Todays Content:

- pow(a, n) {
  | // way1, // way2, // way3
  }
- pow(a, n, p) { - }
- TC of recursive Codes
- SC of recursive Codes

2Q) Given a, n find a^n using recursion (No Overflows)

Eg1:

a	n	a^n
2	5	32
3	4	81

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

n times

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

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

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

n >= 0

```
int pow2(a, n){ Ass: Calculate & return a^n
    if(n==0){ return 1;}
    if(n%2==0){
        return [ pow(a, n/2) * pow(a, n/2) ]
    }
    else{
        return [ pow(a, n/2) * pow(a, n/2) * a ]
    }
}
```

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

$\hookrightarrow a^5 * a^5$

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

$a^{11} = a^5 * a^6$

$\hookrightarrow a^5 * a^5 * a$

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

a^n : n is even

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

$pow(a, n/2)$

n is odd

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

$pow(a, n/2)$

Obs: Same problem, calling only once

```
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

```
int pow3(a=2, n=9){                                   → return: 512
  ✓ if(n==0){ return 1;}
  ✓ int p = pow3(a, n/2);   // p = 16
  ✓ if(n%2==0){ return p*p; }
  ✓ else { return p*p*a; }
}
                                  16*16*2
```

return 16

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

return 4

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

return 2

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

return 1

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

Q) Given a, n, m calculate $a^n \% m$

Constraints:

$1 \le a, x \le 10^9$

$1 \le n, x \le 10^9$

$2 \le m, x \le 10^9$

Note: Take care of overflows

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

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

$= [\underbrace{a^{n/2} \% m}_{} * \underbrace{a^{n/2} \% m}_{}] \% m$

powmod(a, n/2, m)

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

```
powmod(int a, int n, int m){
    if(n==0){return 1;}
    long p = powmod(a, n/2, m);   // p = a^(n/2) % m   // it will break down further & we won't be getting any overflow here
                                  // At max p ≈ m-1  p = 10^9
    if(n%2==0){
        return (p*p)%m;           // we cannot store in int
    }                             // ≈ 10^9 * ≈ 10^9 = [10^18] % m ≈ 10^9
    else{
        return (p*p*a)%m;         // we cannot store in long
                                  // ≈ 10^9 10^9 10^9 → [10^27] % m
        return (p%m * p%m * a%m)%m;
                                  // ≈ 10^9  10^9  10^9 → 10^27 % m
        return [(p*p)%m * a]%m;
                                  // ≈ [10^9 * 10^9] % m
                                  // ≈ 10^9 * 10^9 % m = ≈ 10^9
    }
}
```

```
// Fast exponentiation
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 Code using Recusive Relation // function of n

```
int sum(N){ // Time taken to calculate Sum(n) = f(n) = O(N)
    if(N==1){return 1}
    return (sum(N-1) + N)
}
```

sum(N-1) → f(N-1)

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

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

$= f(N-2) + 2$

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

$= f(N-3) + 3$

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

$= f(N-4) + 4$

// After k steps:

$f(N) = f(N-k) + k, \quad f(1) = 1$

// $N - k = 1 \Rightarrow N = 1 + k \Rightarrow k = N-1$

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

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

```
int fact(N){ // Time taken to calculate fact(N) = f(N) = O(N)
    if(N==1){return 1}
    return [fact(N-1) * N]
}
```

fact(N-1) → f(N-1)

$f(N) = f(N-1) + 1, \quad f(1) = 1$

```
int pow1(a, n){ // Time take to calculate pow1(a, n) = f(n)
    if(N==0) {return 1;}
    return pow1(a, n-1) * a
}
```
$f(n)$

$f(N) = f(N-1) + 1 \quad f(0) = 1$

TODO $\Rightarrow f(N) \Rightarrow O(N)$

```
int pow2(a, n){ Time taken to calculate pow2(a, n) = f(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 ];
    }
}
```
$f(N/2)$ $f(N/2)$ $f(N/2)$

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

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

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

$= 4f(N/4) + 3 \Rightarrow 2^2 f(N/2^2) + 2^2 - 1$

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

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

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

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

$= 8[2f(N/16) + 1] + 7$

$= 16f(N/16) + 15 \Rightarrow 2^4 f(N/2^4) + 2^4 - 1$

// After k steps

$f(N) = 2^k f(N/2^k) + 2^k - 1, \quad f(0) = 1 \quad f(1) = 1$

// $N/2^k = 1 \leftrightarrow N = 2^k \leftrightarrow k = \log_2 N$

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

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

```
int pow3(a, n){ // Time taken to calculate pow3(a,n) = f(n)
    if(n==0){return 1;}
    p = pow3(a, n/2); } T(N/2)
    if(n%2==0){return p*p;}
    else{return p*p*a;}
}
```

$f(N) = f(N/2) + 1$, $f(0) = 1$, $f(1) = 1$

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

$= f(N/4) + 2 \Rightarrow f(\frac{N}{2^2}) + 2$

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

$= f(N/8) + 3 \Rightarrow f(N/2^3) + 3$

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

$= f(N/16) + 4 \Rightarrow f(N/2^4) + 4$

TC: $O(\log_2 N)$

After k steps:

$f(N) = f(N/2^k) + k$ $\quad f(0) = 1$ $\quad f(1) = 1$

$\frac{N}{2^k} = 0$ * $\quad \frac{N}{2^k} = 1 \Rightarrow N = 2^k \Rightarrow k = \log_2 N$

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

$= f(1) + \log_2 N$

$= 1 + \log_2 N \Rightarrow O(\log_2 N)$

```
int powmod(int a, int n, int m){ // pow(a, n, m) = f(N)
    if(n==0){return 1;}
    long p = powmod(a, n/2, m); f(N/2)
    if(n%2==0){ return (p*p)%m;}
    else{ return [(p*p)%m * a]%m;}
}
```

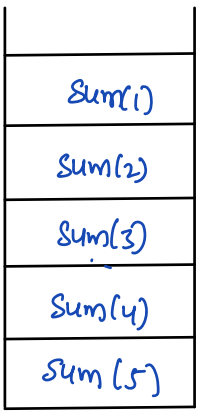
$f(N) = f(N/2) + 1$

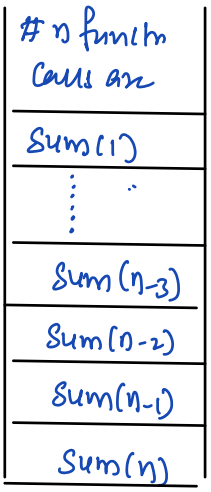
$f(N) = \log_2 N$

Space Complexity for Recursive Code ?

- obsl: function calls stored in Stack, hence it will extra space
- SC: It is stack size

```
int sum(N){ SC:O(N) TC:O(N)
    if(N==1){return 1}
    return (sum(N-1)+N)
}
```






```
int fact(N){ SC:O(N) TC:O(N)
    if(N==1){return 1}
    return (fact(N-1)*N)
}
```

```
int pow1(a,n){ SC:O(N) TC:O(N)
    if(N==0){return 1}
    return pow1(a,n-1)*a
}
```

```
int pow3(a, n){   SC: O(log N)   TC: O(log N)
    if(n == 0){ return 1 }
    p = pow3(a, n/2)
    if(n % 2 == 0){ return p*p }
    else{ return p*p*a }
}
```

//log N Calls
pow(a, 0)
⋮
pow(a, n/8)
pow(a, n/4)
pow(a, n/2)
pow(a, n)

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

Time taken to calculate fib(N) = $f(N) = O(2^n)$

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

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

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

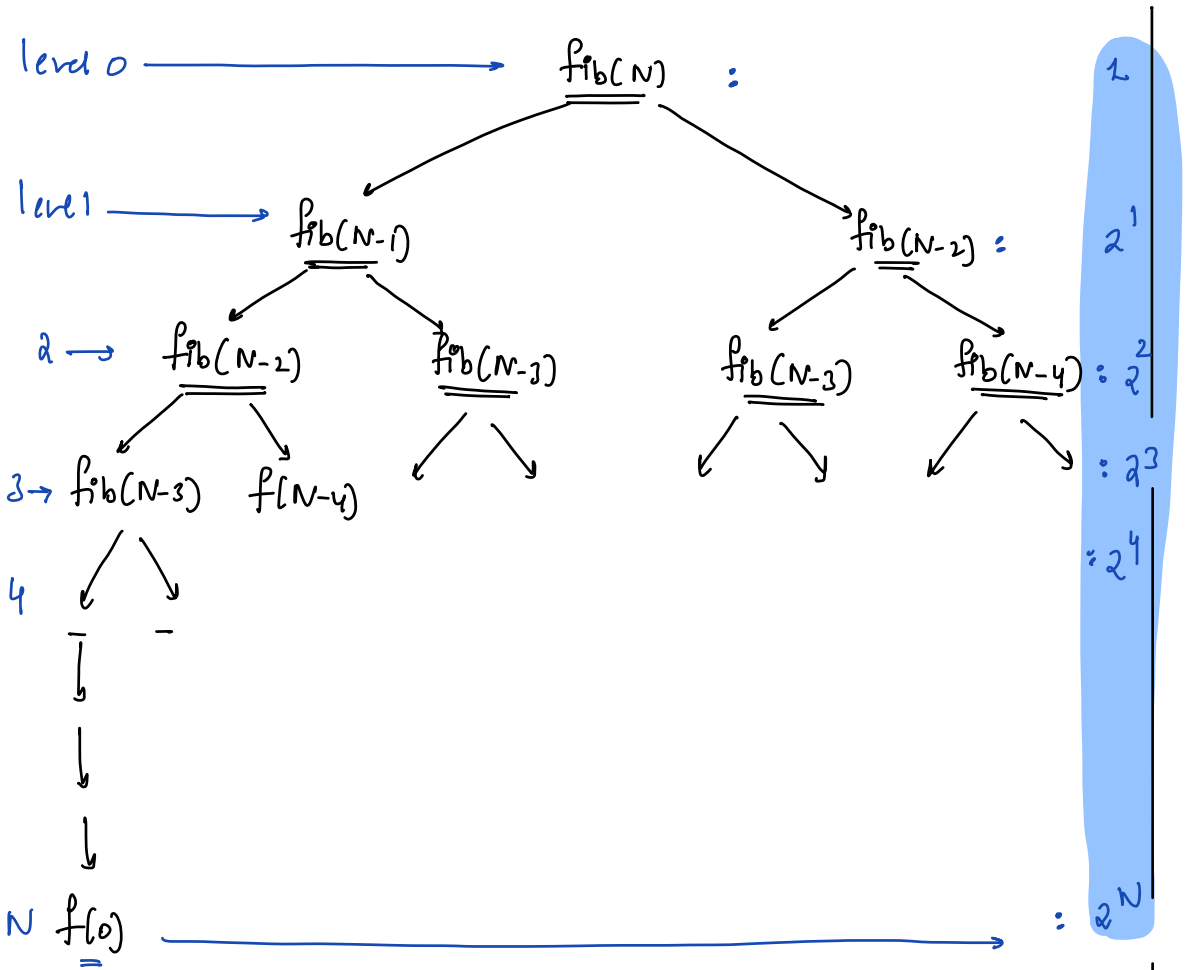
$f(N) = f(N-1) + f(N-2) + 1$, not a good approach

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

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

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

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



Total function calls = $\underbrace{2^0 + 2^1 + 2^2 + 2^3 + \cdots 2^N}$

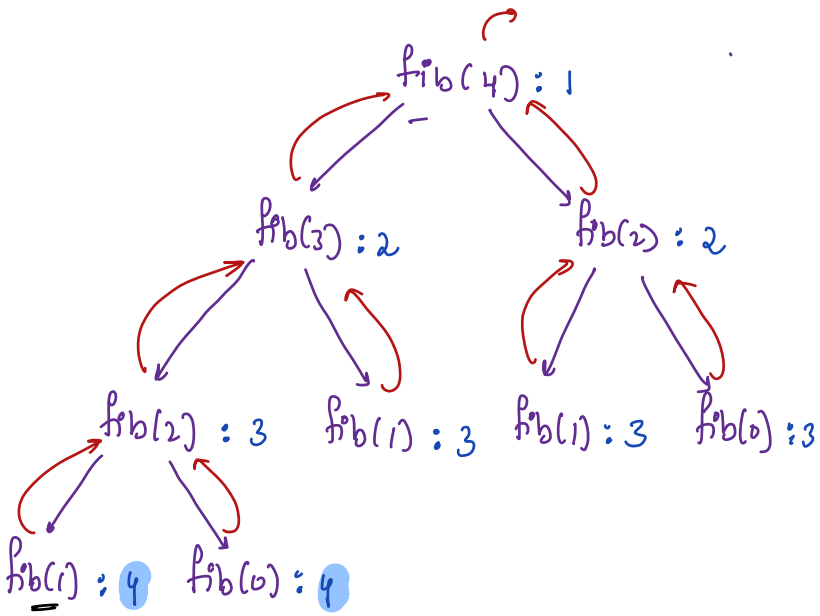
G.P, $a=1$, $r=2$, $t = n+1$

$$\Rightarrow \frac{a \times [r^t - 1]}{r-1} \Rightarrow \frac{1 * [2^{n+1} - 1]}{2-1}$$

$$\Rightarrow \frac{2^{n+1} - 1}{1} \Rightarrow [2^n][2] - 1 \Rightarrow O(2^n)$$

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

Space Complexity : $O(N)$ — at max we will have N function calls on stack



obs: Max stack size = 4