Q1 pow(a, n) a^n — Dont worry about overflow

$\hookrightarrow n \geq 0$

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

pow(a,n)   pow(a,n-1)   a a a a _ _ _ a a

int pow (int a, int n) {

Assumption: return the value of a^n

Base case: if (n==0) return 1

return a * pow(a, n-1)

}

$TC = O(n)$

$SC = O(n)$

n = 5
n = 4
n = 3
n = 2
n = 1
n = 0

depth = 5 + 1

$a^{10} = a \times a^9$

$a^{10} = a^5 \times a^5$

$a^{10} = a^2 \times a^8$

$a^{10} = a^3 \times a^7$

$a^{10} = a^9 \times a^1$ ✗

$a^{10} = a^5 \times a^5$

$a^{11} = a^5 \times a^5 \times a$

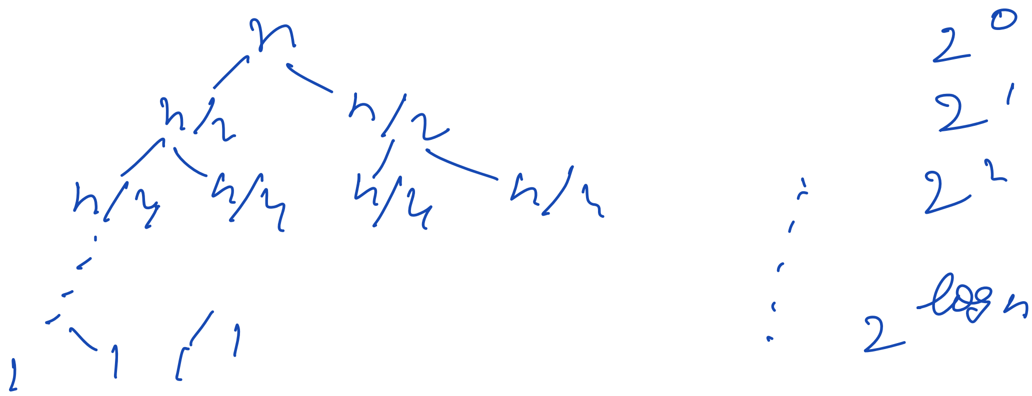
$a^{14} = a^7 \times a^7$

$a^{19} = a^9 \times a^9 \times a$

$a^{16} = a^8 \times a^8$

```
int pow( a, n) {
    if (n==0) return 1
    if (n%2 ==0) // a^n = a^(n/2) x a^(n/2)
      return pow(a,n/2) * pow(a,n/2)

    else { // a^n = a^(n/2) x a^(n/2) x a
      return pow(a,n/2) * pow(a,n/2)
                         * a.
    }
}
```



$2^0 + 2^1 + 2^2 + \cdots + 2^{\log n} \Rightarrow 2n - 1$

TC: $O(n)$

SC: $O(\log(n))$

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

  int p = pow (a, n/2)
  if (n%2 ==0)
      return p×p
  else
      return p×p×a
}
```

//Binary exponentiation

1^n

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

```
main() {
print (pow (2, 10))
}
```

depth $= \log(n)$

pow(2,10)

$p = 32$

ans $= 32 \times 32$

1024

pow(2,5)

$p = 4$

ans $= 4 \times 4 \times 2$

$= 32$

pow (2, 2)
p = 2
ans = 4

pow (2, 1)
p = 1
ans = 2

pow (2, 0)
ans = 1

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

  int p= pow (a, n/2)
  if (n%2 ==0)
      return pxp
  else
      return pxpxa
}
```

TC: $O(\log(n))$

SC: $O(\log(n))$

Q2 Given a, n, m find $a^n \% m$

Constraints:

$1 \le a \le 10^5$

$0 \le n \le 10^6$ → ?

$1 < m \le 10^9$

Make sure no overflows occur.

Ex

a	n	m	→ $a^n \% m$	
2	5	5	$(2^5) \% 5$	$= 2$
3	4	7	$(3^4) \% 7$	$= 4$

$a^n \% m$ ⇒ pow(a, n, m)

if (n is even)

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

Use modulo rule

$(a \times b) \% m$

$((a \% m) \times (b \% m)) \% m$

$[(a^{n/2} \% m) \times (a^{n/2} \% m)] \% m$

pow(a, n/2, m) ⇒ p

$(p \times p) \% m$

n = even

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

n = odd

```
int pow( int a, int n, int m) {
Assumption: returns value a^n % m

   if(n==0)  return 1
   if (a==1)  return 1
   long p = pow(a, n/2, m)
   if (n%2 == 0) {
      return  (p*p) % m
   }
   else {
      return  (p×p×a) % m
      [(p×p) % m  ×  (a % m)] % m

   }
}
```

$$\text{depth} = \log(n)$$

TC in recursion

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

Define $T(N)$ → Number of iterations required to find sum(N)

For $sum(N) \longrightarrow T(N)$

$sum(N-1) \longrightarrow T(n-1)$

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

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

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

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

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

Generalize

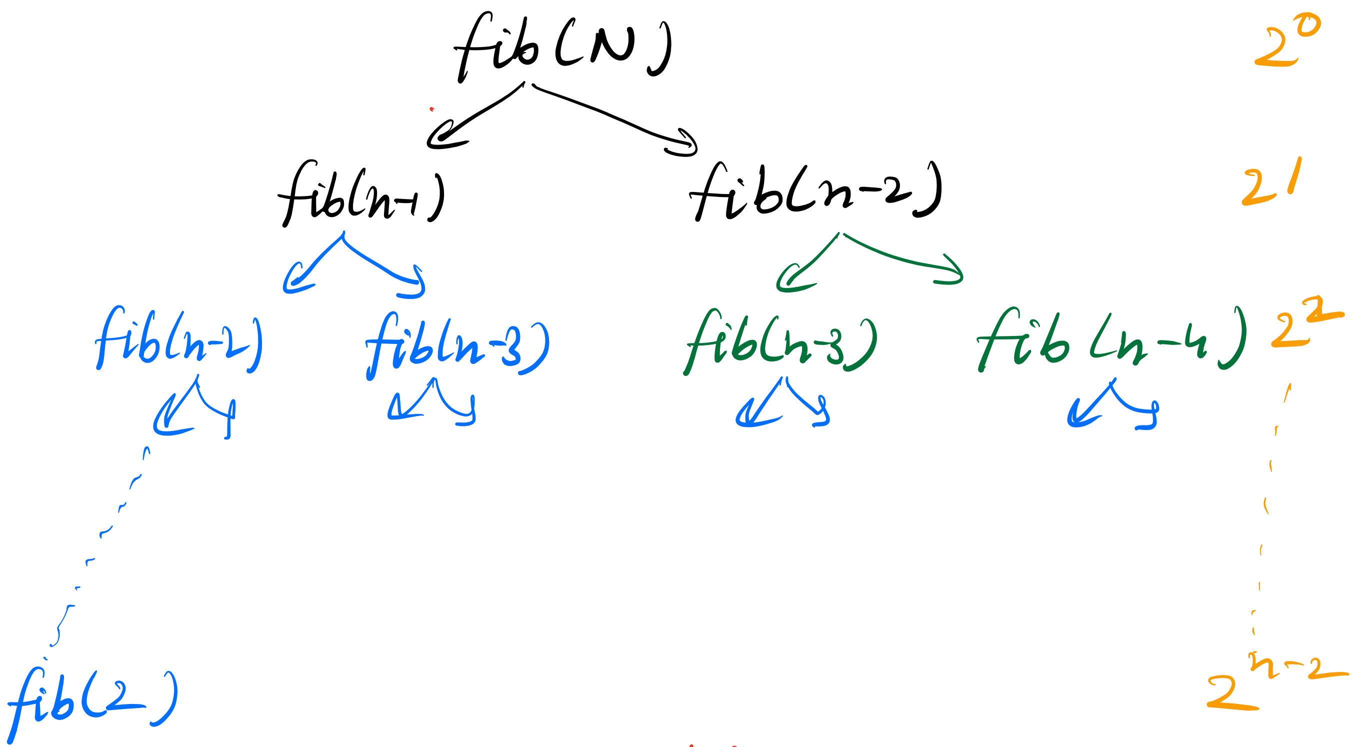
$$T(N) = T(N-k) + k \qquad T(1) = 1$$

$$n - k = 1 \qquad k = n-1$$

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

TC: $O(n)$

Recursion Tree



depth = n-1

$$2^0 + 2^1 + 2^2 + \cdots + 2^{n-2}$$

$a = 1 \qquad r = 2 \qquad N = n-1$

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

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

TC: $O(2^n)$

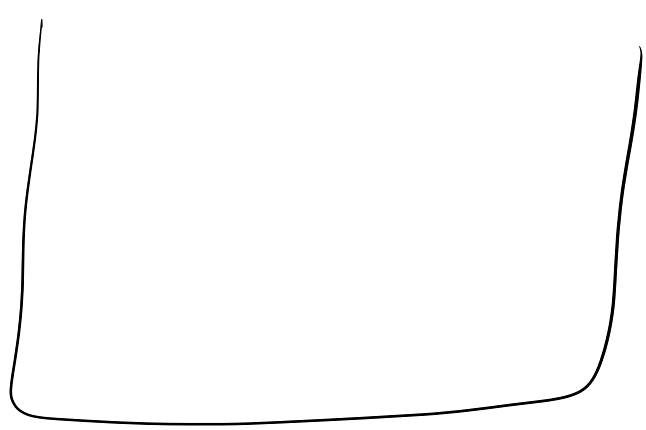
SC in recursion

space is taken in the call stack.

Max stack size across the whole execution of code is the Space complexity.

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

main()
  print (fib(5))
```



Recursion tree (best)

TC: Number of nodes in tree

SC: Max height (depth) of the tree.

1) Draw recursion tree

{done}

$$(pf(e) - pf(s-1)) \% 2 = 0$$

$$pf(e) \% 2 = pf(s-1) \% 2$$