Recursion-2

Content

→ pow(a,n)

→ pow(a,n,p)

→ TC of recursive codes

→ SC of recursive codes

Question:

Given a,n. Find a^n using recursion [n>=0]

Note: Don't worry about overflows

eg

a	n	a^n
2	5	$2^5 = 32$
3	4	$3^4 = 81$

Approach:

```
int pow1 (a,n){ // ass: calculate & return a^n
    if(n==0)
        return 1
    return (pow1(a,n-1) * a);
}
```

$$a^n = \underbrace{\overbrace{a*a*a* \cdots \cdots a}^{n\ \text{times}}}_{n-1\ \text{times}} * a$$

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

$n=0,\ a^0=1$

Approach 2

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

$= a^5 \times a^5$

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

$a^{11} = a^6 \times a^5$

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

$$\boxed{a^n = \begin{cases} a^{n/2} \times a^{n/2} & \text{if } n \text{ is even} \\ a^{n/2} \times a^{n/2} \times a & \text{if } n \text{ is odd} \end{cases}}$$

```
int pow2 (a,n) {
    if(n==0) return 1
    if (n%2 ==0) {
        return ( pow2(a,n/2) x pow2(a,n/2))
    }
    else {
        return( pow2(a,n/2) x pow2(a,n/2) x a)
    }
}
```

Approach 3

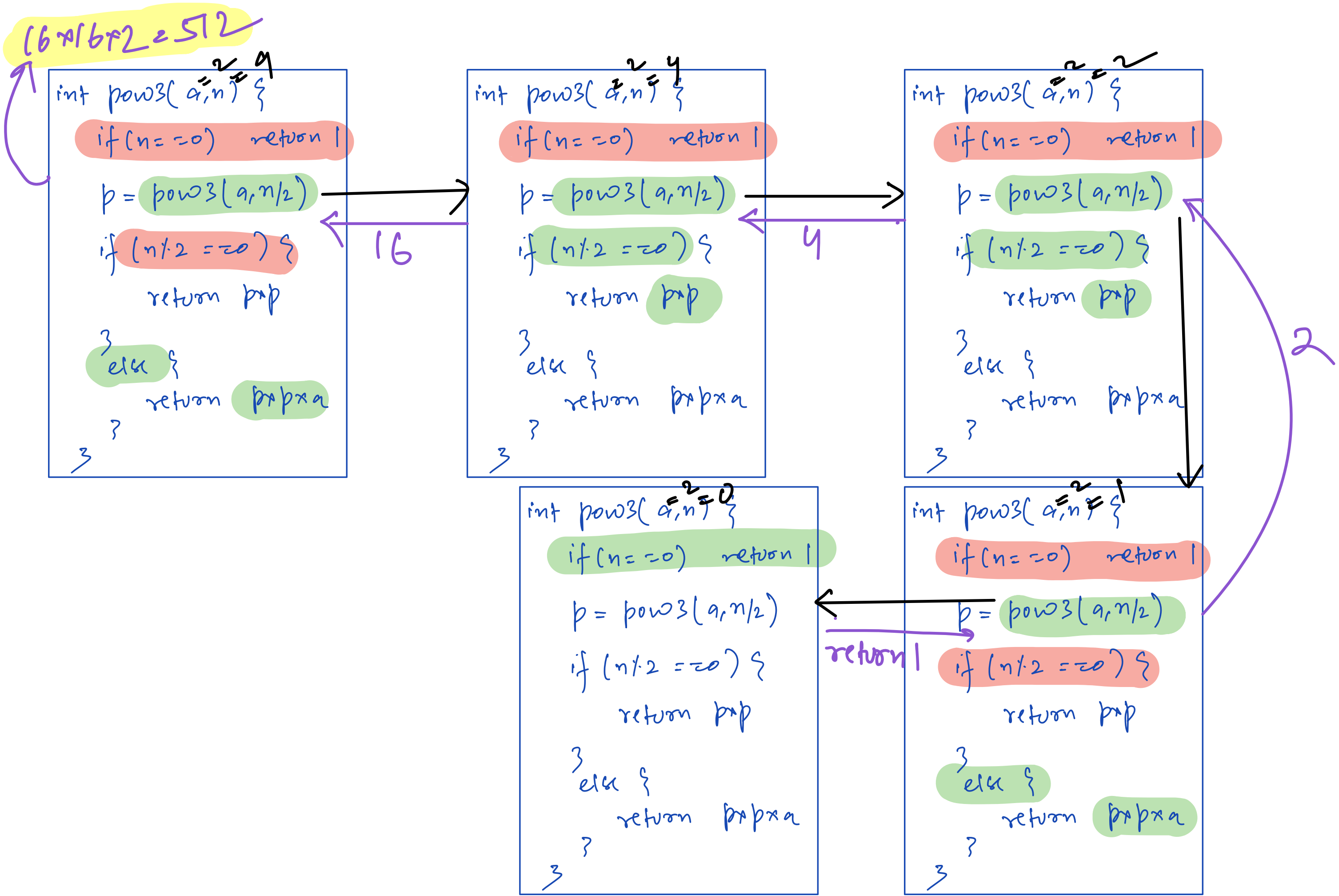
```
int pow3( a,n) {
    if(n==0)   return 1

    p = pow3(a,n/2)

    if (n%2 ==0) {
        return p*p
    }
    else {
        return p*p*a
    }
}
```

Tracing (Dry Run)

Calculate $2^9 = 512$



Question 2

Given a, n, m. Calculate $a^n \% m$.

Note: take care of overflows

Constraints:

$1 <= a <= 10^9$

$1 <= n <= 10^9$

$2 <= m <= 10^9$

```
int powmod( int a, int n, int m) {
    return pow3(a,n) % m
}
```

$\rightarrow a^n = (10^9)^{10^9}$ × Can't be stored

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

$$a^n \% m = (a^{n/2} \times a^{n/2}) \% m$$

$$= (a^{n/2} \% m \times a^{n/2} \% m) \% m$$

```
int powmod ( int a, int n, int m ) {
    if (n==0)  return 1
    ~~int~~ long p = powmod (a, n/2, m) // p = a^(n/2) % m  => [0, m-1]
                                           at max, p = m-1
                                                     ≈ 10^9
    if (n%2 == 0)
        return (p*p)%m
               (10^9*10^9) = 10^18 % m
    else
        return (p*p*a) % m
               10^9*10^9*10^9 = 10^27 % m
}
               ((p*p)%m * a) % m
                10^9 10^9
               (10^18 % m * a) % m
               (10^9 * 10^9) % m
               10^18 % m
```

$(p\%m \times p\%m \times a\%m)\%m$

$10^9 \times 10^9 \times 10^9$

10^{27} ✗

$((p\%m \times p\%m)\%m \times a\%m)\%m$

final Code

```
int powmod ( int a, int n, int m ) {
    if (n==0)  return 1
    long p = powmod (a, n/2, m)
    if (n%2 == 0)
        return (p*p) % m
    else
        return ((p*p)%m * a) % m
```

}

TC for recursive codes using recurrence relation

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

$\underbrace{sum(N-1)}$ time to calculate $= f(N-1)$

say time taken to calculate sum(N) $= f(N)$

$f(n) = f(n-1) + 1 \quad (O(1))$

using base condition

$f(1) = 1$

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

$\downarrow$

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

$= f(n-3) + 3$

after K steps

$f(n) = f(n-K) + K \qquad f(1) = 1$

$n - K = 1 \implies K = n-1$

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

$f(n) = O(N)$

②

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

$\underbrace{fact(N-1)}_{f(N-1)}$

time taken to calculate fact(N) =

$f(N)$

$f(N) = f(N-1) + 1 \qquad f(1) = 1$

$f(N) = O(N)$

③

```
int pow1(a,n) {
    if(n==0) return 1
    return pow1(a,n-1) * a
}
```

$\underbrace{pow1(a,n-1)}_{f(n-1)}$

time taken to calculate

$pow1(a,n) = f(n)$

$f(n) = f(n-1) + 1 \qquad f(0) = 1$

$f(n) = O(N)$ ← TODO (if not understand)

④

```
int pow2(a,n) {
    if(n==0) return 1
    if(n%2 == 0) {
        return (pow2(a,n/2) * pow2(a,n/2))
    }
    else {
        return( pow2(a,n/2) * pow2(a,n/2) * a)
    }
}
```

$\underbrace{pow2(a,n/2)}_{f(n/2)} + \underbrace{pow2(a,n/2)}_{f(n/2)}$

time taken to calculate

$pow2(a,n) = f(n)$

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

$f(0) = 1$

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

$$\Downarrow$$

$$2f(n/4)+1 = 2\left(2f(n/4)+1\right)+1$$

$$= 4f(n/4) + 2 + 1 \qquad = 2^2 f\left(\tfrac{n}{2^2}\right) + 2^2 - 1$$

$$\Downarrow$$

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

$$f(n) = 4\left(2f(n/8)+1\right) + 2 + 1 \quad 3$$

$$= 8f(n/8) + 4 + 2 + 1 \quad 7$$

$$f(n) = 2^3 f\left(\tfrac{n}{2^3}\right) + 2^3 - 1$$

After K Steps

$$f(n) = 2^K f\left(\tfrac{n}{2^K}\right) + 2^K - 1 \qquad \begin{array}{l} f(0)=1 \\ f(1)=1 \end{array}$$

$$n/2^K = 1 \Rightarrow 2^K = n \qquad K = \log_2 n$$

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

$$= nf(1) + n - 1 = n + n - 1 = O(N)$$

$$f(n) = O(N)$$

(5)

```
int pow3(a,n) {
    if(n==0) return 1
    p = pow3(a,n/2)
    if (n%2 ==0) {
        return p*p
    }
    else {
        return p*p*a
    }
}
```

$f(n/2)$

time taken to calculate $pow3(a,n) = f(N)$

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

$$\downarrow$$

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

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

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

After K steps

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

$f(0) = 1$

$f(1) = 1$

$$n/2^K = 1 \Rightarrow K = \log_2 N$$

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

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

$$f(n) = O(\log_2 N)$$

⑥
```
int powmod ( int a , int n , int m) {
    if (n==0)  return 1
    long p = powmod (a, n/2, m)
    if (n%2 ==0)
        return (p*p) % m
    else
        return ((p*p)%m * a) % m
}
```

$\underbrace{powmod(a, n/2, m)}_{f(n/2)}$

$powmod(a, n, m) = f(n)$

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

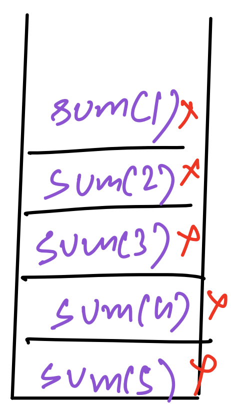
$\boxed{f(n) = O(\log_2 N)}$

Space Complexity for recursion

Observation: Function calls are stored in stack, hence it will take extra space.

So, space complexity = stack size (max. stack size)

```
int sum(N) {
    if(N==1) return 1
    return sum(N-1)+N
}
```
⑦



N=5

sum(5)
↓
sum(4)
↓
sum(3)
↓
sum(2)
↓
sum(1)

max stack size of the function = N

$SC: O(N)$

②
```
int pow3(a,n) {
    if(n==0) return 1
    p = pow3(a,n/2)
    if(n%2 ==0) {
        return p*p
    }
    else {
        return p*p*a
    }
}
```

say $N = 16$

pow3(a,0) ✗
pow3(a,1) ✗
pow3(a,2) ✗
pow3(a,4) ✗
pow3(a,8) ✗
pow3(a,16) ✗

pow3(a,16)
↓
pow3(a,8)
↓
pow3(a,4)
↓
pow3(a,2)
↓
pow3(a,1)
↓
pow3(a,0)

SC: $O(\log_2 N)$

TC of fibonacci

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

$$\text{return } \underbrace{fib(N-1)}_{f(n-1)} + \underbrace{fib(N-2)}_{f(n-2)}$$

time to calculate $fib(N) = f(n)$

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

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

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

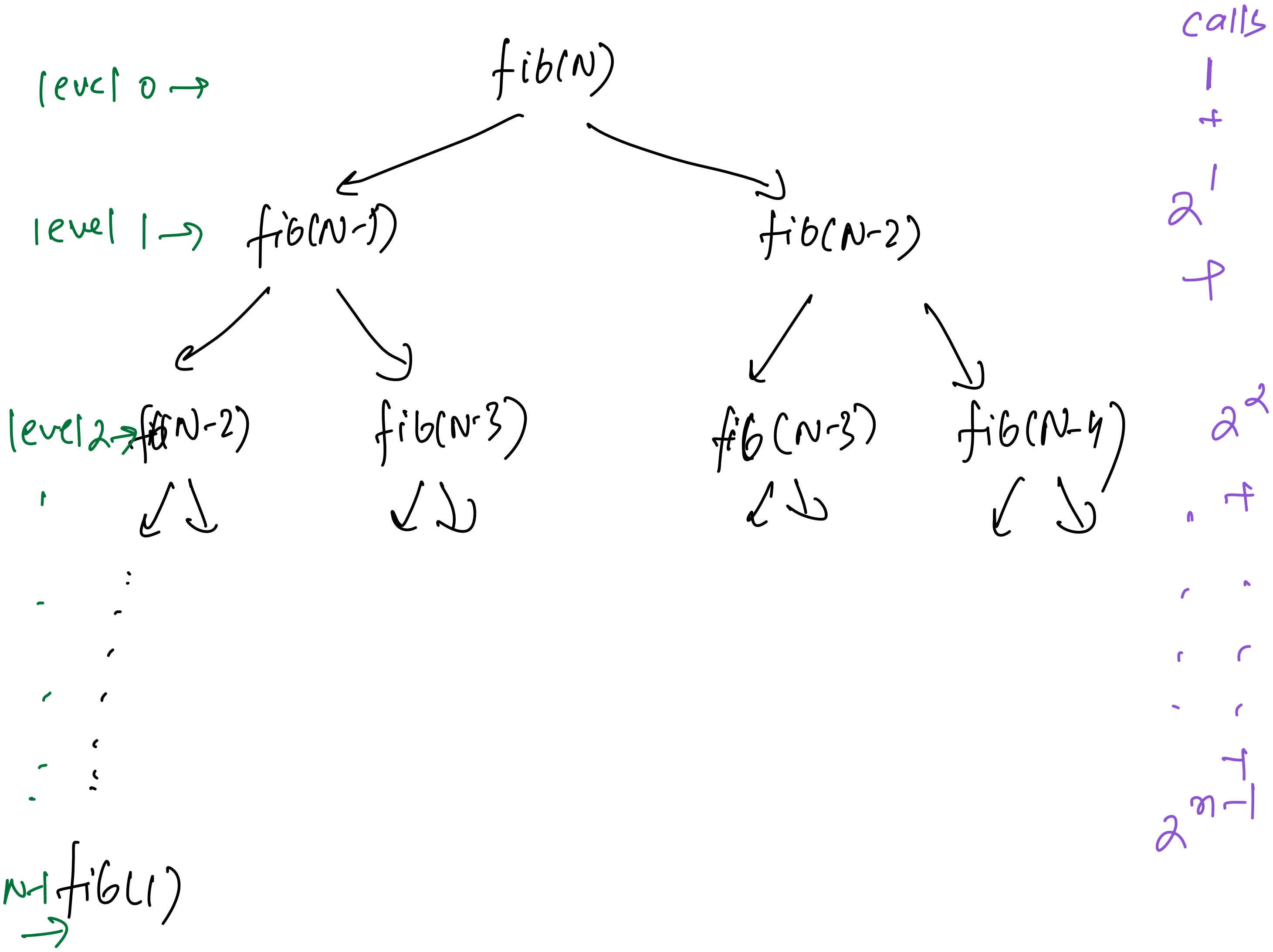
$\downarrow$

$f(n-2) + f(n-3) + 1 \Rightarrow 2f(n-2) + f(n-3) + 2$

$= 3f(n-3) + 2f(n-4) + 4$

$= 5f(n-4) + 3f(n-5) + 7$

→ NOT a good approach



total function calls $= 2^0 + 2^1 + \ldots + 2^{n-1}$

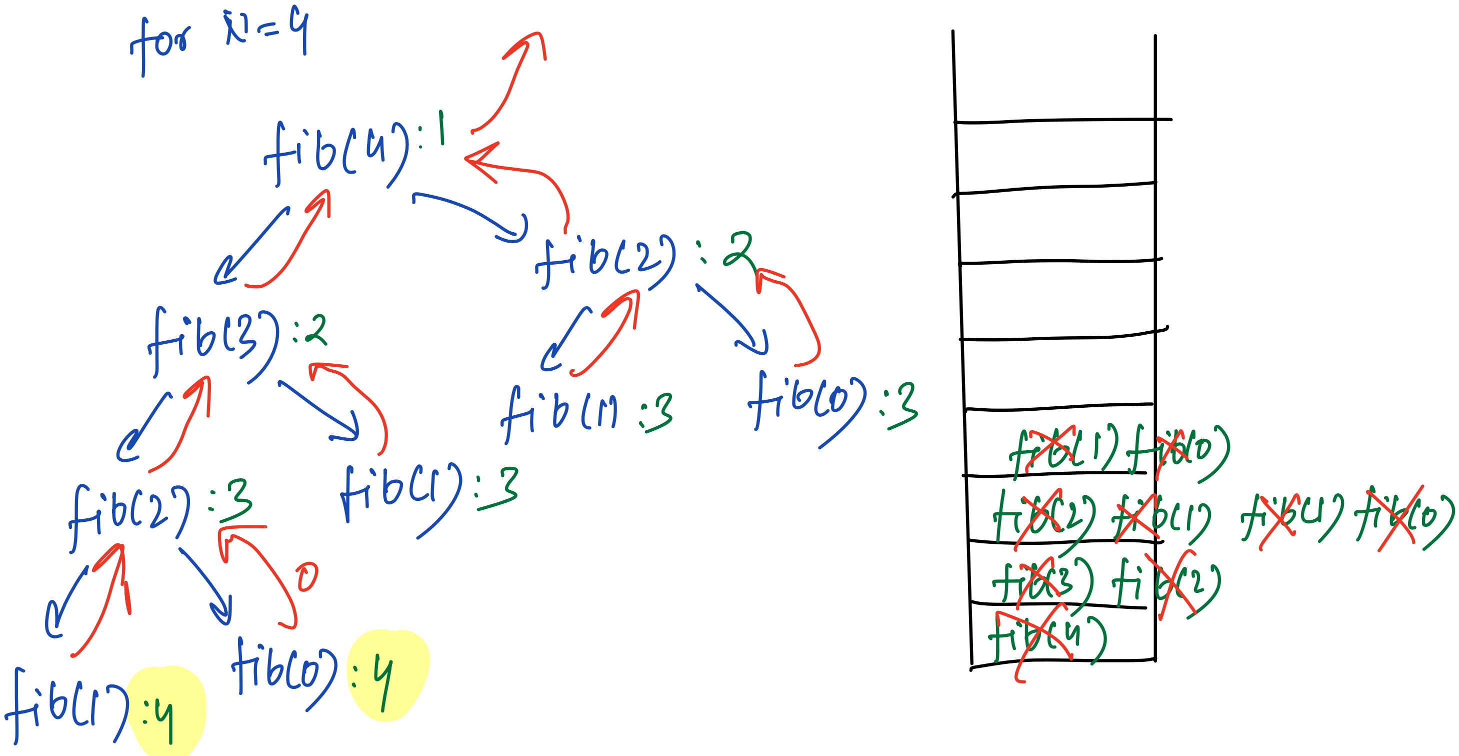
G.P.: $a = 2^0$, $n = n$, $r = 2$

$$a\left(\frac{r^n-1}{r-1}\right) = 1\left(\frac{2^n-1}{2-1}\right) = 2^n-1$$

$$f(n) = O(2^n)$$

SC of fibonacci

for n=4



max stack size = N

$$SC: O(N)$$