Recursion - 2

Yuval Noah Harari

"When the Spaniards first arrived in Mexico, natives bearing incense burners were assigned to accompany them wherever they went. The Spaniards thought it was a mark of divine honour. We know from native sources that they found the newcomers' smell unbearable.)"

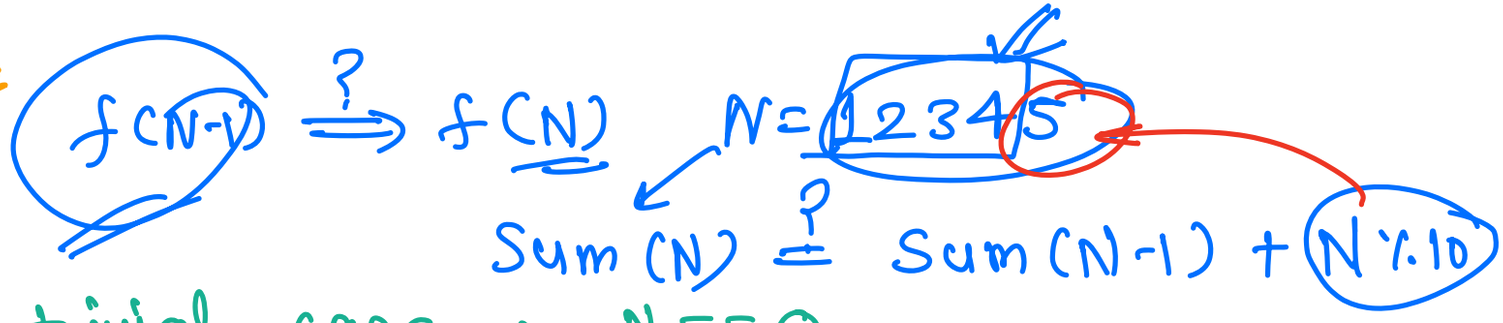
— Yuval Noah Harari, Sapiens: A Brief History of Humankind

Music: "How you want to leave" from TVF Pitchers
by Vaibhav Bundhoo

Q. Sum of digits: Given a num $N \geq 0$, find sum of digits using recursion

ex: $N = 123 \Rightarrow 6$ | $N = 47953$, ans = 28

idea



trivial case $\Rightarrow$ N == 0

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

midnight

Q. Given a, n find a^n using recursion. ($N \geq 0$)

ex:

a	n	a^n
3	4	81
2	5	32

Brute force
① using for loop

idea:

(2)

a^n = $\overbrace{a \times a * a \times .. \times a}^{n-1 \text{ items}} \times a$

$\underbrace{\qquad\qquad\qquad\qquad}_{n \text{ times}}$

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

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

trivial case, n==0 ⇒ 1 / n==1 ⇒ a

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

(3rd)

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

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

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

if n is even

$$a^n = a^{n/2} * a^{n/2}$$

pow(a,n) = pow(a,n/2) * pow(n/2)

else

$$a^n = a^{n/2} * a^{n/2} * a$$

pow(a, n) = pow(a,n/2) * pow(a,n/2) * a

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

  if (n is even) {
    return [pow(a,n/2) * pow(a,n/2)]

  else
    return pow(a,n/2) * pow(a, n/2) * a

}
```

Obsⁿ: calculate pow(a, n/2), & then reuse it

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

  int p= pow(a, n/2)

 if (n is even) {
    return p*p

 else
    return p*p*a;

}
```

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

$p * p * a$

Dry run for $a = 2, n = 9$, $\Rightarrow$ ans = 512

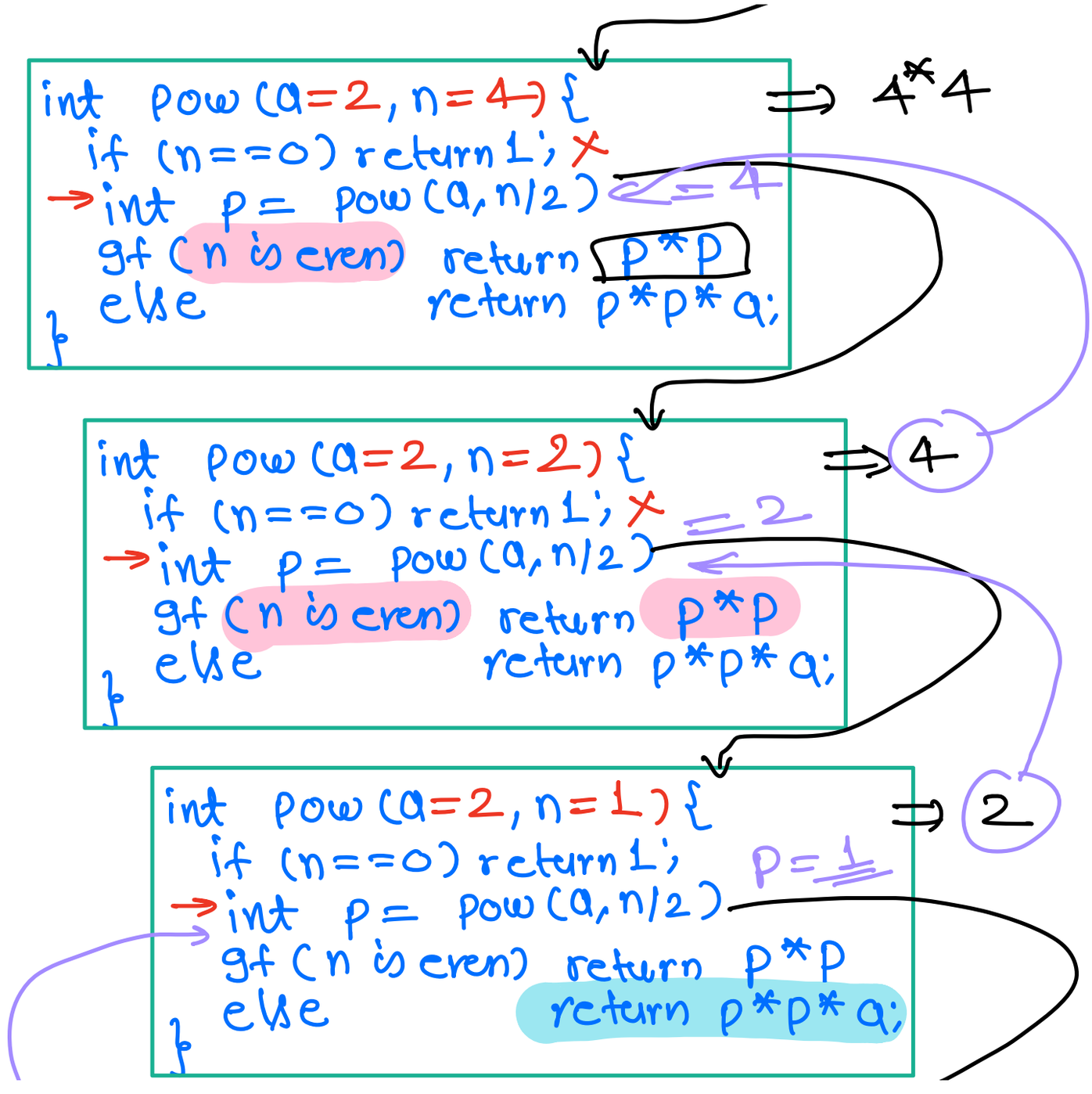
```
int pow(a=2, n=9){
  if (n==0) return 1;
→int p= pow(a, n/2)
  if (n is even) return p*p
  else           return p*p*a;
}
```

p = 16

(n/2 == 0)

(n == 1)

16 * 16 * 2 ⇒ 512


int pow(a=2, n=4) {
if (n==0) return 1;
int p = pow(a, n/2)
gf (n is even) return p*p
else return p*p*a;
}
⇒ 4*4
int pow(a=2, n=2) {
if (n==0) return 1;
int p = pow(a, n/2)
gf (n is even) return p*p
else return p*p*a;
}
⇒ 4
int pow(a=2, n=1) {
if (n==0) return 1;
int p = pow(a, n/2)
gf (n is even) return p*p
else return p*p*a;
}
⇒ 2
p=1

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

Q. Given a, n, m calculate $a^n \% m$

(MOD = $1000*1000*1000+2$)

Constraints

$1 \le a \le 10^9$

$1 \le n \le 10^9$

$2 \le m \le 10^9$

$a \approx 10^9,\ n \approx 10^9,\ m \approx 10^9$

```
long powmod(a, n, m)
  if (n==0) return 1;
  long p= powmod(a, n/2, m)   // a^(n/2) % m
  if (n is even) {
     return (p*p) % m
```

$p \in (0, M-1)$

$[0 \sim 10^9]$

$10^9 * 10^9 \Rightarrow (10^{18})$

```
    else
        return (p*p*a) %m
}
```

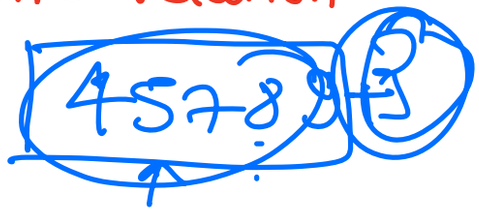
$10^9 * 10^9 * 10^9 \approx 10^{27}$

```
    return ((p*p %m) * a) %m
```

```
int powmod (a, n, m)
    if (n==0) return 1;
        p = powmod(a, n/2, m)   // a^(n/2) %m   <- int
    if (n is even) {
        return 1L * p*p % m
    else
        return ((1L * p*p) * a) % m
}
```

9. Time complexity for recursive functions using recursive relation

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



f(n-1)

time taken to calculate sum(N) = f(n)

$f(n) = f(n-1) + 1$

$= f(n-2) + 1 + 1$

$= f(n-3) + 1 + 1 + 1$

$f(n-1) = f(n-2) + 1$

$f(0) = 1$

$f(1) = 1 + 1$

$f(2) = f(1) + 1$

$$f(n) = f(n-k) + k$$

$\Rightarrow$ when $k = n$ we reach base cond^n.

$\Rightarrow f(n-n) + n$

$f(n) = n \Rightarrow$ T.C. $= O(N)$

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

N ⇒ 1+2+3+...N

Sum of N natural num

Sum(N) = 1+2+3+..N-1+N

$f(n)$ = $f(n-1)$

$f(n) = f(n-1) + 1$

$\Rightarrow f(n) = n$

$\Rightarrow$ T.C. $= O(n)$

```
int pow(a,n){
    if(n==0) return 1;
    return pow(a,n-1)*a;   // one more operation is mul
}
```

}

required.

$$f(n) = f(n-1) + 1$$

$$\Rightarrow f(n) = n \Rightarrow \text{T.C.} = O(N)$$

§. T.C.

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

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

T.C. = O(N)

$$f(n) = 2f(n/2) + 1 \Rightarrow 2^1 f(n/2^1) + 2^1 - 1$$

$$f(n/2) = 2f(n/4) + 1$$

$$= 2[2f(n/4) + 1]$$

$$=$$

$= 4f(N/4) + 3 \Rightarrow 2^2 f(N/2^2) + 2^2 - 1$

$f(n/4) = 2f(n/8) + 1$

$= 4 * [2 * f(n/8) + 1) + 3$

$= 8 * f(n/8) + 7$

$f(n/8) = 2 * f(n/16) + 1$

$= 8 * (2 * f(n/16) + 1) + 7$

$f(n) = 16 * f(n/16) + 15$

$f(n) = 2^4 * f(n/2^4) + (2^4 - 1)$

$f(n) = 2^k * f(n/2^k) + (2^k - 1)$

Base condition $\Rightarrow \frac{n}{2^k} = 0 \Rightarrow n = 0 \Rightarrow f(0)$

$\Rightarrow f(1) = 1 \Rightarrow \frac{n}{2^k} = 1$

$\Rightarrow \quad n = 2^x \quad \Rightarrow \quad x = \log_2 n$ — (1)

$\Rightarrow f(n) = n * f(n/n) + n - 1$

$= n * f(1) + n - 1$ ($f(1) \to 1$)

$= n + n - 1 = 2n - 1$

$f(n) = 2n - 1$

$\Rightarrow$ T.C. $= O(n)$

Q.

$f(n)$

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

$= f(n/2^1) + 1$

$$f(n) = f(n/2) + 1 \quad , \quad f(0) = 1, f(1) = 1$$

$f(n/2) = f(n/4) + 1$

$$= f(n/4) + 2 \quad = f(n/2^2) + 2$$

$$= f(n/8) + 3 \quad = f(n/2^3) + 3$$

$$f(n) = f(n/2^k) + k$$

$\Rightarrow \frac{n}{2^k} = 1$ we reach base condition

$\Rightarrow n = 2^k \Rightarrow k = \log_2 N$

$\Rightarrow$

$$f(n) = f(n/n) + \log_2 N$$

$$= f(1) + \log_2 N$$

$$\Rightarrow f(n) = \log_2 N + 1$$

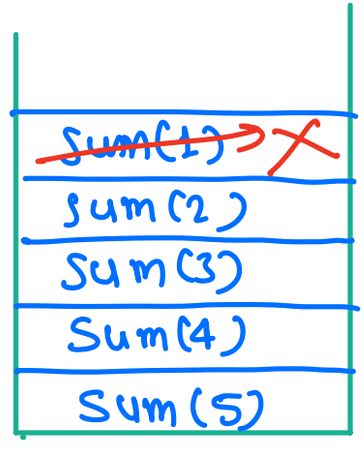
$$T.C. = O(\log_2 N)$$

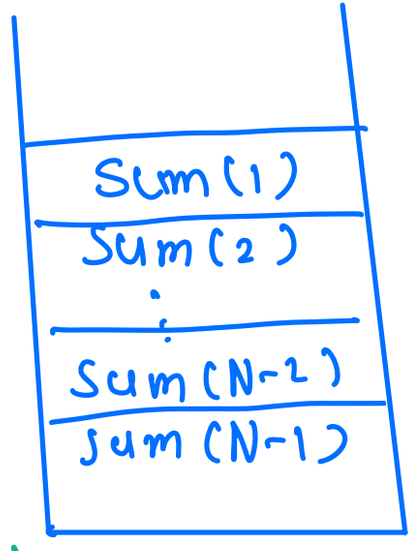
5. Space complexity for recursive functions

Obs^n: we need space to keep track of recursive function calls

S.C. ⇒ stack size

```
int sum(N) {          // f(n)
    if (N==1) return 1;
    return sum(N-1) + N;   // f(n-1)
}
```



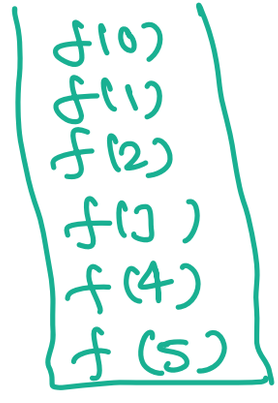


Sum(5)

S.C. = O(N)

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

S.C. = O(N)



n+1

Error: stack overflow

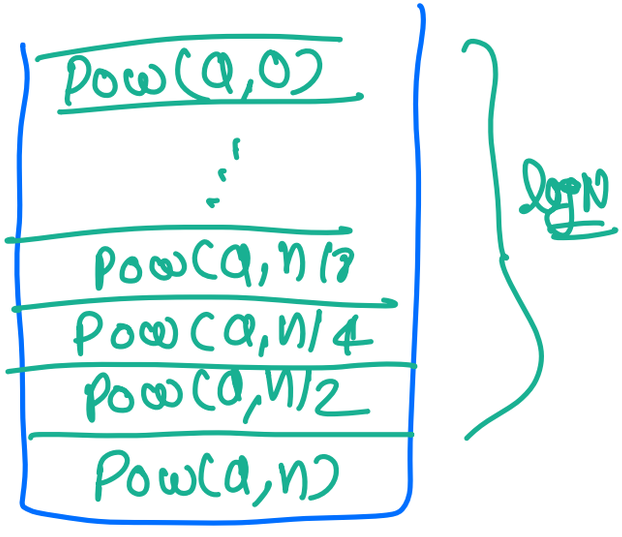
```
int pow(a,n){
    if(n==0) return 1;
    return pow(a,n-1)*a;
}
```

S.C. = O(n)

Q.

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

S.C. = O(log N)



Q. space complexity

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

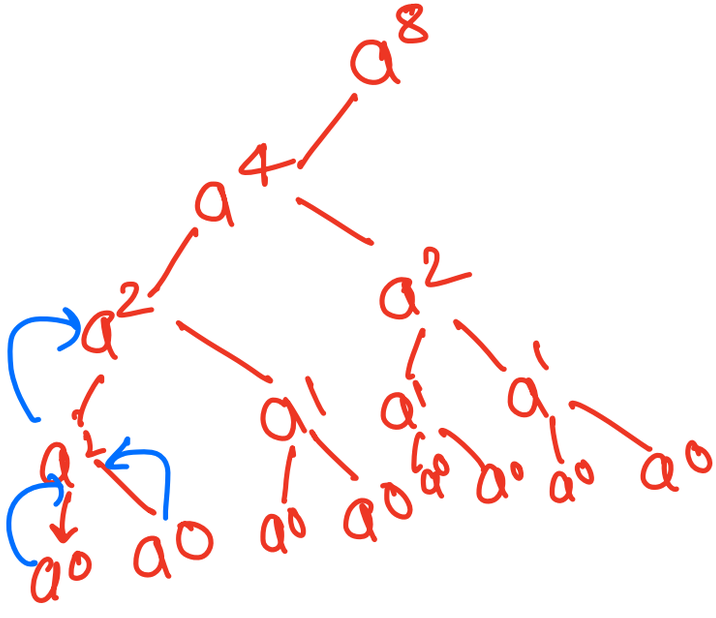
T.C. = O(N)

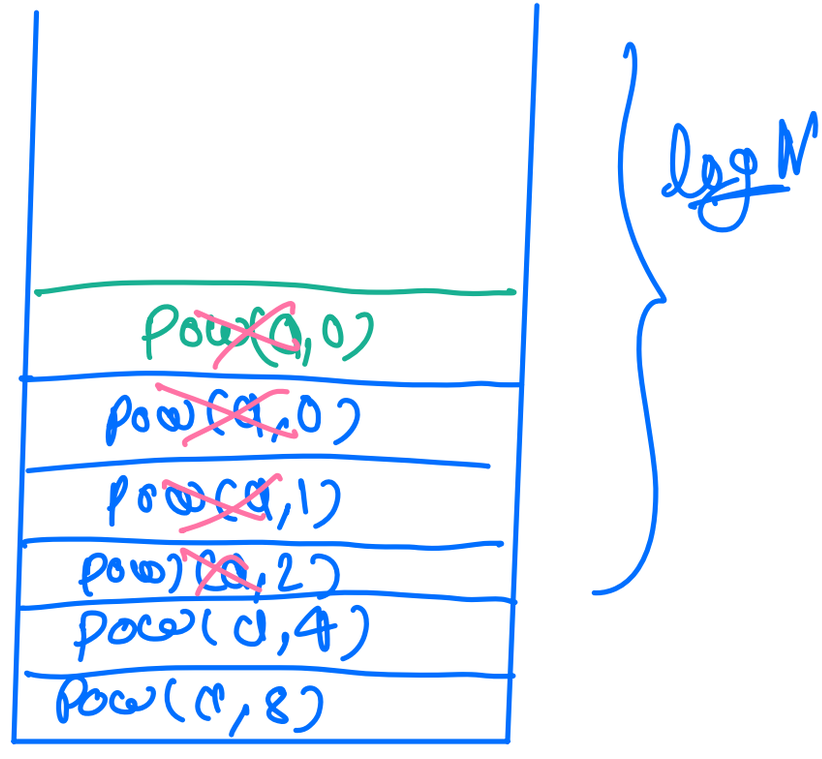
```
if (n is even) {
    return [pow(a, n/2) * pow(a, n/2)]
else
    return pow(a, n/2) * pow(a, n/2) * a
}
```

?

10:48

pow(a, n) → pow(a, n/2)





9. fibonacci (Time complexity)

```
int fib (n) {
    if (n==0 or n==1) return n;
    return fib(n-1) + fib(n-2);
}
```

T.C =

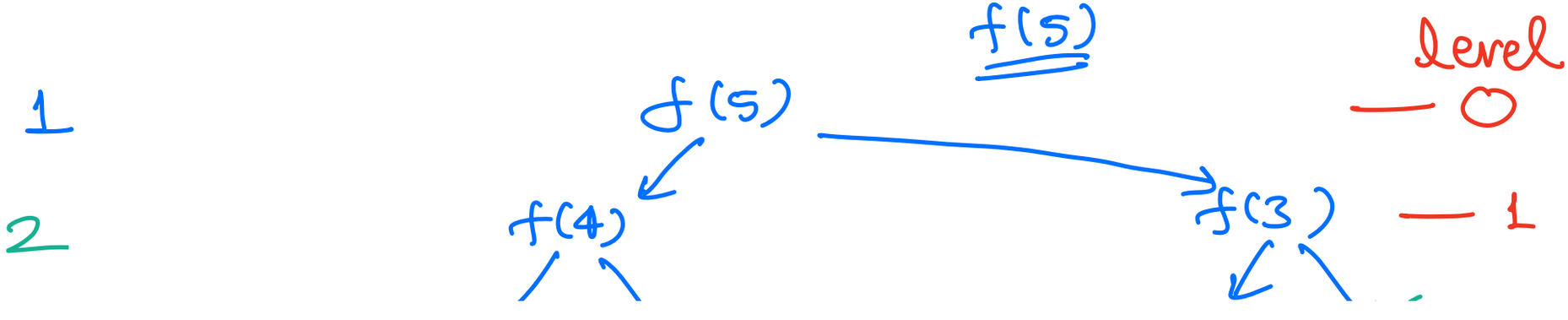
$$f(N) = f(N-1) + f(N-2) + 1$$

$$f(N-2) + f(N-3) + 1 + (N-2)$$

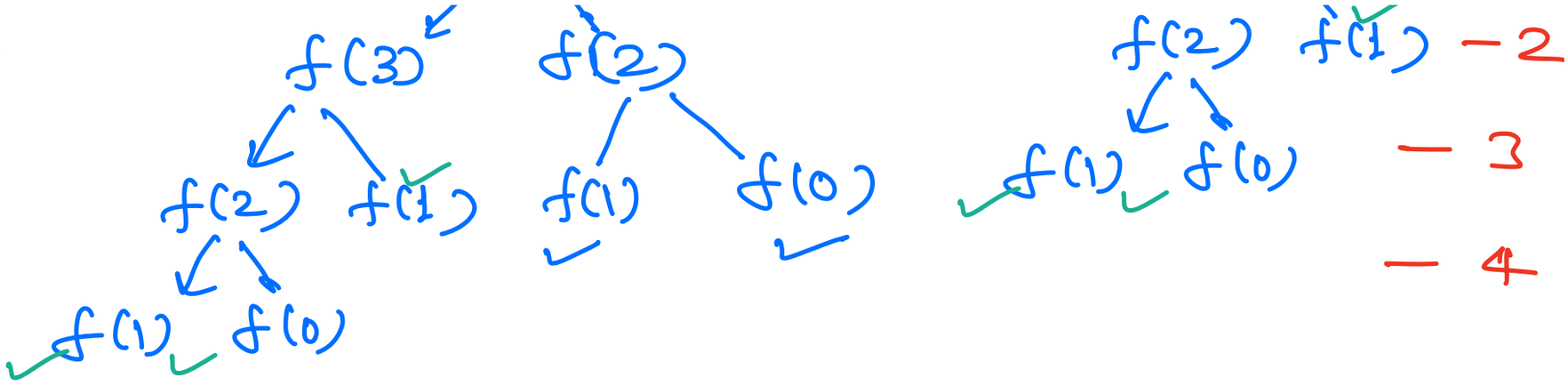
$$1 + fib(N-3) + fib(N-4)$$

$$\Rightarrow f(N-2) + 2f(N-3) + f(n-4) + 3$$

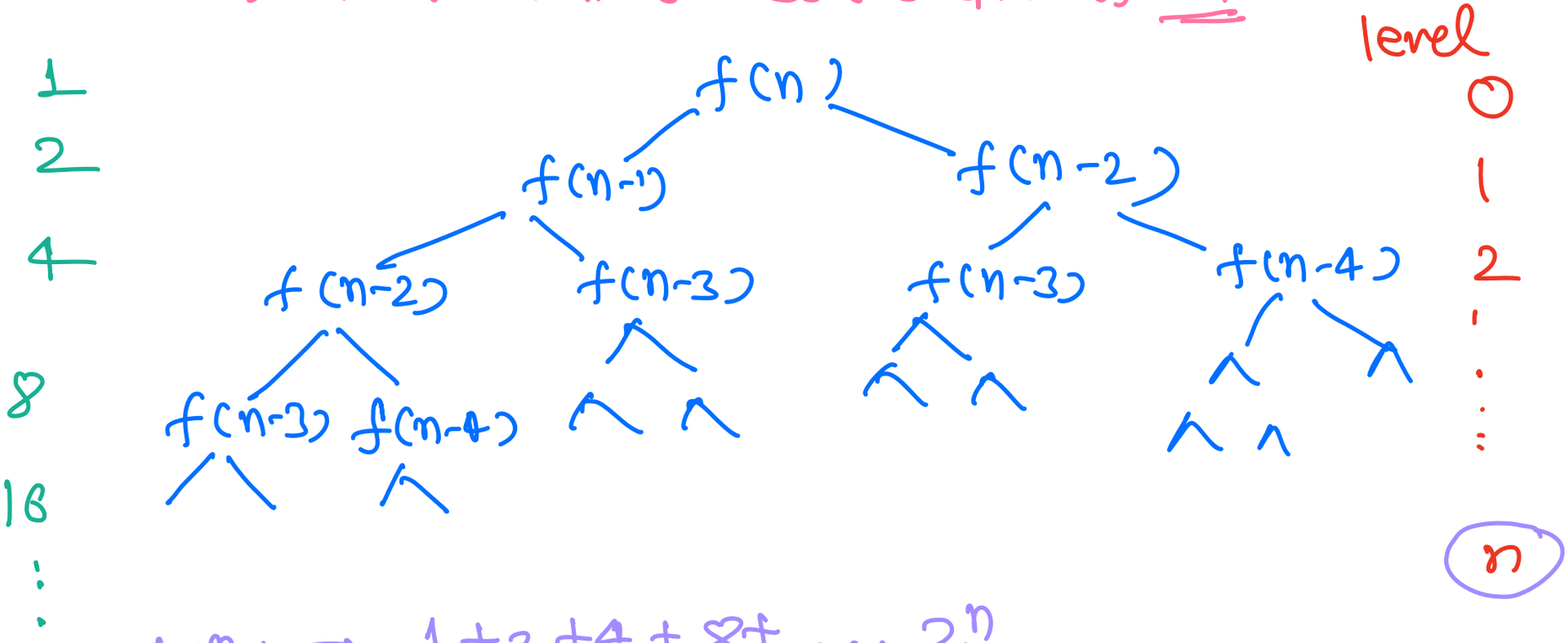
f(5)



4



Obj^n : recursive fn call depth is n



Obj^n : $\Rightarrow$ $1+2+4+8+\ldots 2^n$

n

$= 2^{n+1} - 1 \Rightarrow 2 \cdot 2^n - 1$

$\Rightarrow O(2^n)$

So. T.C. fib recursive fn is $O(2^n)$

S.C. $O(n)$ the size of stack

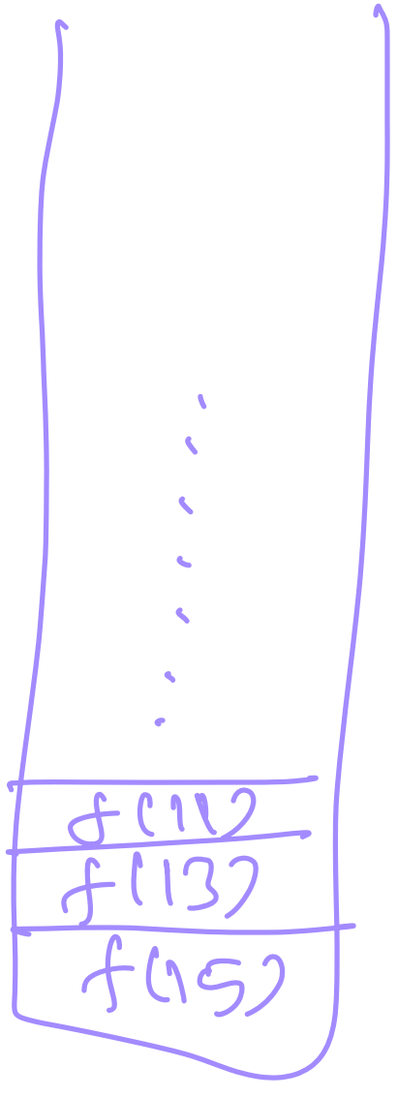
Doubt senion

```
bool isPalindrome(string st, int start, int end){
    if (start > end){
        return true;
    }

    if (st[start] != st[end]){
        return false;
    }
    return isPalindrome(st, start+1, end-1)
}
```

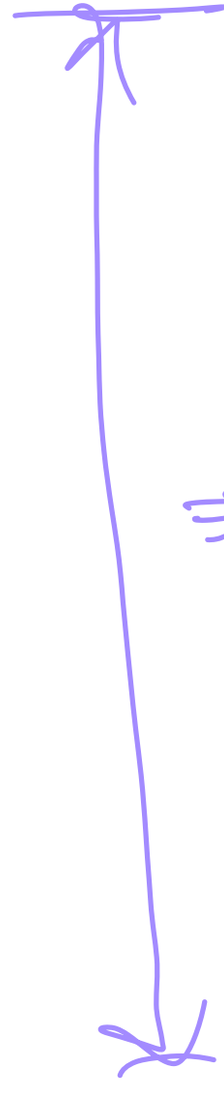
T.C. = O(N)
S.C. = O(N)

St = ABCDEFGHGFEDCBA , N = 15



f(15) → f(13) → f(11) → f(9) → f(7) → f(5) → f(3) → f(1) → f(-1)

O(1) + O(1) + O(1) + ...



N/2

O(N/2)

⇒ S.C

⇒ O(N)

$f(n) = f(n-K) + K/2$

$f(n) = f(n-2) + 1$

$= f(n-4) + 2$

$= f(n-6) + 3$

$= f(n-8) + 4$

Base $n = K$

$f(n) = f(n-n) + n/2$

$= f(0) + n/2$

$f(n) = n/2 + 1$ $O(n)$

Q.

```
int fun(x, n){
1.    if (n==0) return 1;
```

2. if (n is even)
3. return fun (x^2, n/2)
else
4 return x * fun (x^2, (n-1)/2)
}

fun (2, 10) ⟹ (2^2, 5) → (4, 5)

(4, 5) → (4 * (4, 2)

4 * (16, 2)

4 * (16^2, 1)

⟹ 4 * (256)

1024 = 1024

5. $\frac{5}{2} * 3 + 10$ operator

Stack

* / − + <, &

9.

```
int bar(x,y){
    if (y==0) return 0;
    return (x + bar(x,y-1))
}
```

Obsn ⇒ x*y

(2,2)

2 + (2,1)

2 + (2 + (2,0))

⇒ 2+2

⇒ x^2

```
int foo(x,y)
    if(y==0) return 1
    return bar(x, foo(x,y-1))
```

foo (3,5)

x^4

x
+x
+x
+x
} y times

bar (3, foo (3,4))

bar (3, foo (3,3))

3*3*3*3*3 ⇒ 243