Welcome :)

Agenda : Recursion 2

2-3 problems

T.C of recursion

S.C

Q1 Given a $N \geq 0$, find sum of digits using recursion

eg: sum(123) = 6
sum(257) = 14

$\Rightarrow$ TODO

Q2 Given a, n. Find a^n using recursion.

eg:

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

$$a^n = \overbrace{a * a * a * a - \cdots - a * a}^{\#\ n \text{ times}}$$

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

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

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

Code

```
int pow1( a, n)   Ass : given a,n , return a^n
{
   if( n==0) return 1
   return pow(a,n-1) * a
}
```

```
int pow2 ( a,n) ass : same as above
{
   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          ]
   }
}
```

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

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

$a^{11} = a^5 * a^5 * a$

$a^{13} = a^6 * a^6 * a$

if(n is even)

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

if (n is odd)

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

obs: Calculate pow(a, n/2) once and store it.

```
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
}
```

return 512 = 16*16*2

```
int pow3(a, n)   a=2  n=9
{
    if(n==0) return 1;
    p = pow3(a, n/2);
    if(n%2 ==0) return p*p
    else return p*p*a
}
```

↓ n=4   ↑ 16

```
int pow3(a, n)   n=4
{
    if(n==0) return 1;
    p = pow3(a, n/2);
    if(n%2 ==0) return p*p
    else return p*p*a
}
```

↓ n=2   ↑ 4

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

↓ n=1   ↑ 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
}
```

↓ n=0   ↑ 1

```
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
}
```

Stack Tracing

TODO

Q3 Given a, n, m calculate $a^n \% m$

constraints

$1 \leq a \leq 10^9$

$1 \leq n \leq 10^9$

$1 \leq m \leq 10^9$

Ass: calculate & return $a^n \% m$

```
modpow( a, n, m)
{
   if (n==0) return 1;
   ~~int~~ long p = modpow( a, n/2, m);
   if( n%2 ==0)  // even
   {
       return (p*p) %m
   }
   else {
       return (p*p*a) % m        wrong.

       return ((p*p)%m *a)%m
   }
}
```

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

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

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

$\simeq 10^9$ ✓

$10^9 * 10^9$

$\simeq 10^{18} * 10^9$

$= 10^{27}$

✗ cannot store in long variable

Fast Exponentiation

```
long powmod ( a, n, 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
    }
}
```

T.C of recursive code using Recursive Relation

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

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

Time taken to calculate sum(N) = f(N)

$f(N) = f(N-1) + 1$ → $O(1)$ (for all extra time)

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

$= f(N-2) + 2$

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

$= f(N-3) + 3$

⋮

Generalize

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

$N - k = 1$

$k = N-1$

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

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

$= f(1) + N-1$

$= 1 + N - 1$

$f(N) = N \implies O(N)$

#2

```
int fact(N)        T.C => O(N)
{
  if( N==1 ) return 1
  return fact(N-1) * N ;
}
```

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

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

#3

```
int pow1( a, n)        T.C => O(N)
{
  if( n==0 ) return 1
  return pow(a,n-1) * a
}
```

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

#3

```
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 ]
  }
}
```

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

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

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

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

$$= 8f(N/8) + 7$$

$$f(N/8) = 2f(N/16) + 1$$

$$= 16f(N/16) + 15$$

After k substitutions

$$f(N) = 2^k f(N/2^k) + 2^k - 1$$

$f(0) = 1$

$f(1) = 1$

$$N/2^K = 1$$

$$2^K = N$$

applying log

$$K = \log_2 N$$

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

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

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

$$= N + N - 1 \Rightarrow O(N)$$

#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
}
```

// Time taken by pow3(a, n) = f(N)

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

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

$$= f(N/4) + 2$$

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

$$= f(N/8) + 3$$

after K substitutions

$$f(N) = f\left(\frac{N}{2^K}\right) + K$$

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

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

$$\Rightarrow \quad N = 2^K$$

$$K = \log_2 N$$

$$f(N) = f\left(\frac{N}{N}\right) + \log_2 N$$

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

#5

```
long powmod(a, n, 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
    }
}
```

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

$$T.C \Rightarrow O(\log_2 N)$$

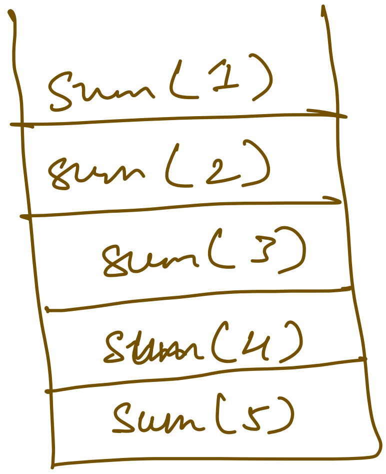
Space Complexity for recursive code.

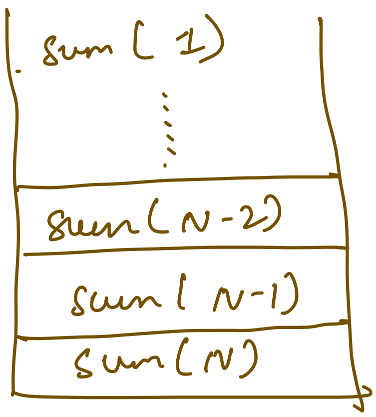
① We use space, b/c we store function calls.

S.C ⇒ Our stack size

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

S.C ⇒ O(N)





#2

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

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

S.C ⇒ O(N)

#3

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

S.C O(N)

#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
}
```

pow(a, 0)
pow(a, 1)
⋮
pow(a, n/4)
pow(a, n/2)
pow(a,n

Stack size = log N

SC => log N

#5 fib

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

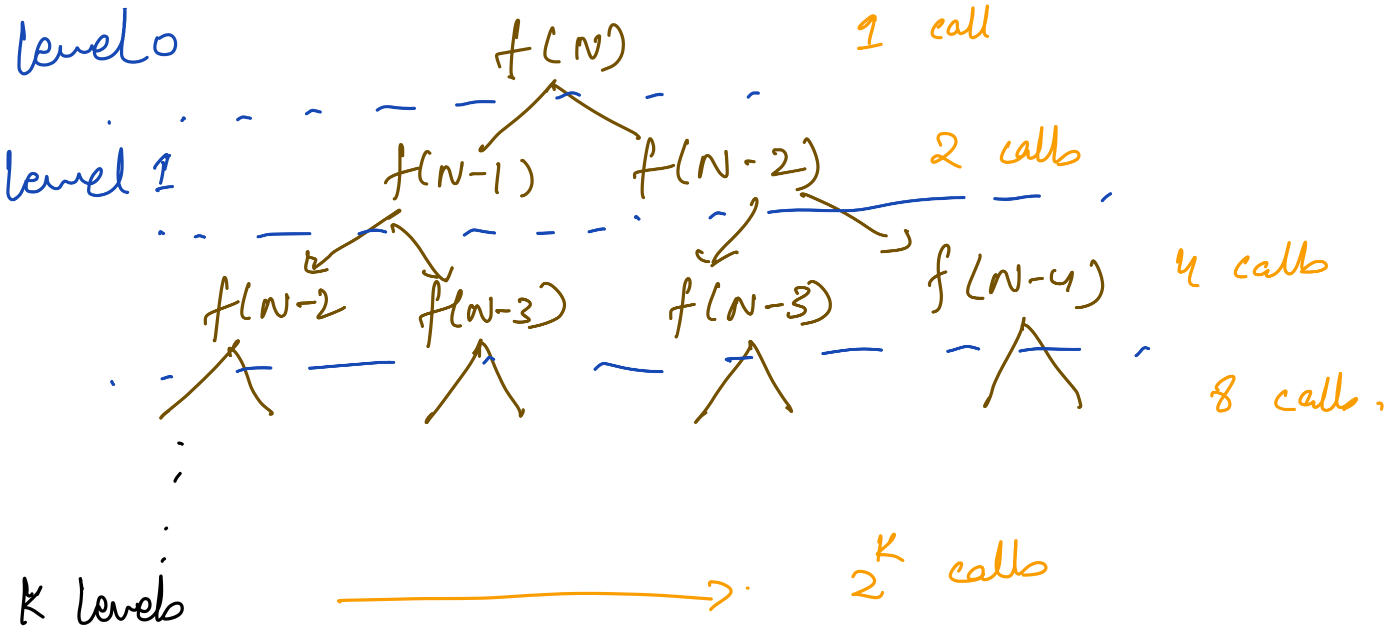
$$f(N) = f(N-1) + f(N-2) + 1$$

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

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

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

Note: solving using substitution is tricky.



Total func^n calls $\Rightarrow$ $1 + 2 + 2^2 + 2^3 + 2^4 \ldots 2^N$

$\Rightarrow 2^{n+1} - 1$

$\Rightarrow 2 \times 2^n - 1$

$\Rightarrow O(2^n)$

T.C $\Rightarrow O(2^n)$

S.C $\Rightarrow O(N)$