Recursion - Part II

May 6th, 2022

AGENDA:

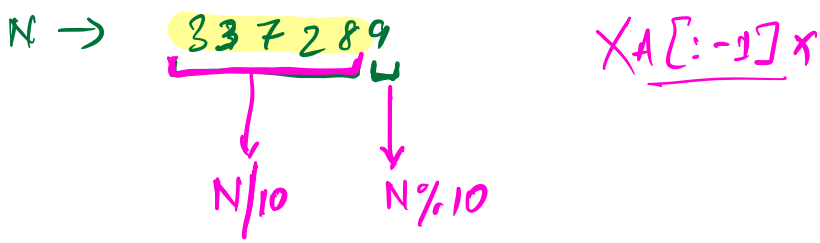
- Some more problems
  - Sum of digits
  - Power(a,n)
  - Power(a,n,p)
- Time and Space complexity

Q. Given a no. N, return sum of digits.

1. Assumption

def sod(N)
↳ Returns the sum of digits.

2. Main Logic.



sod(N) = sod(N/10) + N%10

3. Base condition

if N==0 return 0 ✓

if N<10 return N ✓

1, 2, 3, 4, 5, 6, 7, 8, 9

```
def sod(N):
    if N<10:
        return N
    return sod(N/10) + N%10
```

if N==0: return 0. — (9)

(**)

N/10 → 0

N%10 → (9)

* Revere a number.

XX Number → string → Reverse → Back to no.

$$\begin{cases} a = N \% 10 \\ N = N/10 \end{cases}$$

Q. Implement your own power function.

pow(a,n) ⟶ a^n

**

**
pow( )

Recursion

Main Logic.

** $a^n = a * a * a * a * a * a * \dots * a * a$

$a^n = a * a^{n-1}$

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

Base condition

* if n == 1 : return a

pow(a, N=1) = a

* if n == 0 : return 1

undefined

0^0 = 0 or 1 ?

$5^0 = 1$

$0^{10} = 0$.

a > 0

```
def pow(a,n):
    if n==0:
        return 1
    return a* pow(a, n-1)
```

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

$\downarrow$

$a * a^8$

$\downarrow$

$a * a^7$

N function calls.

a^n = a * a * a * a * a * ..... a * a * a

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

* a^{10} = $a^5 * a^5$ ← $a^N = a^{N/2} * a^{N/2}$

a^{14} = $a^7 * a^7$

a^{15} = $a^7 * a^7 * a$ ← $a^N = a^{N/2} * a^{N/2} * a$

Another approach.

```
def pow(a,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
```

O(N) ←

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

Optimised approach.

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

```
def pow(a,N):
```

```
def pow(a,N):
    if N==0:
        return 1
    he = pow(a,N/2)
    ha = he*he

    if N%2==0:
        return ha
    else:
        return ha*a
```

→ O(log N)

Dry Run.

pow(a,10) ↗ a^{10}

he = pow(a,5) ← a^5

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

pow(a,5)

he = pow(a,2) = a^2 ← a^2

ha = a^4

pow(a,2)

$a^4 * a$

ha*a

he = pow(a,1) = a

ha = a^2 ← a

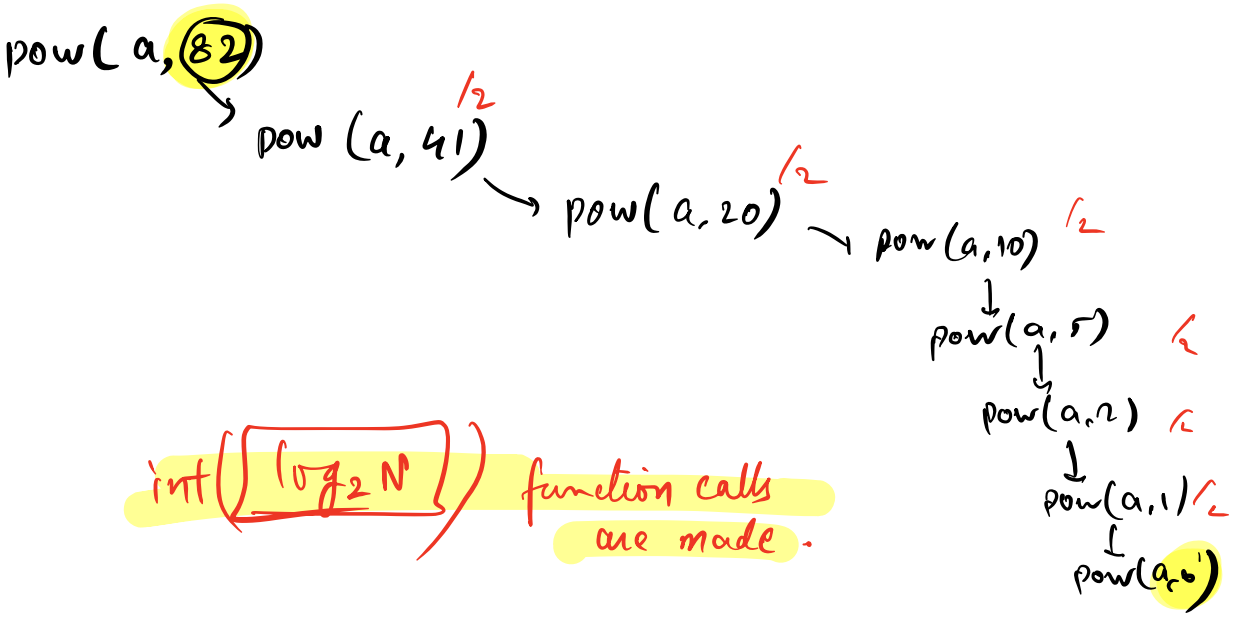
pow(a,1)

he = pow(a,0)

he = 1

ha = 1

ha*a = 1*a = a


pow(a, 82)
pow(a, 41)
/2
pow(a, 20)
/2
pow(a, 10)
/2
pow(a, 5)
/2
pow(a, 2)
/2
pow(a, 1)
/2
pow(a, 0)
int(⌊log2 N⌋) function calls are made.

Q.

**

Calculate

pow(a, n, p) → $a^n \% p$

Overflow.

a^n → 

```
def pow(a,N,p):
    a = a%p
    if N==0:
        return 1
    he = pow(a, N/2, p)
    ha = (he%p * he%p) %p

    if N%2==0:
        return ha
    else:
        return (ha*a)%p
```

(ha%p * a%p) %p

Break till 9:55

Time complexity of Recursive codes.

```
def sum(N):                          // sum of nos. from 1 to N
    if N==1 :          → O(1)
        return 1
    return sum(N-1) + N            → O(1)
```

$T(N)$ → Time taken to solve problem of size N.

Time function $f(x)$

$T(N-1)$ → Time " " " " size N-1

$$T(N) = 1 + T(N-1) \quad **$$

** Time function of recurrence relation

Substitute N with N-1 in the above equation

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

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

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

Substitute $N \to N-2$

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

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

$$** \; T(N) = 3 + T(N-3)$$

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

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

$T(N) = 5 + T(N-5)$

$k \qquad k$

Upon Generalisation,

$T(N) = k + T(N-k)$ (**) → $T(1)$, $N-k \to 1$

```
def sum(N):
    if N==1:
        return 1
    return sum(N-1) + N
```

$N - k = 1$

$k = N - 1$

$T(1) = 1$ (**)

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

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

$T(N) = N$

$\therefore$ TC: $\to O(N)$

* Time function $T(N)$ → Size N

$T(N-1)$ ← Size $N-1$

$T(Y)$ ← Size Y

$T(M)$ Size M

$T(N/2)$ ← Size $N/2$

Q. pow (a, N)

pow(a, N) → T(N)

pow(a, N/2) → T(N/2)

```
def pow (a,N) :
    if N==0 :
        return 1
    he = pow(a, N/2)
    ha = he*he

    if N % 2 == 0 :
        return ha
    else:
        return ha*a
```

O(1)

Recurrence relation on Time:

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

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

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

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

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

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

$$T(N) = 3 + 1 + T(N/16)$$

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

Generalising for general 'k'.

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

$T(N/2^k) \rightarrow T(1)$

** $T(1) = 1$

$$\frac{N}{2^k} = 1$$

$$\Rightarrow N = 2^k$$

$$\Rightarrow k = \log_2 N$$

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

$$= \log_2 N + 1$$

** TC : $O(\log_2 N)$

**

(A-1) ** $T(N) = 2\,T(N/2) + 1$

Given $T(1) = 1$

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

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

$= 4\,T(N/4) + 3$ **

2^2 2^2 $2^2 - 1$

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

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

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

2^3 2^3 $2^3 - 1$

For general k,

$T(N) = 2^k\,T\left(\frac{N}{2^k}\right) + 2^k - 1$ General K.

* ** $T(N) = K\,T(N/K) + K - 1$ Not for general K.

$3\,T(N/3) + 2$

K should be a power of 2.

K = 3

K = 5

K = 10

K = 2

$f\left(\frac{x}{4}\right) * 4 = f(x)$ XX

$$T(N) = 2^k \; T\left(\frac{N}{2^k}\right) + 2^k - 1 \qquad T(1) = 1$$

$T\left(\frac{N}{2^k}\right) \rightarrow T(1)$

$T(1) = 1$

** * $\frac{N}{2^k} = 1 \rightarrow N = 2^k$

$\Rightarrow \log_2 N = \log_2(2^k)$

$\Rightarrow \log_2 N = k$

$k = \log_2 N$

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

$$= N * 1 + N - 1$$

$$= 2N - 1$$

TC: $O(N)$

Q.

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

Given, $T(0) = 2$

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

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

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

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

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

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

for general, k,

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

** $T(0) = 1$

$T(0)$

$K = N$

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

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

Time complexity — $O(2^N)$

Q. Fibonacci

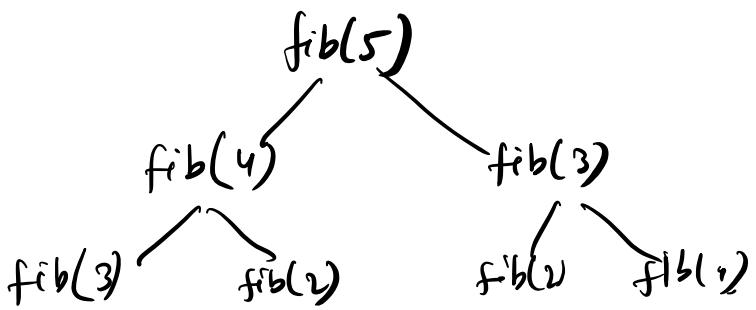
$$T(N) = T(N-1) + T(N-2) + 1$$

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

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

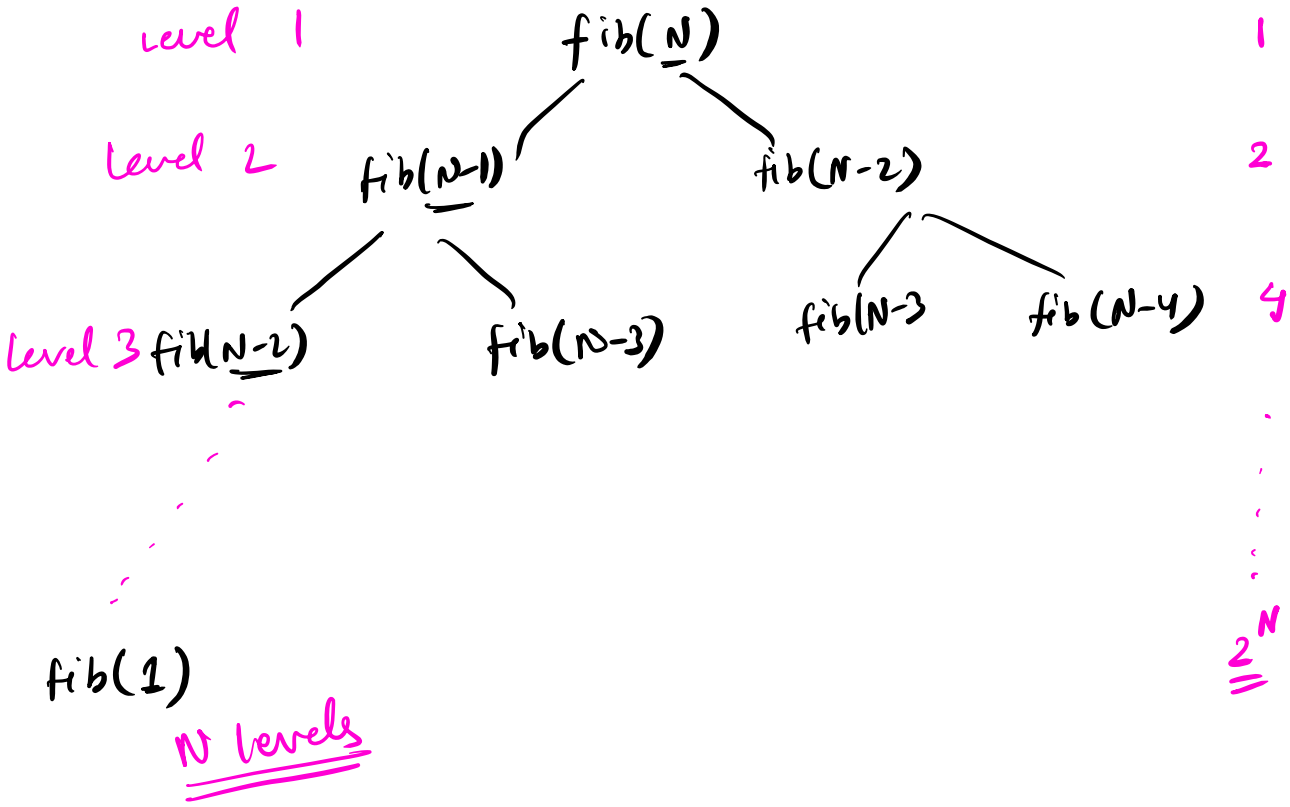
$$T(N) = T(N-2) + 2T(N-3) + T(N-4) + 1$$

Recursive Tree Method.



- At every step, tree divides up into 2 branches.
  my function call doubles by 2.

2^N exponential



Total no. of functions calls

$= 1 + 2 + 4 + 8 + 16 + \ldots\ldots 2^N$

$= 2^{N+1}$

Time complexity = $O(2^N)$

```
def fibo(n):
    if n==1 || n==2
        return 1
    return fibo(n-1) + fibo(n-2)
```

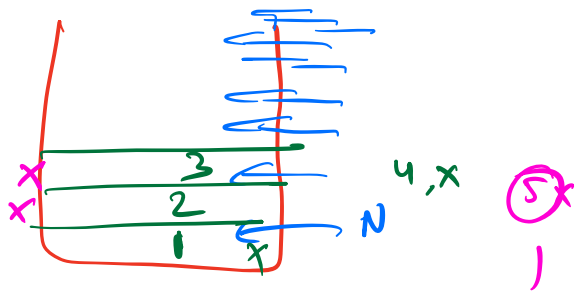
$O(2^N)$

↳ Dynamic programm Iterative manner.

O(N)

Space complexity

Stack

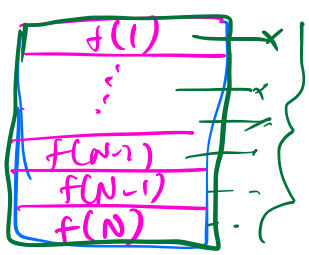


O(N) * *

** Maximum Stack size at any point of the program.

* sum(N)

SC: O(N)



** Fibonacci

fibo(N)

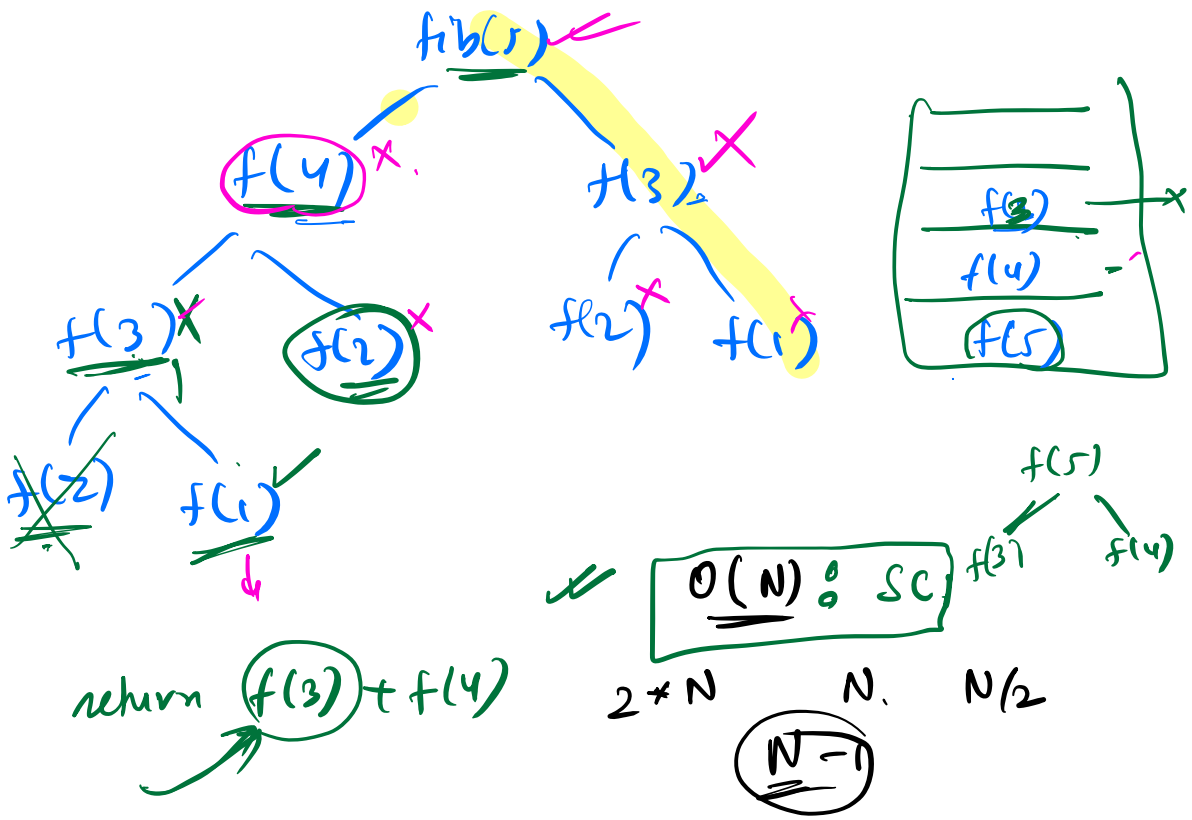
return fibo(N-1) + fibo(N-2)

fib(5)

2, 5, 10, 25, 32.

Max stack size = ?

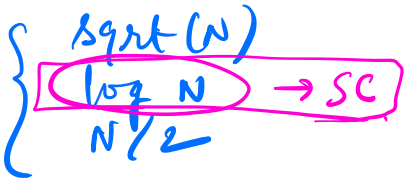
N/2 N, 2*N, N^2, 2^N



return (5*2) +3

Q. Power (a, N)

pow(a,10) → pow(a,5) → pow(a,2) → pow(a,1)



H.W

$T(N) = 2\,T(N/2) + O(N)$

Doubt session

```
bar(x, y)
    y == 0 → return 0
    return x + bar(x, y-1)
```

bar(3, 0)
bar(3, 1) = 3 + bar(3, 0) = 3

```
foo(x, y)
    y == 0 → return 1.
    return bar(x, foo(x, y-1))
```

foo(3, 1) = 1

bar(3, foo(3, 0))
bar(3, 1) = 3
1+
foo(3, 2) = 3
= bar(3, foo(3, 1))
= bar(3, 1)
= 3

foo(3, 5) = bar(3, foo(3, 4))

foo(3, 4)

bar(3, 2)
= 3 + bar(3, 1)
= 6

foo(3, 3)
= bar(3, foo(3, 2))
= bar(3, 3)
= 9

foo(3, 2) → foo(3, 1) → foo(3, 0)

foo(3, 4)
= bar(3, foo(3, 3))
= bar(3, 9)
= 27

bar(3, 3) = 3 + bar(3, 2)
= 9
bar(3, 4) = 12
bar(3, 5) = 15
⋮
bar(3, 9) = 27
bar(3, 27) = ?