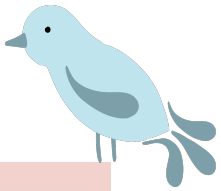
Recursion 2



> The capacity to learn is a gift; The ability to learn is a skill; The willingness to learn is a choice.
>
> BRIAN HERBERT

Good Evening

Today's content →

01. Sum of digits
02. Pow1 function
    Pow2 function
    Pow3 function / Fastpow function
03. Powmod function
04. Time & Space complexity

01. Find the sum of digits of a number.

sumD(1234) = 1 + 2 + 3 + 4 = 10

sumD(639) = 6 + 3 + 9 = 18

sumD(6) = 6

$N = d_1\ d_2\ d_3\ d_4 \cdots d_{x-1}\ d_x$

$sumD(N) = d_1 + d_2 + d_3 + \ldots + d_{x-1} + d_x$

$sumD(N) = sumD(x-1) + d_x$

$sumD(x-1)$ → $N/10$

d_x → $N\%10$

$sumD(N) = sumD(N/10) + N\%10$

```
int sumD(N)
{
    if (n==0) return 0          // if(N<10) return N

    return sumD(N/10) + N%10
}
```

01. Powl function

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

a	n	→	a^n
2	5	→	$2^5 = 32$
2	4	→	$2^4 = 16$

$$2^5 = \underbrace{2*2*2*2}_{2^4} * 2$$

$$2^4 = 2*2*2*2$$

Assumption → Calculate & return a^n

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

    return powl(a, n-1) * a;
}
```

$$a^n = \underbrace{a*a*a*a\ldots*a}_{a^{n-1}} * a$$

*

$a^{10} = a^9 * a$ → $a^5 * a^5$

$a^{11} \rightarrow a^{10} * a$ → $a^5 * a^5 * a$

$a^{14} = a^{13} * a$ → $a^7 * a^7$

$a^{23} \rightarrow a^{22} * a$ → $a^{11} * a^{11} * a$

- a^n : n is even

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

a^n : n is odd

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

```
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 ;
    }
}
```

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

Tracing

Return 512

```
int pow3(a,n)  a=2, n=9
|  if(n==0) return 1;
|  int y = pow3(a,n/2)        //16
|  if (n%2==0) return y*y;
|  else return y*y*a;
3                    16*16*2
```

return 16

```
int pow3(a,n)  a=2, n=4
|  if(n==0) return 1;
|  int y = pow3(a,n/2)        //4
|  if (n%2==0) return y*y;
|  else return y*y*a;
3                    4*4
```

return 4

```
int pow3(a,n)  a=2, n=2
|  if(n==0) return 1;
|  int y = pow3(a,n/2)        //2
|  if (n%2==0) return y*y;
|  else return y*y*a;
3                    2*2
```

return 2

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

return 1

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

Mod operator

$9 \% 5 \longrightarrow 4$

$8 \% 5 \longrightarrow 3$

$2 \% 5 \longrightarrow 2$

$0, 1, 2, 3, 4, 5, 6, 7, 8 \} \ \% 3 \longrightarrow [0, 1, 2]$

$-\infty \ldots \infty \} \ \% m = [0 \quad m-1]$

01. $(a+b) \% m = (a\%m + b\%m) \% m$

02. $(a*b) \% m = (a\%m * b\%m) \% m$

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

Constraints

$1 \le a \le 10^9$

$1 \le n \le 10^9$

$2 \le m \le 10^9$

Note :- Overflow condition must be handled

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

$$powmod(a,n,m) = \{ \underbrace{a^{n/2} \% m}_{powmod(a, n/2, m)} * a^{n/2} \% m \} \% m$$

```
powmod(a, n, m)
    if (n==0) return 1;
    int y = powmod(a, n/2, m);   // y = a^(n/2) % m
    if (n%2==0)
        return (y*y)%m
    else
        return (y*y*a)%m
```

Max y = m - 1

max y ≒ m ≒ 10^9

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

int is not sufficient, we need long

$10^9 * 10^9 * 10^9 = 10^{27}$

long variable is also insufficient

Solution for multiplication of 3 no

$((y * y)\%m * a\%m)\ \%\ m$

$10^9 * 10^9$

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

$(10^9 * 10^9)\ \%\ m \simeq 10^9$

```
powmod(a, n, m)  //Handle typecasting
if (n==0) return 1
long y = powmod(a, n/2, m);
if (n % 2 == 0)
    return (y * y) % m
else {
    return ((y * y) % m * a % m) % m
}
```

10:13 → 10:23

Time complexity

01. Figure out recursive relation / recurrence relation
02. Generalised it for k steps
03. Compose the generalised eqn with base case condition.

// Assumption → Time taken to calculate sum(N) = f(n)

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

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

$f(n) = f(n-1) + 1$ } recurrence relation

$f(1) = 1$

$f(n) = f(n-1) + 1$ → 1st

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

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

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

$$f(n) = f(n-2) + 2 \longrightarrow 2^{nd}$$

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

$$f(n-2) = f(n-3) + 1$$

$$f(n) = f(n-3) + 1 + 2$$

$$f(n) = f(n-3) + 3 \longrightarrow 3^{rd} \text{ substitution}$$

After k substitution

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

$$n - k = 1$$

$$k = n - 1$$

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

$$= 1 + n - 1$$

$$f(n) = n$$

Time complexity = $O(n)$

02.

Assumption → Time to execute fact(n) = f(n)

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

f(n-1)

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

$f(1) = 1$

TC: O(N)

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

TC: O(N) TODO

3. Assumption → Time taken for Pow3(a,n) = f(n)

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

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

$f(0) = 1$

$f(1) = 1$

$f(n) = f(n/2) + 1 = f\left(\frac{n}{2^1}\right) + 1$ — 1

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

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

$f(n) = f(n/4) + 2 = f\left(\frac{n}{2^2}\right) + 2$ — 2

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

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

$f(n) = f(n/8) + 3 \quad f\left(\frac{n}{2^3}\right) + 3$ — 3

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

$f(n) = f(n/16) + 1 + 3$

$f(n) = f(n/16) + 4 \quad f\left(\frac{n}{2^4}\right) + 4$ — 4

After k substitutions

$f(n) = f\left(\frac{n}{2^k}\right) + k$

~~$f(0) = 1$~~

$\frac{n}{2^k} = 0 \longrightarrow$ can't solve

$$f(n) = f\left(\frac{n}{2^k}\right) + k \qquad f(1) = 1$$

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

$$n = 2^k$$

$$k = \log_2 n$$

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

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

Time complex = $O(\log_2 n)$

03. Assumption → Time taken to calculate pow2(a,n) ↓ f(n)

```
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 ;
    }
}
```

pow2(a,n/2) → f(n/2) ; pow2(a,n/2) → f(n/2)

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

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

$f(0) = 1$

$f(1) = 1$

$$f(n) = 2f(n/2) + 1 \qquad 2^1 f\left(\frac{n}{2^1}\right) + (2^1 - 1) \quad \text{— 1}$$

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

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

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

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

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

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

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

$$f(n/8) = 2f\left(\frac{n}{16}\right) + 1$$

$$f(n) = 8 * \left(2f\left(\frac{n}{16}\right) + 1\right) + 7$$

$$f(n) = 16f\left(\frac{n}{16}\right) + 15 \qquad 2^4 f\left(\frac{n}{2^4}\right) + 2^4 - 1 \quad -4$$

After k^{th} iteration

$$f(n) = 2^k f\left(\frac{n}{2^k}\right) + 2^k - 1 \qquad f(1) = 1$$

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

$$n = 2^k$$

$$k = \log_2 n$$

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

$$= n * 1 + n - 1$$

$$= 2n - 1$$

Time complexity = $O(n)$

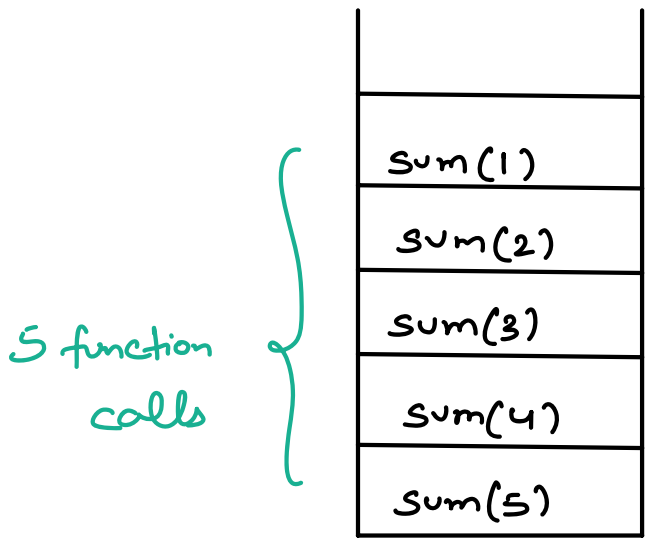
Space complexity

↳ SC : Stack space

↳ Function calls are getting stored in stack so, return stack size.

n=5

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



TC : O(N)

SC : O(N)

02.

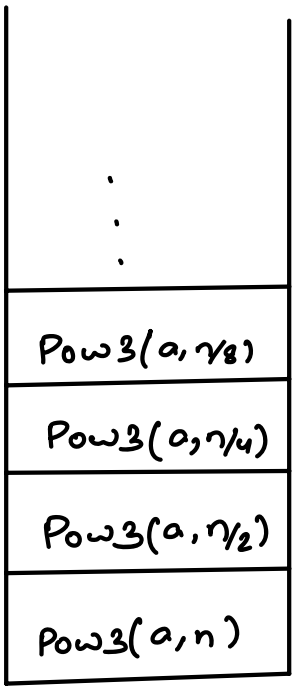
```
int fact(n)
{
    if (n==1) return 1;
    return fact(n-1)*n;
}
```

TC : O(n)

SC : O(n)

03.

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



TC : $O(\log_2 n)$

SC : $O(\log_2 n)$

05.

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

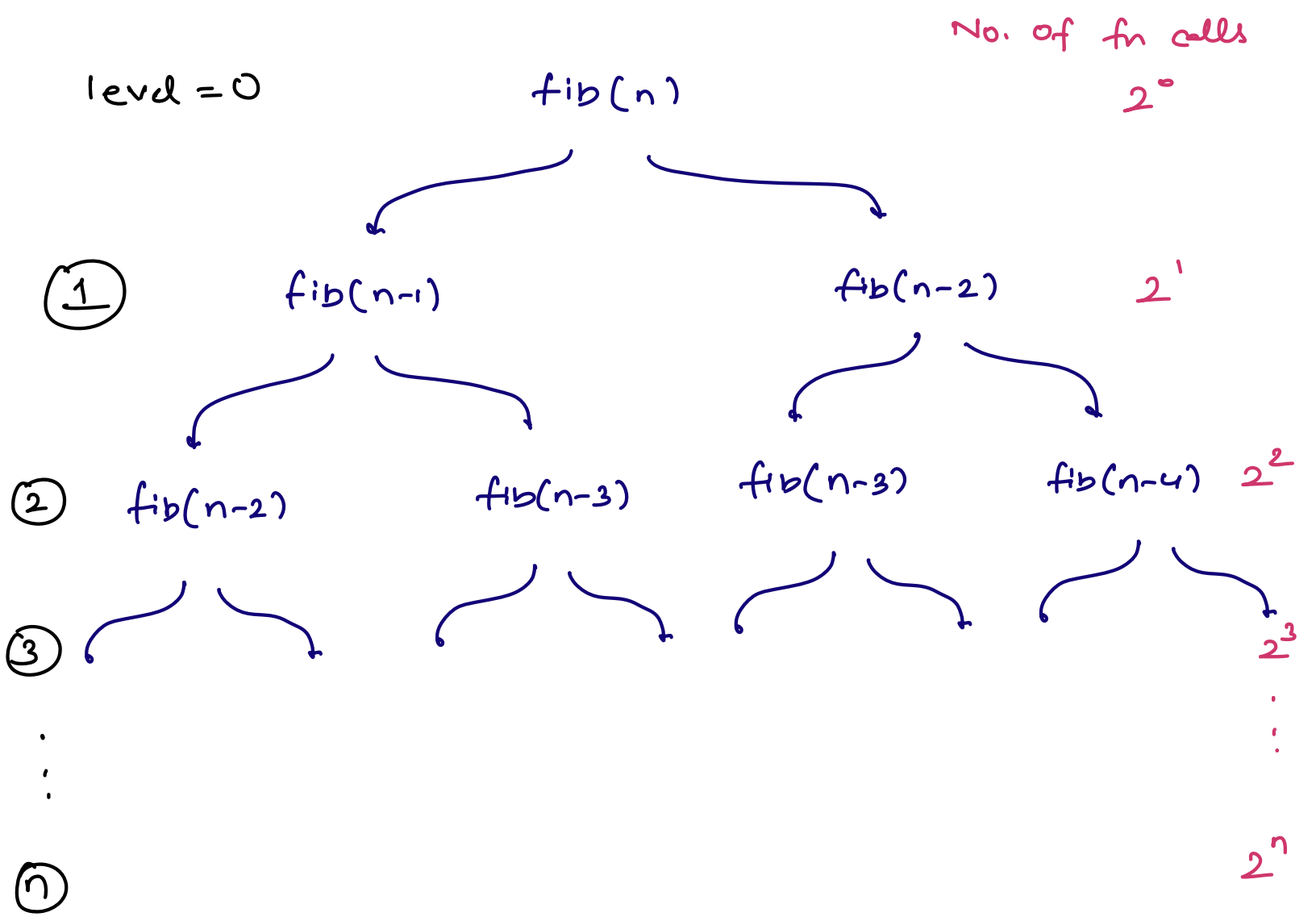
Assumption → Time taken by fib(n) = f(n)

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

$f(n-2) = f(n-3) + f(n-4) + 1$

$f(n-1) = f(n-2) + f(n-3) + 1$

* Not a good Approach → For more than one recursive call



Total no. of function calls = $\underbrace{2^0 + 2^1 + 2^2 + 2^3 + \cdots 2^n}_{GP}$

$a = 2^0 \quad r = 2 \quad n = n+1$

$$sum = \frac{a(r^n - 1)}{r - 1} = \frac{1 * (2^{n+1} - 1)}{2 - 1}$$

$$= 2^{n+1} - 1$$

Time complexity = $O(2^n)$

TODO → Space complexity = $O(n)$