Today's Content

- pow(a,n) {

  //way1, //way2, //way3

  }
- pow(a,n,p) { — }
- SC/TC of recursive codes.

Qn ⇒ Given a, n find a^n using recursion.

eg

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

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

Main Logic

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

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

$a^n =$ pow1 (a,n-1) * a

// Why write base condition?

* To stop the recursion.

```
int pow2(a,n) { // Assumption: Calc & return 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]
    }
}
```

* Same subproblem, call it once

Main Logic

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

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

Pow is odd?

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

↳ $a^5 * a^5 * a$

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

a^n: n is even ⇒ $a^{n/2} * a^{n/2}$

n is odd ⇒ $a^{n/2} * a^{n/2} * a$

Subproblem

pow2(a,n/2)

```
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)
    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)
    if(n%2==0) {
        return p*p
               4* 4=16
    }
    else {
        return p*p*a
    }
}
```

return 4

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

return 2

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

Qn: Given a, n, m Calc $(a^n) \% m$

Note: Take care of overflows.

Constraints

$1 <= a <= 10^9$

$1 <= n <= 10^9$

$2 <= m <= 10^9$

Assumption: Calc & return $(a^n) \% m$

Main Logic

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

$\downarrow$

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

```
powmod (int a, int n, int m) {
    return pow3(a, n) % m   → Wrong! Overflow!!

long p = powmod(a, n/2, m)   // p : a^{n/2} % m
                                p : [0 m-1]
                                pmax ≈ 10^9
if(n % 2 == 0) {
    return (p * p) % m
}
else {
    return (p * p * a) % m
}
}
```

$(p*p)\%m$: $10^9 * 10^9 = 10^{18} \rightarrow [10^{18}] \% m \Rightarrow (10^9 = int)$

$(p*p*a)\%m$: $10^9 * 10^9 * 10^9 = [10^{27}] \% m$

$[\underbrace{p \% m}_{10^9} * \underbrace{p \% m}_{10^9} * \underbrace{a \% m}_{10^9}] \% m$

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

$[[10^{18}]\%m * 10^9]\%m$

$\Rightarrow [10^{9^2} * 10^9] \% m$

$\Rightarrow [10^{18}] \% m$

$= 10^9$

```
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
    }
}
```

Break: 8:01 AM $\Rightarrow$ 8:11 am

TC for Recursive Code using Recurrence Relations

time taken to calc sum(N) = f(N)

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

$\underline{sum(N-1)}$ → $f(N-1)$

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

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

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

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

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

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

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

After K steps :

$$f(N) = f(\underline{N-K}) + K \quad , \quad f(1) = 1$$

$N - K = 1$

$\boxed{K = N-1}$

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

$$= 1 + N - 1$$

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

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

$f(N-1)$

// Time taken to compute fact(N) = $f(N)$

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

TC: $O(N)$

// Time taken to compute pow1(a,n) = $f(n)$

$f(0) = 1$

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

$$\boxed{f(n) = f(n-1) + 1}$$

// To do

```
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 to calc pow3(a,n) = $f(n)$

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

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

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

$\hookrightarrow f(n/8) + 1$

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

$f(1) = f(0) + 1$
$= 2 \Rightarrow O(1)$

After K steps

$f(n) = f\left(\frac{n}{2^K}\right) + K$, $f(0) = 1$, $f(1) = 1$

$\frac{n}{2^k} = 0$ $\Rightarrow n = 0$ Log(0) = undef

$\frac{n}{2^K} = 1 \Rightarrow \boxed{n = 2^K}$ or $\boxed{K = \log_2 N}$

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

* 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
    }
}
```

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

$f(n/2)$

$f(n) = f(n/2) + 1$
$= O(\log n)$

```
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
    }
}
```

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

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

// Time taken to compute pow2(a,n) = f(n)

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

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

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

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

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

$2^2 f(n/2^2) + 2^2 - 1$

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

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

$2^3 f(n/2^3) + 2^3 - 1$

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

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

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

$2^4 f(n/2^4) + 2^4 - 1$

After K steps:

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

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

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

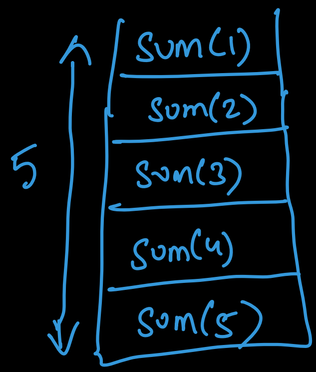
$= n \times f(1) + n - 1$

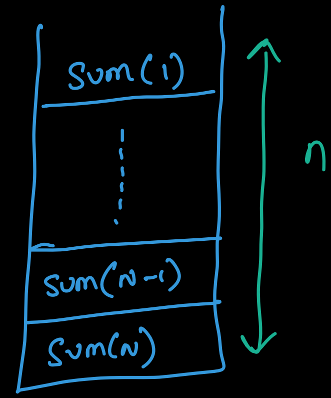
$= 2n - 1$

$= O(n)$

Space complexity for Recursive Code?

* Obs1: Function calls are stored in a stack.
* SC = Size of stack.

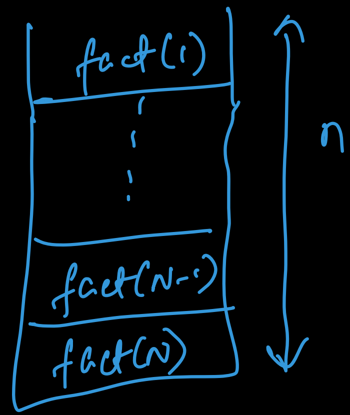





```
int sum(N) {
    if(N==1) {return 1}
    return sum(N-1) + N
}
```

SC: O(n)

TC: O(n)




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

SC: O(n)

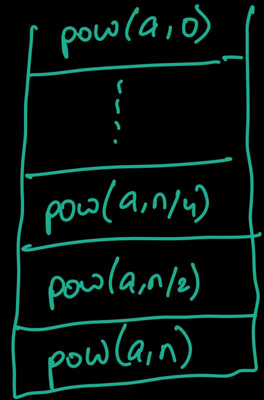
TC: O(n)

SC: O(n) Stack size = N

TC: O(n)

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

```
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 }
}
```



log N

SC : $O(\log N)$

TC : $O(\log N)$

```
int fib(N) {   // Time to calc fib(N) = f(N)
    if(N <= 1) { return N }
    return fib(N-1) + fib(N-2)
}
```

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

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

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

$$f(n) = f(n-1) + f(n-2) + 1$$ ⇒ not a good approach.

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

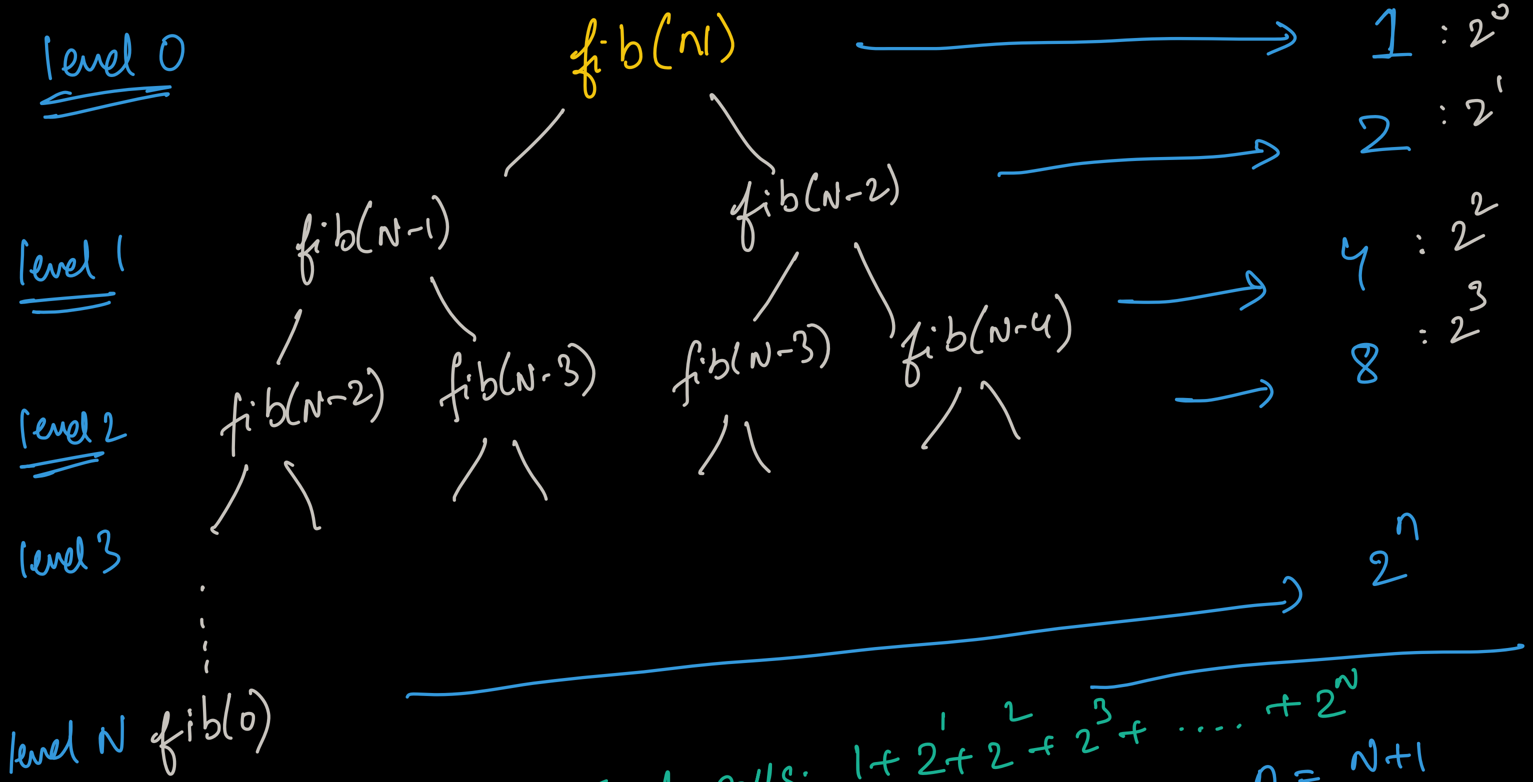
$$f(n-2) = f(n-3) + f(n-4) + 1$$

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

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

↳ complexity is inc.

$$= f(n-2)$$



Total calls: $1 + 2^1 + 2^2 + 2^3 + \dots + 2^N$ $\quad n = N+1$

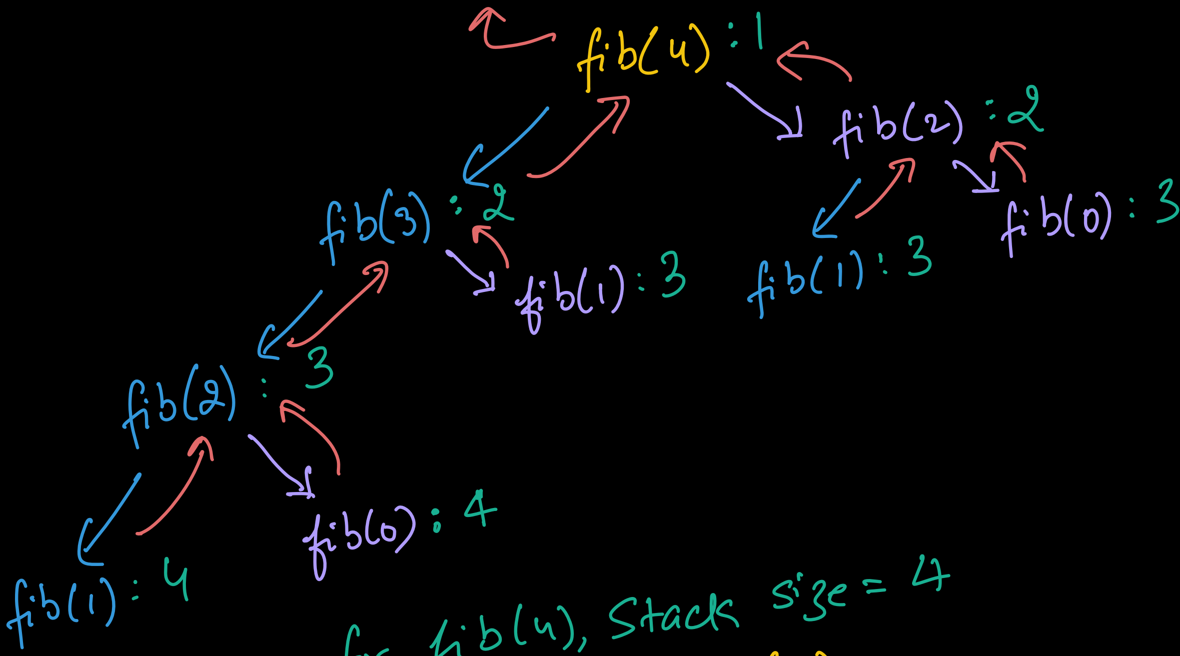
$a = 1, \; r = 2$

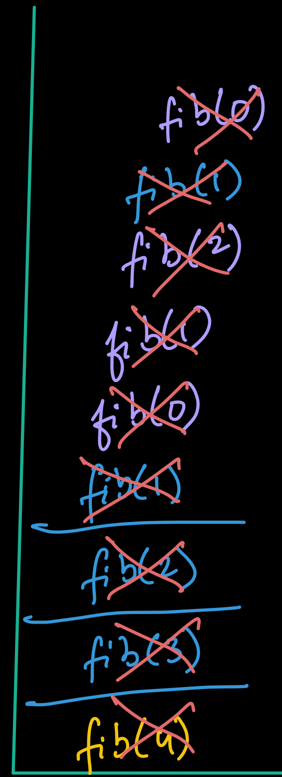
$$\frac{a(r^n - 1)}{r - 1} = \frac{1(2^{N+1} - 1)}{2 - 1}$$

$$= 2^{N+1} - 1 \implies O(2^N)$$

Space complexity?

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





for fib(4), Stack size = 4

∴ fib(n), SC: O(n)

Doubts: Subarr with (OR = 1).