Recursion - 2

Content

- → pow(a,n)
- → pow(a,n,p)
- → TC of recursive codes
- → SC of recursive codes

Question 1

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 1

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

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

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

$a^0 = 1$

$a^1 = a$

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) * pow2(a,n/2) );
    }else {
        return( pow2(a,n/2) * pow2(a,n/2) * a );
    }
}
```

Approach 3

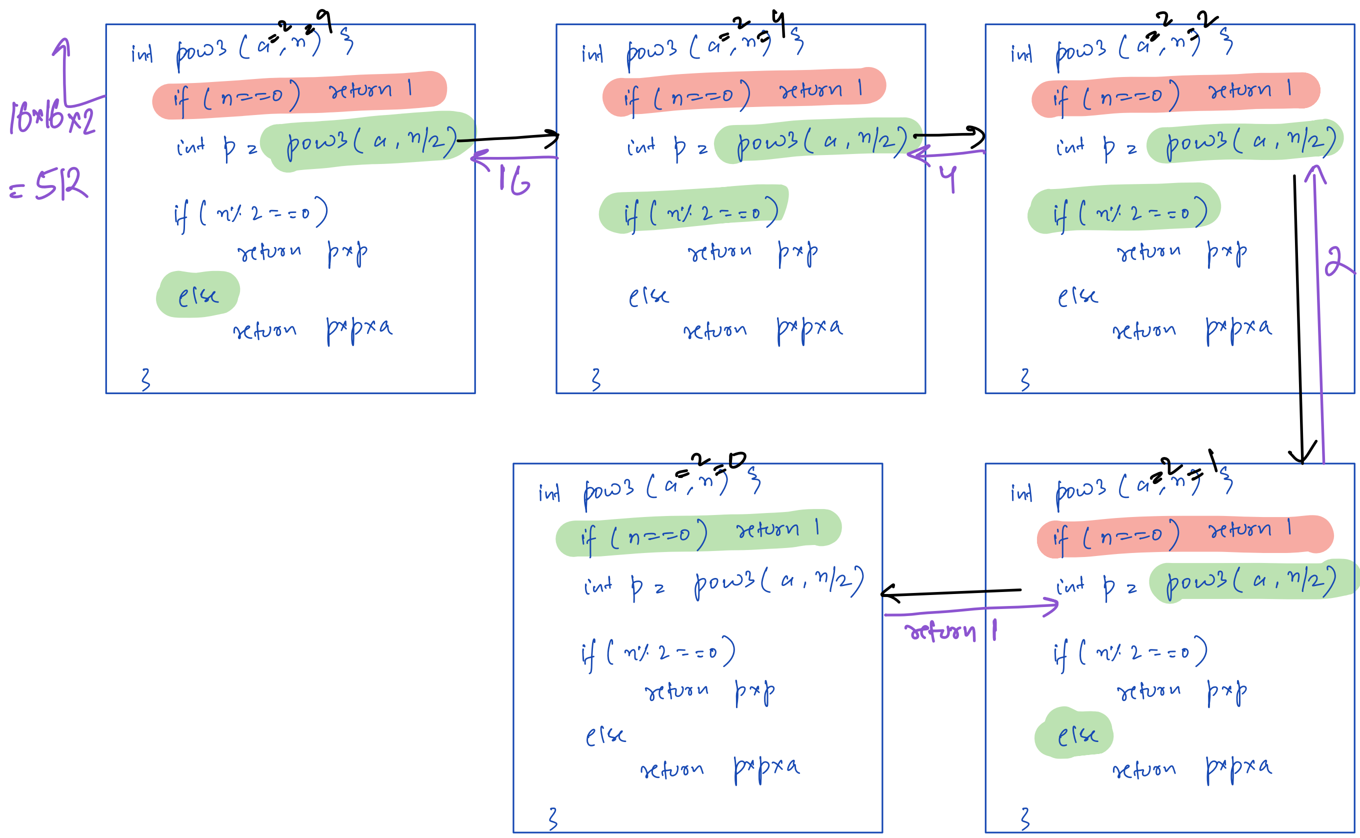
```
int pow3(a,n) {
    if (n==0)  return 1
    int p = pow3(a,n/2)
```

```
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 : $1 <= a <= 10^9$

$1 <= n <= 10^9$

$~~1~~ 2 <= m <= 10^9$

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

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

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

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

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

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

long

// $p = a^{n/2} \% m \Rightarrow [0, m-1]$

at max $p = m-1$

$= 10^9$

$(10^9 \times 10^9) = 10^{18} \% m$

}

$10^9 \times 10^9 \times 10^9 = 10^{27} \% m$

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

$\underbrace{}_{10^9 \times 10^9}$

$((10^{18} \% m) \times a) \% m$

$(10^9 \times a) \% m$

10^9

$= 10^{18} \% m = 10^9$

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 $= f(N-1)$

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

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

using base condition $f(1) = 1$

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

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

$\quad = f(n-3) + 3$

After K steps

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

$n - K = 1 \Rightarrow K = n-1$

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

$f(n) = O(N)$

②

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

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

time taken to calculate fact(N) = f(N)

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

$f(N) = O(N)$

③

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

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

time taken to calculate powl(a, n) = f(n)

$f(0) = 1$

$f(n) = O(N)$

TODO

(4)

```
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
}
```

$\underbrace{pow3(a, n/2)} \hookrightarrow f(n/2)$

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

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

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

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

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

After K steps:

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

$f(0) = 1$

$f(1) = 1$

$n/2^K = 1 \Rightarrow n = 2^K \Rightarrow \boxed{K = \log_2 n}$

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

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

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

5) int pow2(a, n) {

if (n == 0) return 1

if (n % 2 == 0)

$$\text{return}\left(\underbrace{pow2(a, n/2)}_{f(n/2)} \times \underbrace{pow2(a, n/2)}_{f(n/2)}\right)$$

else

$$\text{return}\left(pow2(a, n/2) \times pow2(a, n/2) \times a\right)$$

}

time taken to calculate pow2(a, n) = f(n)

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

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

$$f(n) = 2\left[2f(n/4) + 1\right] + 1 = 4f(n/4) + 3 \qquad 2^2 f\left(\frac{n}{2^2}\right) + 2^2 - 1$$

$$f(n) = 8f(n/8) + 7 \qquad 2^3 f\left(\frac{n}{2^3}\right) + 2^3 - 1$$

After K steps

$$f(n) = 2^K f(n/2^K) + 2^K - 1 \qquad f(0) = 1 \quad f(1) = 1$$

$$\boxed{K = \log_2 N} \qquad \boxed{2^K = n}$$

$f(n) = n f\left(\frac{n}{n}\right) + n - 1$

$= n + n - 1 \quad = 2n - 1$

$f(n) = O(N)$

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

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

$\text{powmod}(a,n,m) = f(n)$

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

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

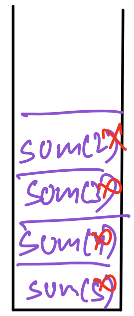
Space complexity for recursion

Observation : function calls are stored in stack, hence it will be extra space.

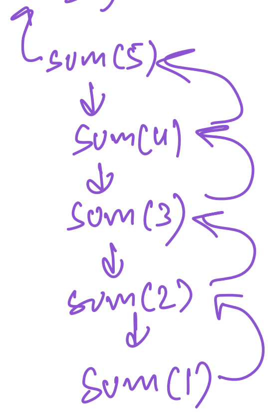
So, space complexity = stack size (max. stack size)

(1)

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



say N=5



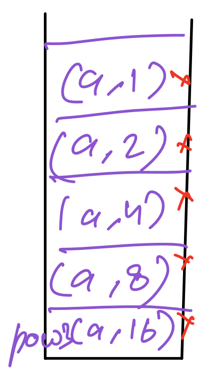
max stack size of the program = N-1

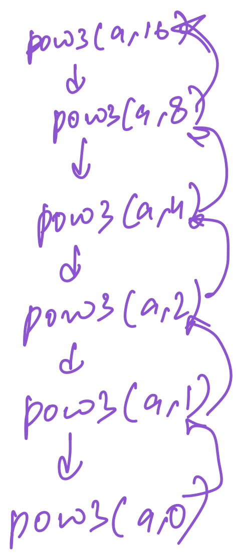
SC : O(N)

(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





SC : $O(\log_2 N)$

TC of fibonacci

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

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

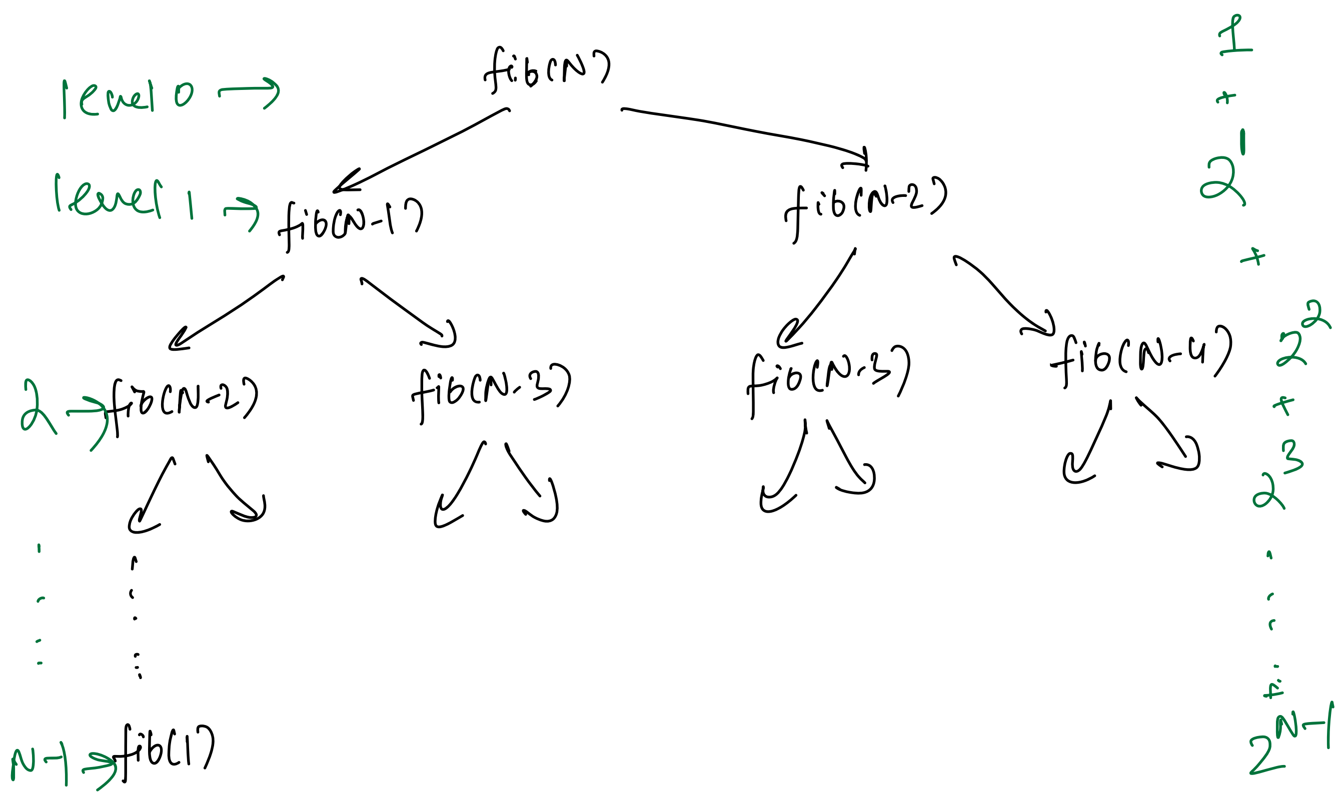
time to calculate $fibo(N) = f(n)$

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

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

$$\begin{aligned} f(n) &= f(n-1) + f(n-2) + 1 \\ &= f(n-2) + f(n-3) + 1 + f(n-2) + 1 \\ &= 2f(n-2) + f(n-3) + 2 \\ &= 3f(n-3) + 2f(n-4) + 4 \end{aligned}$$

→ NOT a good approach



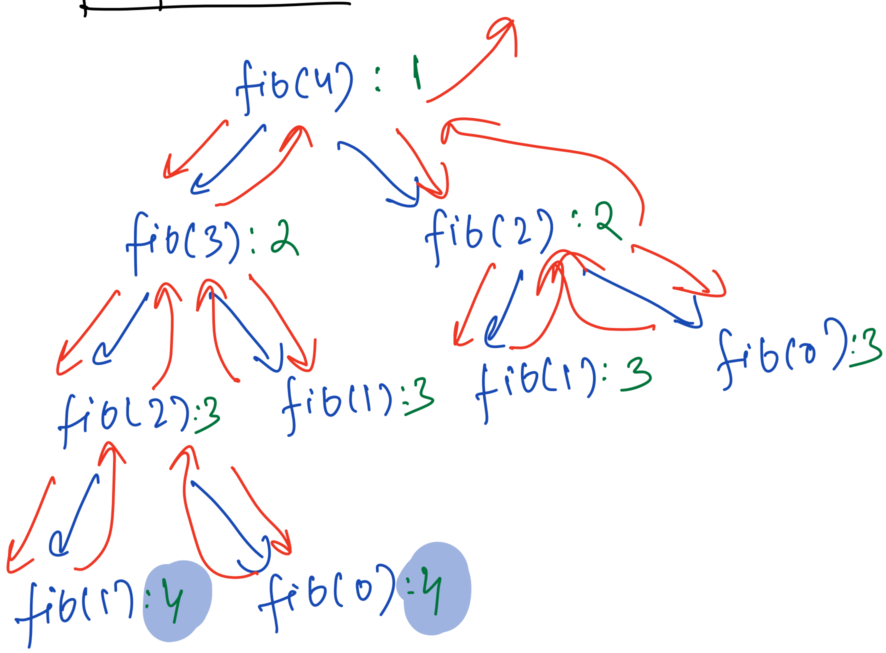
total function calls = $2^0 + 2^1 + \dots + 2^{n-1}$

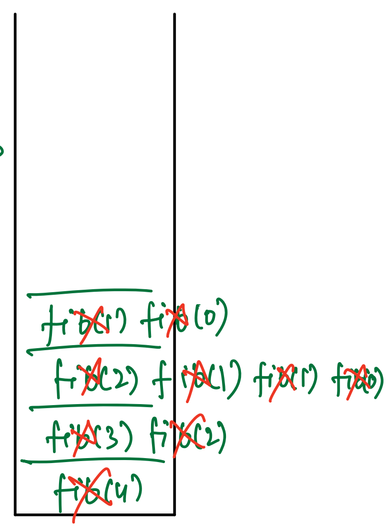
G.P $\quad a = 1, \; r = 2, \; n = n$

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

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

SC of fibonacci $\qquad$ for N=4





SC : O(N)