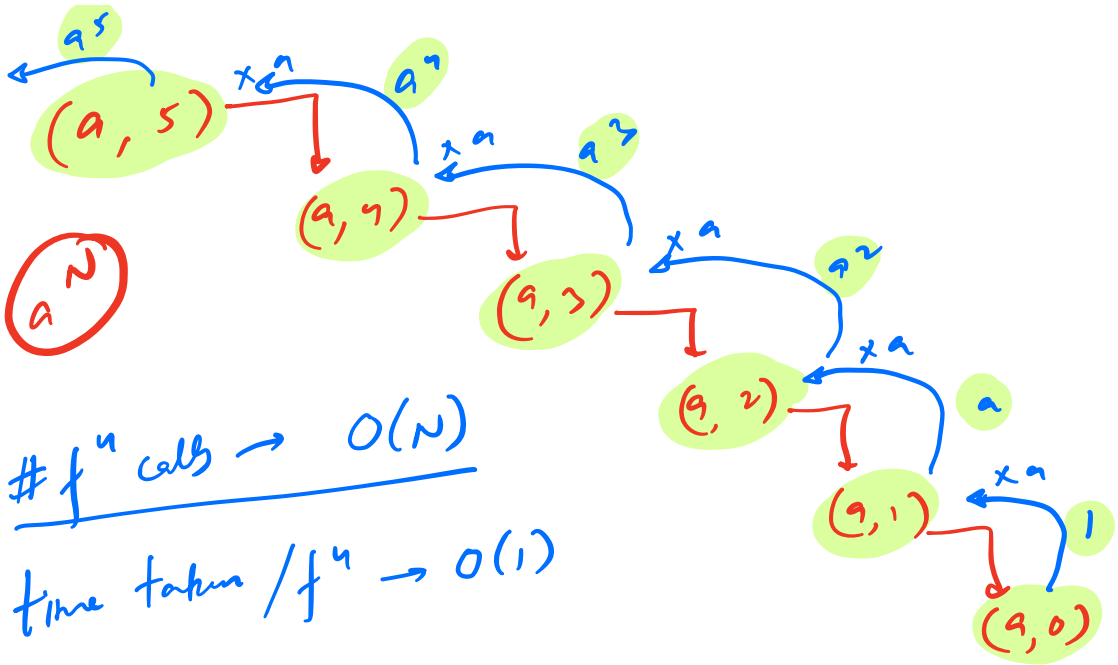
pow(2, -1)

↓

pow(2, -2)

↓

Stack Overflow!



f^n calls → O(N)

time taken / f^n → O(1)

TC = O(N)

TC = # f^n calls × (time taken / f^n call)

II

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

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

$a^{20} = a^{10} \times a^{10}$

$a^5 = a^2 \times a^2 \times a$

$a^{17} = a^8 \times a^8 \times a$

$a^{15} = a^7 \times a^7 \times a$

N: EVEN

$$a^N = a^{N/2} \times a^{N/2}$$

N: ODD

$$a^N = a^{N/2} \times a^{N/2} \times a$$

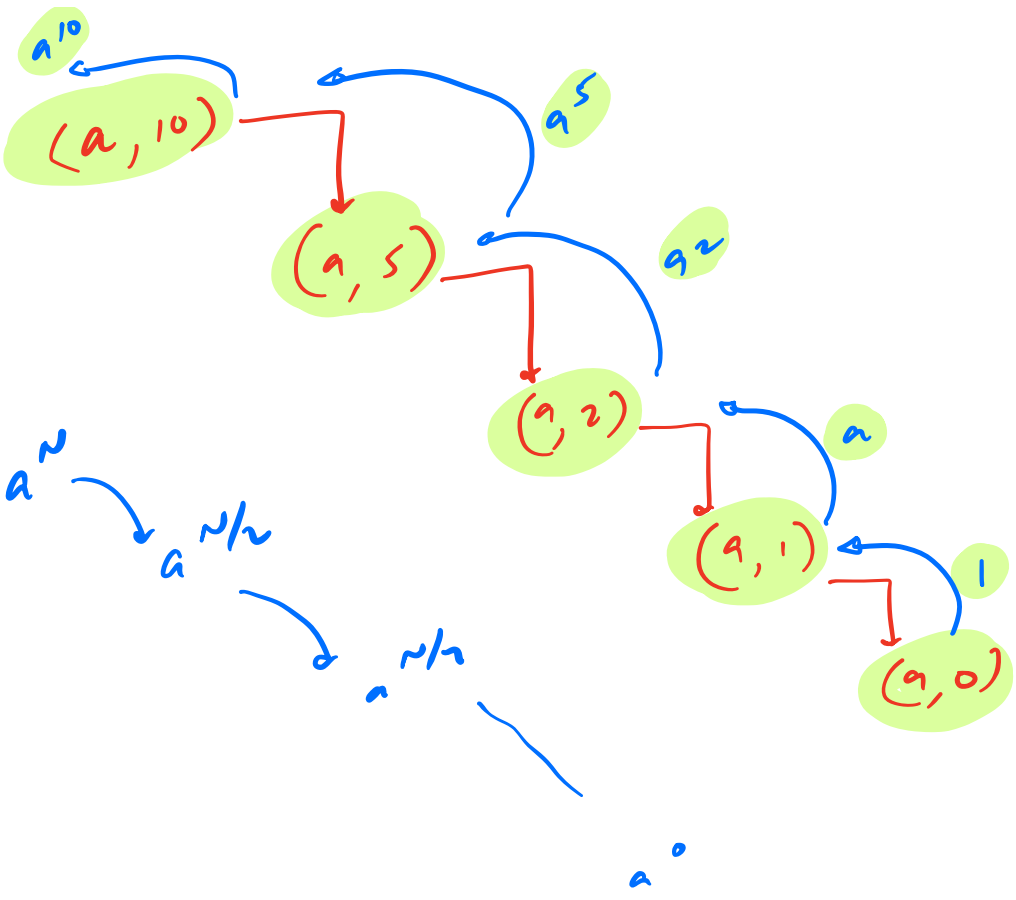
fast power exponentiation

```
int pow(a, N) {
  // Ass: ret a^N
    if (N==0) ret 1;
    hp = pow(a, N/2);
    if (N%2==0) {
        ret hp * hp;
    } else {
        ret hp * hp * a;
    }
}
```

$M = 10^9+7$

$< M$

```
int pow(a, N, M) {
  // Ass: ret a^N %. M
    if (N==0) ret 1;
    hp = pow(a, N/2, M);
    if (N%2==0) {
        ret ((hp %. M) * (hp %. M)) %. M;
    } else {
        ret ((hp * hp) %. M * a) %. M
    }
}
```



f^n calls $\rightarrow O(\log N)$

time taken / f^n call $\rightarrow O(1)$

TC = # f^n calls $\times$ time taken / f^n call

$\log(N) \times O(1)$

TC = $O(\log N)$

N	vs	$\log_2 N$
10^9		$\log_2(10^9)$
$\downarrow$		$\downarrow$
10^9		~30

$a^N \rightarrow a^{N/2}$

$T(N) = T(N/2) + 1$ → ①

put $N = N/2$ in ①

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

substitute this val in ①

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

$T(N) = T(N/4) + 2$ → ②

put $N = N/4$ in ①

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

substitute in ②

$T(N) = T(N/8) + 3$

$T(N) = T(N/16) + 4$

$T(1) = 1$

General form

$T(N) = T(N/2^K) + K$

$N/2^K = 1$

$N = 2^K$

$K = \log N$

$T(N) = T(1) + \log N$

$T(N) = 1 + \log N$

$T(N) = \log(N)$

9

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

$N \to N/2$

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

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

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

$N \to N/4$

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

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

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

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

$N/2^k = 1$

$N = 2^k$

$k = \lg N$

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

$N \times 1 + N - 1 = 2N - 1$

$T(N) = 2N - 1$

$TC = O(N)$

$$T(N) = 2T(N-1) + 1 \qquad T(0) = 1$$

$N \to N-1$

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

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

$$T(N) = 4T(N-2) + 3$$

$N \to N-2$

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

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

$$T(N) = 8T(N-3) + 7$$

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

$N - K = 0$

$N = K$

$K = N$

$$T(N) = 2^N T(0) + 2^N - 1$$

$$2^N \times 1 + 2^N - 1$$

$$2 \cdot 2^N - 1$$

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

$$TC = O(2^N)$$

⑧ Fibonacci 1

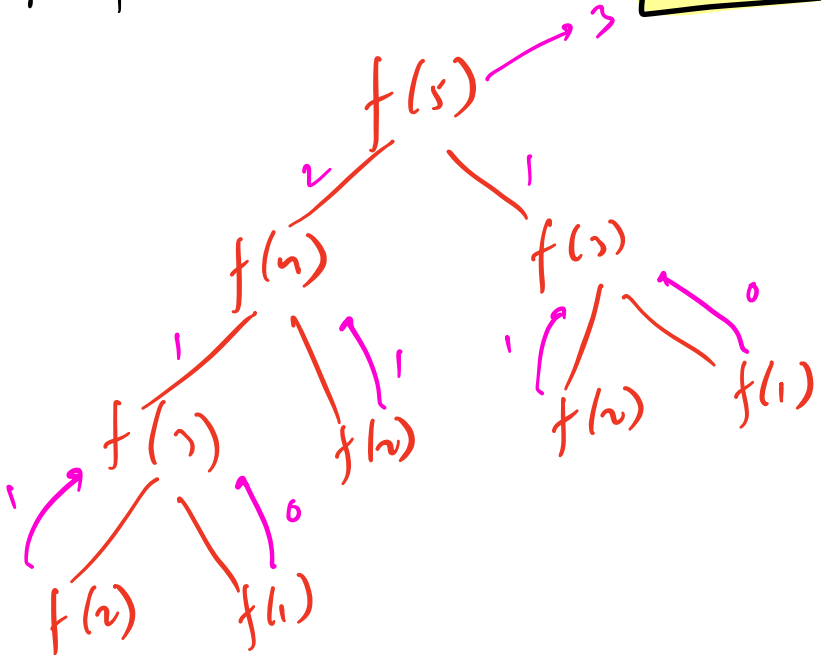
```
int f(N) {
  if( N<= 2) ret (N-1);
  ret f(N-1) + f(N-2);
}
```

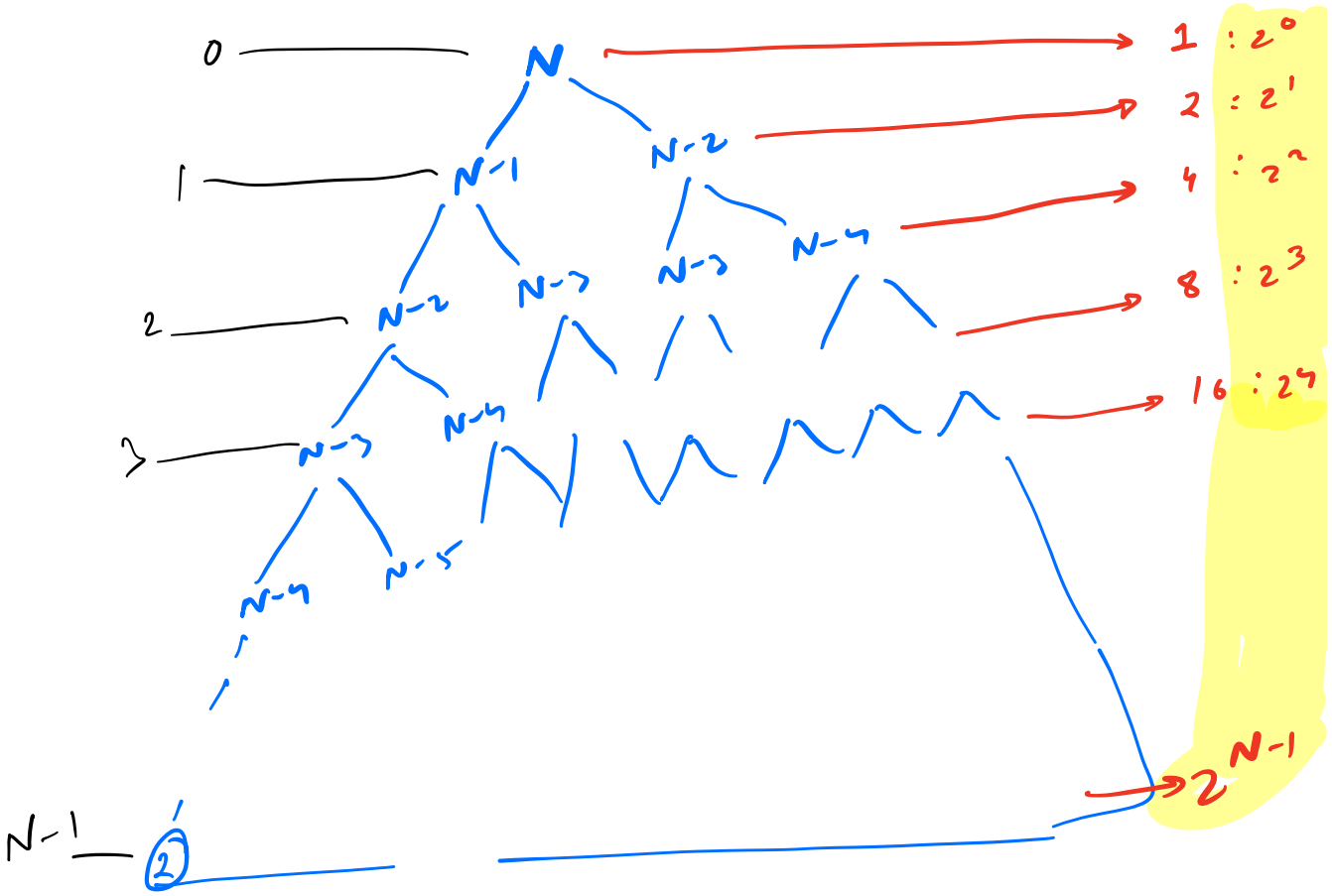
$T(N) = T(N-1) + T(N-2) + 1$

$\downarrow$

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

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





Total # f^n calls

$= 2^0 + 2^1 + 2^2 + \cdots + 2^{N-1}$

$= 2^N - 1$

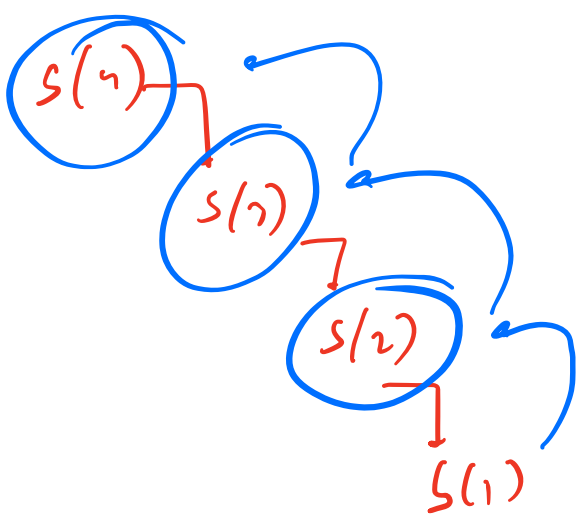
$= O(2^N)$

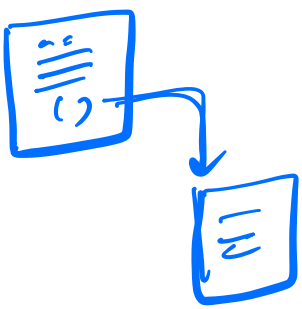
time taken / f^n call $\rightarrow O(1)$

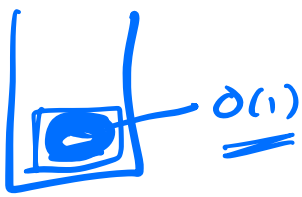
$TC = 2^N \times O(1)$

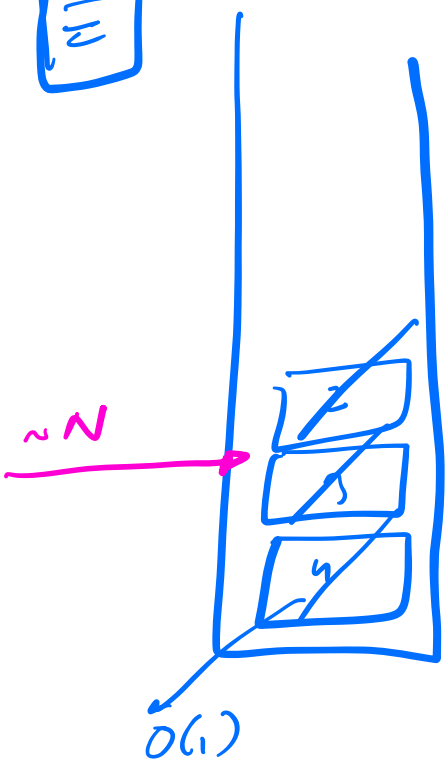
$TC = O(2^N)$

```
int s(N) {
    if(N==1) ret 1;
    ret N + s(N-1);
}
```



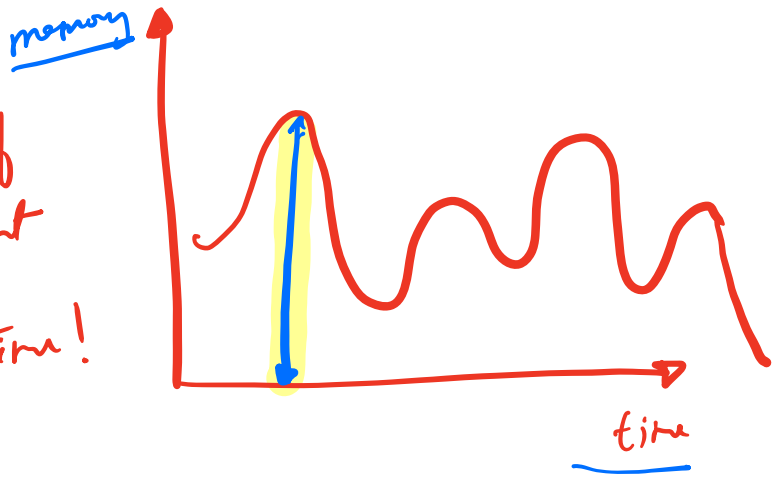




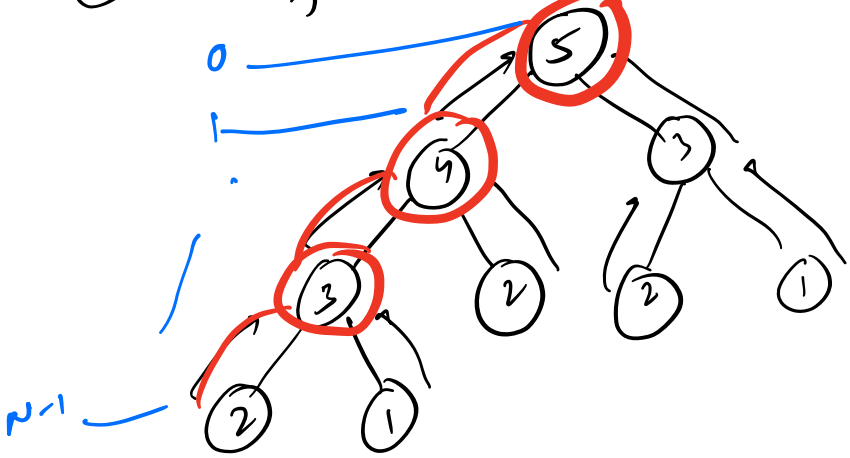


$SC = O(N)$

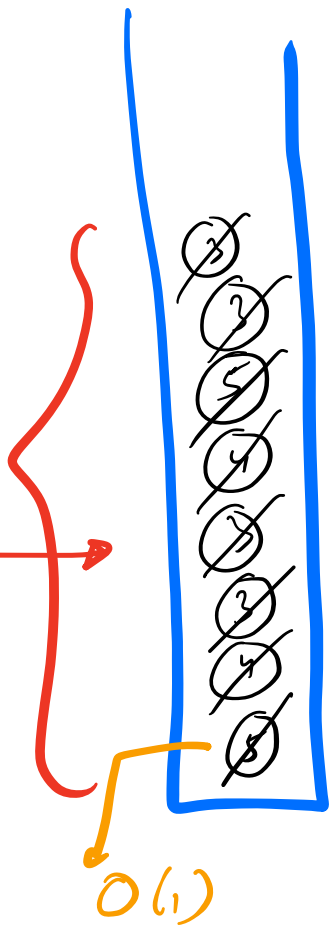
SC = MAX AMT. of space taken at ANY point in time!



④ fibonacci



MAX # Active f^n calls in stack = N



SC = O(N)

O(1)

SC = MAX # Actual f^n calls in stack
× Space taken by 1 f^n