Recursion - 2

Q1. Given a no. write recursive program to calculate sum of digits of that no.

eg ⇒ 4 2 6 8 9 ⇒ o/p = 29

// assumption

sumDigit(N) → returns sum of digits of N.

// main logic

sumDigit(42689) = 9 + sumDigit(4268)

N%10 ⇒ last digit

42689%10 ⇒ 9

N/10 ⇒ digits remaining after removing last digit

42689/10 ⇒ 4268

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

// base case

```
if(N<10)
    return N;
```

pseudo → pseudo

102%10 = 2

4892%10 = 2

400%10 = 0

100/10 = 10

5268/10 = 526

N = 9

Q2. Implement power function : given a, N, return a^N

a^N % d

ex ⇒ a = 3, N = 3 ⇒ $a^N = 3^3 = 27$

⇒ $a^N = a \times a^{N-1}$

pseudo

assumption pow(a, N) → return a^N

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

recursion

$2^5 \Rightarrow 2 \times 2^4$

$2^4 = 2 \times 2^3$

$2^3 = 2 \times 2^2$

$2^2 = 2 \times 2^1$

$2^1 = 2 \times 2^0$

$2^0 = 1$

$a^5 \rightarrow a \times a^4 \rightarrow a \times a^3 \rightarrow a \times a^2 \rightarrow a \times a^1 \rightarrow a \times a^0$

5 times multiplication

$a^N \rightarrow$ N multiplication

```
result = 1
for( i = 0; i < N; i++) {
        result = result x a
}
```

$\Rightarrow a^N$

N iterations / multiplications

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

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

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

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

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

$\Rightarrow a^{21} = a^{10} \times a^{10} \times a$

$a^N = a^{N/2} \times a^{N/2}$ if $N\%2 == 0$

$a^N = a^{N/2} \times a^{N/2} \times a$, if $N\%2 != 0$.

$\Rightarrow a^{64} \Rightarrow$ 64 multiplications using normal recursion or iteration

$a^{64} \rightarrow a^{32} \times a^{32}$ ✓

$a^{32} \rightarrow a^{16} \times a^{16}$ ✓

$a^{16} \rightarrow a^{8} \times a^{8}$ ✓

$a^{8} \rightarrow a^{4} \times a^{4}$ ✓

$a^{4} \Rightarrow a^{2} \times a^{2}$ ✓

$a^{2} \Rightarrow a^{1} \times a^{1}$ ✓

$\Rightarrow$ 6 multiplications

a^N is divided in $a^{N/2} \times a^{N/2}$ but N becomes 1

TC $\Rightarrow O(\log N)$

---

pseudo

TC $\Rightarrow O(\log N)$

```
int pow(a, N) {
    if(N == 0)
        return 1
    int halfpow = pow(a, N/2);
    if(n%2 == 0)
        return (halfpow%d x halfpow%d)%d
    else
        return (halfpow%d x halfpow%d x a%d)%d
}
```

a^N

$a^{N\%d}$

$\downarrow$

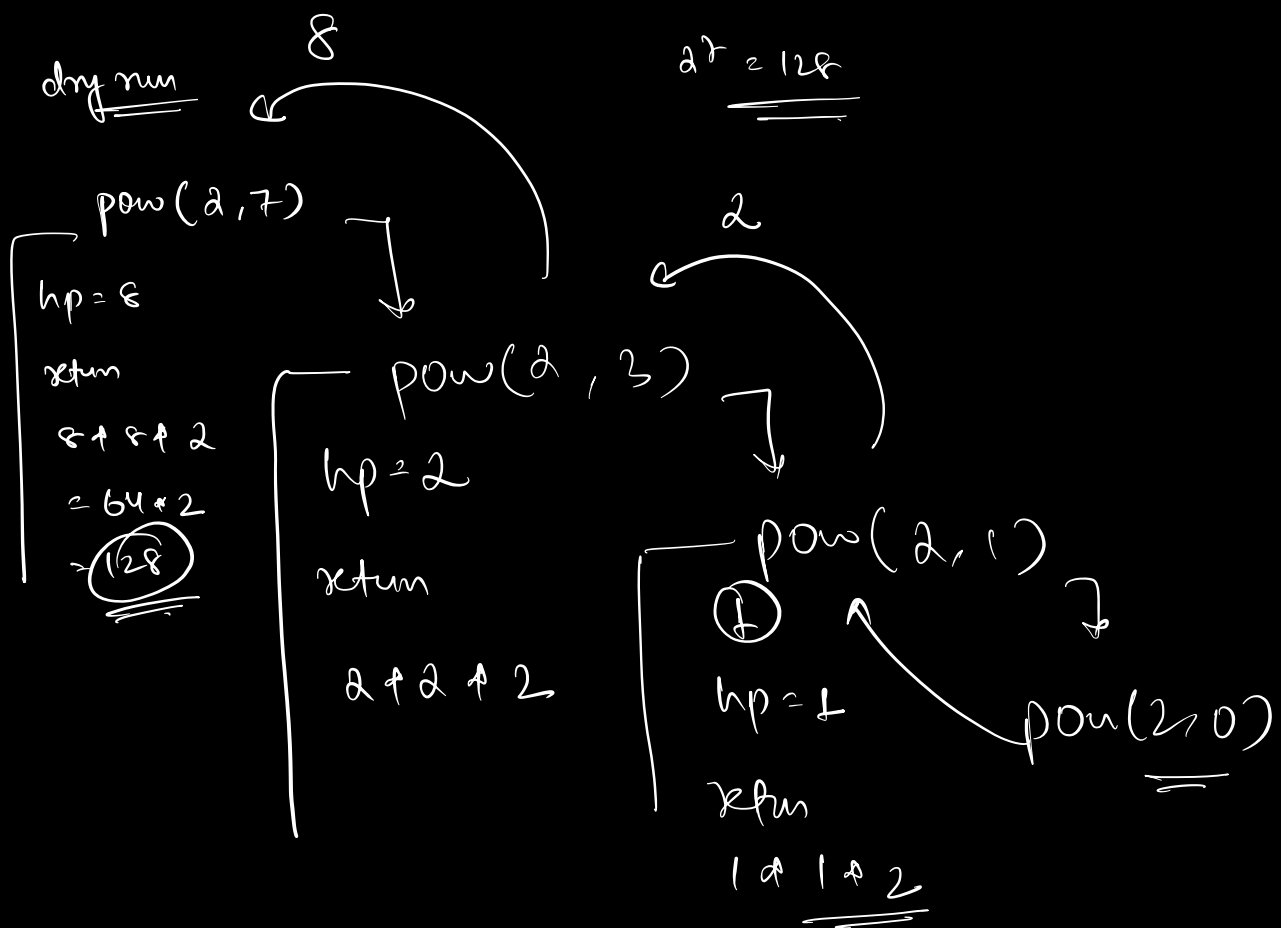
$(a^{N/2} \times a^{N/2})\%d$

$\downarrow$

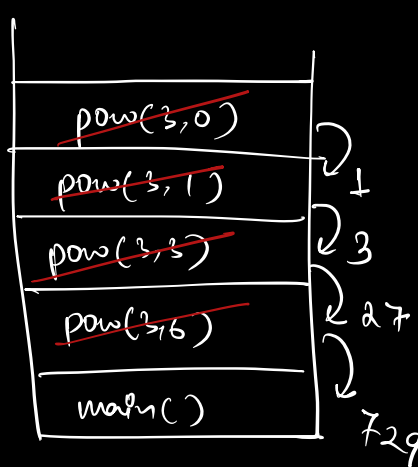
$(a^{N/2\%d} \times a^{N/2\%d})\%d$

dry run

$2^7 = 128$




```
int pow(a,N){
    if(N==0)
        return 1
    int halfpow = pow(a,N/2);
    if(n%2==0)
        return halfpow * halfpow
    else
        return halfpow * halfpow * a
}
```



pow(3,6)

hp = 1
N = 1
return 1 * 1 * 3

hp = 3
N = 3
return = 3 * 3 * 3

hp = 27
N = 6
return 27 * 27

729

1000 → check how pow() function of your language works.

instead return a^N, → return $a^N \% d$

a^{100}

a^{1000}

$a^N \rightarrow a^{N\%.d}$

$D = 10^9+3$

(Int)

: Time complexity for recursion :-

SUM(N) = N + SUM(N-1)

↓

(N-1) + SUM(N-2)

↓

(N-2) + SUM(N-3)

⋮

SUM(1)

= 1

$N \rightarrow N-1 \rightarrow N-2 \rightarrow N-3 \rightarrow \cdots \longrightarrow 1$

no. of addition / step = N.

TC ⇒ O(N)

T(N) ⇒ SUM(N)

T(N-1) ⇒ SUM(N-1)

SUM(N) = N + SUM(N-1) ⇒ (N-1) + SUM(N-2)

T(N) = 1 + T(N-1) ⇒ 1 + T(N-2)

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

= 2 + T(N-2)

SUM(N-2) = (N-2) + SUM(N-3)

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

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

= 3 + T(N-3)

kth step

$T(N) = k + T(N-k)$ ⇐

if kth last step.

$k = N$.

$T(N-k) = T(0)$

$\Rightarrow N = k$

$T(N-k) = 1$ (last step needs only 1 addn)

↓

$T(N-N) = T(0)$

$T(N) = N + T(0)$

$\Rightarrow \boxed{T(N) = O(N)}$

* Recurrence Relation

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

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

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

$= 2T(N-2)+1 \rightarrow 2^1 T(N-2) + (2^1-1)$

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

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

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

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

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

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

kth

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

last step $\downarrow$

$T(N-k) = 1$

$\downarrow$

$T(N-k) = T(0)$

$\Rightarrow N = k$

if k is last step

$N-k=0$

$T(0)$

$k = N$

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

$$= 2^N + 2^N - 1$$

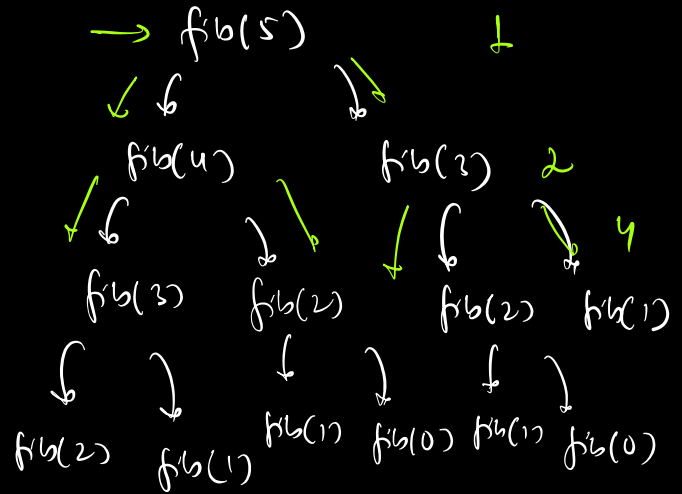
$$= 2(2^N) - 1$$

$$\boxed{T(N) = O(2^N)}$$

```
int fib(N) {
  if(N == 0)
    return 0
  else if( N == 1)
    return 1

return fib(N-1) + fib(N-2);
}
```

$T = O(2^N)$



$fib(N) \Rightarrow fib(N-1) + fib(N-2)$

$fun(N) = 2\,fun(N-1) + 1$

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

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

$\downarrow$

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

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

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

$= 4T(N/4) + 3 \Rightarrow 2^2 \cdot T\left(\frac{N}{2^2}\right) + 2^2 - 1$

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

$\downarrow N/4$

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

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

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

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

k^{th} step

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

if k is last step $\Rightarrow k = N$

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

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

$$\Rightarrow N = 2^k$$

$$\Rightarrow k = \log N$$

$$a^{\log_a N} = N$$

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

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

$$= 2N - 1$$

$$T(N) = O(N)$$

```
int pow(a, N) {
    if(N == 0)
        return 1
    if(n%2 == 0)
        return pow(a, N/2) x pow(a, N/2)
    else
        return pow(a, N/2) x pow(a, N/2) x a
}
```

→ int halfpow = pow(a, N/2)

hp x hp

pow(a, N) ⇒ 2 pow(N/2)

$T(N) \Rightarrow 2T(N/2) + 1 \Rightarrow O(N)$

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

$a^N = a^{N/2} \times a^{N/2} \rightarrow T(N/2)$

$a^{N/2} = a^{N/4} \times a^{N/4}$

$T(N/2)$