Q → Find a^b using recursion.

$a = 2 \quad b = 3 \quad 2^3 = 8$

↳ $2 * 2 * 2$

$x^n = x * x * x \ldots$ (n times)

$2^4 = 2 * \underbrace{2 * 2 * 2}_{2^3} * 2$

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

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

TC = O(b) SC = O(b)

$2^6 = 2^3 * 2^3$

$2^7 = 2 * 2^3 * 2^3$

$x^n \Rightarrow x^{n/2} * x^{n/2}$, n → even

↳ $x * x^{n/2} * x^{n/2}$, n → odd

2^{10} → $x^n = x * x^{n-1}$ ⇒ 10 steps

$2^{10} = 2^5 * 2^5 = 1024$ (32, 32)

$2^5 = 2 * 2^2 * 2^2 = 32$ (4, 4)

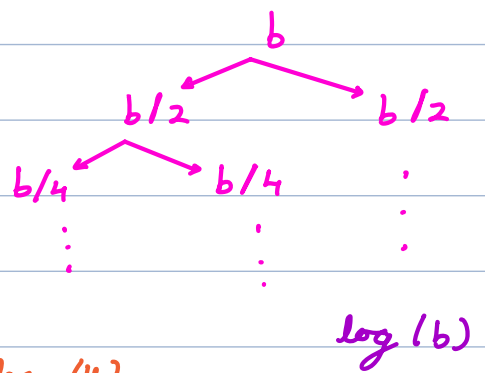
$2^2 = 2^1 * 2^1 = 4$ (2, 2)

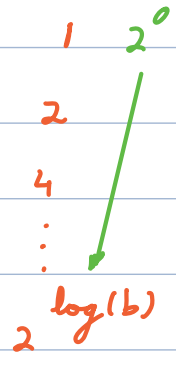
$2^1 = 2 * 2^0 * 2^0 = 2$ (1, 1)

4 steps

```
int pow (a, b) {
    if (b == 0) return 1
    if (b % 2 == 0)
```

```
    return pow(a, b/2) * pow(a, b/2)
  else
    return a * pow(a, b/2) * pow(a, b/2)
}
```





$[0 \quad \log(b)]$

$$\frac{1\,(2^{\log b + 1} - 1)}{2 - 1}$$

$$= 2 * 2^{\log b} - 1$$

$$= 2 * b - 1$$

$TC = O(b)$

$x^{\log_x(y)} = y$

$2^{\log_2(8)} = 8$

$SC = O(\log(b))$ ✓

```
int pow(a, b) {                    // Fast Power  v. imp.
  if (b == 0) return 1
  p = pow(a, b/2)
  if (b % 2 == 0)   return p * p
  else  return a * p * p
}
```

HW → Try iterative code.

$TC = O(\log(b))$

$SC = O(\log(b))$

```
pow(2, 9) {   // 512
  p = pow(2, 4) {   // 16
    p = pow(2, 2) {   // 4
      p = pow(2, 1) {   // 2
        p = pow(2, 0) {   // 1
          return 1
        }
```

return 2*1*1 = 2
}
return 2*2 = 4
}
return 4*4 = 16
}
return 2*16*16 = 512
}

$N \to N/2 \to N/4 \to \dots \frac{N}{2^k} = 1$

$\Rightarrow N = 2^k$

$= k = \log(N)$

Q → Print array elements using recursion.

i
0 1 2 3
A = [3 5 1 8] o/p → 3 5 1 8

0

```
void printArray (A[], i) {
    if (i == A.length) return
    print (A[i])
    printArray (A, i+1)
}
```

TC = O(A.length)

SC = O(A.length) → O(1)

Tail Recursion → If recursive call is last step (some languages) then function calls are not stored.

Q → Find max element of array using recursion.

i
0 1 2 3
A = [3 5 1 8] o/p → 8

```
int maxArray(A[], i) {        // i → 0
    if (i == A.length) return Int_Min
    return max(A[i], maxArray(A, i+1))
}
```

TC = O(A.length)

SC = O(A.length)

Q → check if the given string is palindrome.

str = reverse(str)

Eg → racecar Ans = true

race Ans = false

```
boolean isPalindrome(s, l, r) {        // l → 0, r → N-1
    if (l >= r) return true
    if (s[l] != s[r]) return false
    return isPalindrome(s, l+1, r-1)
}
```

TC = O(N) SC = O(N) → O(1)

(tail recursion)

Tower of Hanoi

Given 3 towers (A, B & C) & N disks of different sizes.

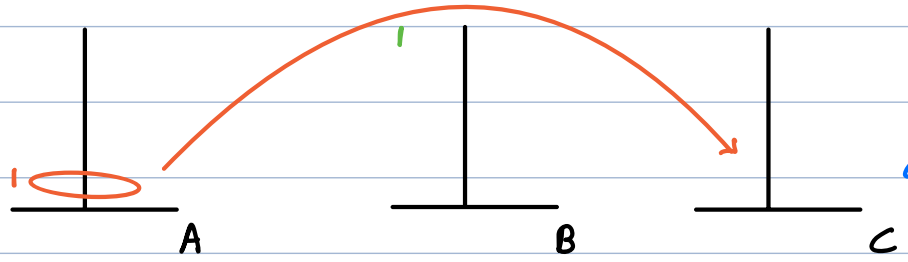
Move all disks from A to C.

Constraints → 1) In step only 1 disk can be moved.

2) large disk cannot be placed over small disk.

Goal → Use minimum steps.

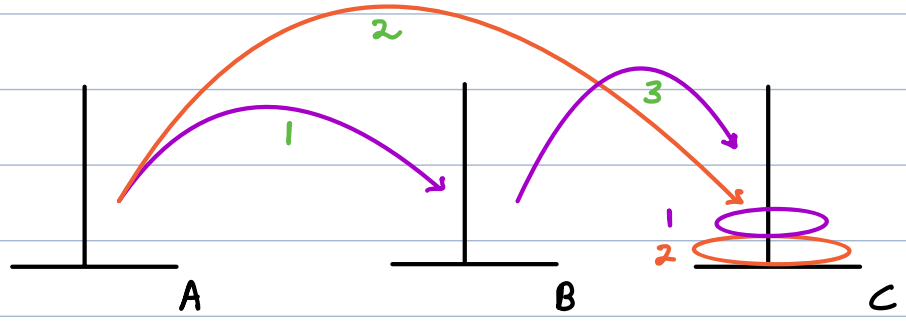
N = 1



steps = 1

o/p = 1, A → C

N = 2



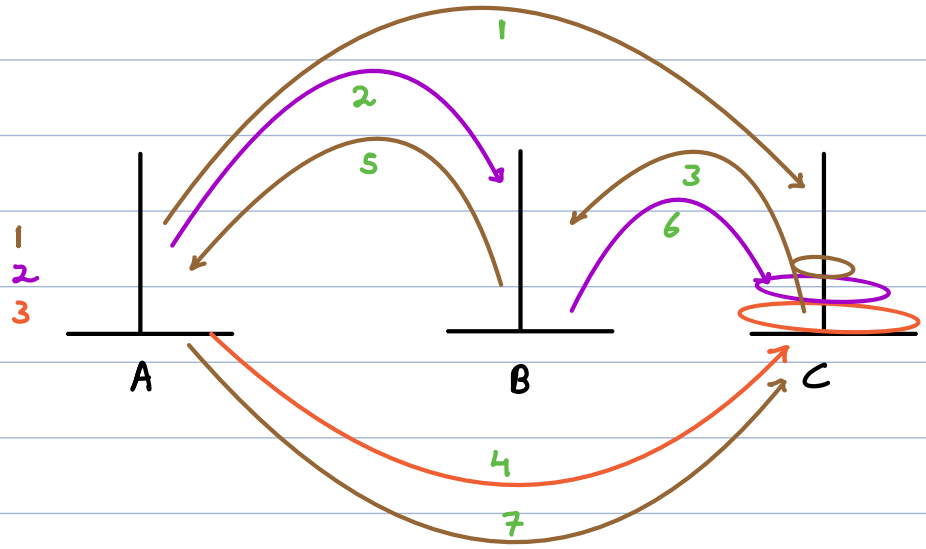
steps = 3

o/p = 1, A → B

2, A → C

1, B → C

N = 3



steps = 7

o/p → 1, A → C

2, A → B

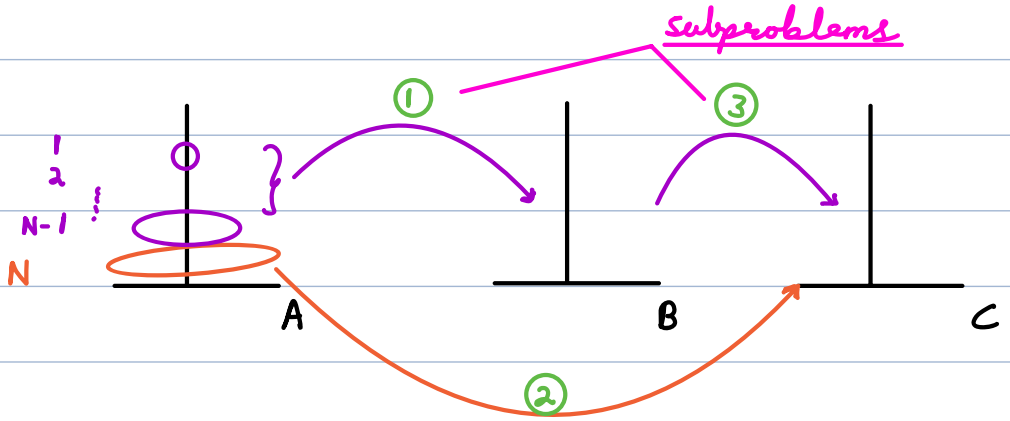
1, C → B

3, A → C

1, B → A

2, B → C

1, A → C

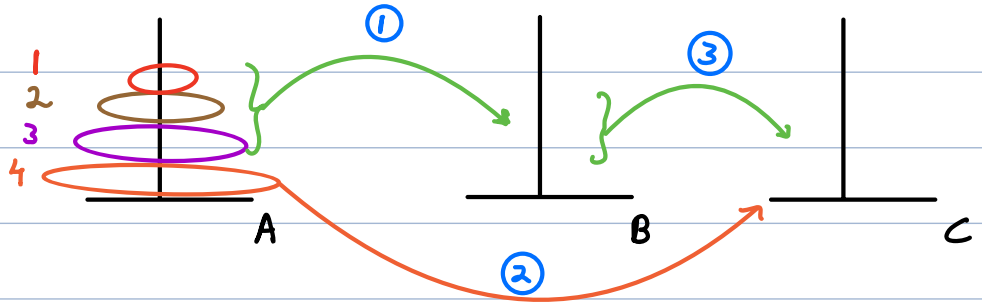


Move (N-1) disks from A → B

Move N^{th} disk from A → C

Move (N-1) disks from B → C

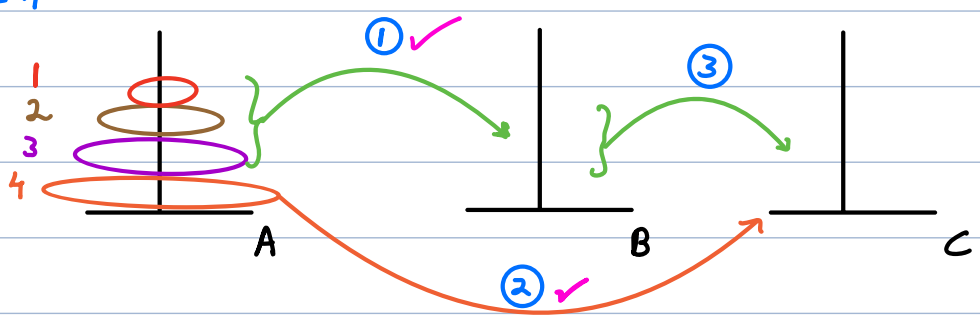
N = 4



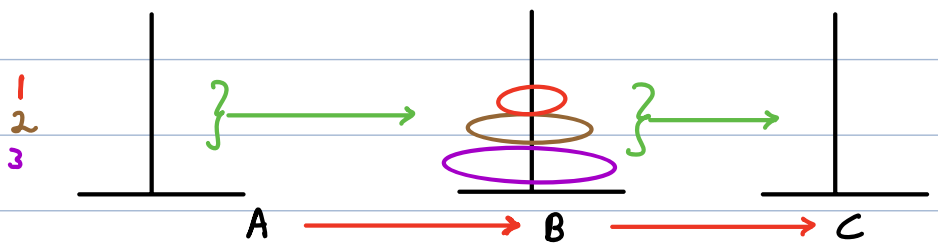

```
void toh (N, T1, T2, T3) {        T1 → T3
    if (N == 1) {
        print (N, T1 → T3)
        return
    }
    toh (N-1, T1, T3, T2)   // A C B  (A → B)
    print (N, T1 → T3)      // N, A → C
    toh (N-1, T2, T1, T3)   // B A C  (B → C)
}
```

(A B C)

N = 4



N = 3



steps = 7 # steps = 7

N → # steps = $2^N - 1$

TC = $O(2^N)$

SC = $O(N)$

N	# steps		
1	1		$2^1 - 1$
2	3	(1 + 1 + 1)	$2^2 - 1$
3	7	(3 + 1 + 3)	$2^3 - 1$
4	15	(7 + 1 + 7)	$2^4 - 1$
⋮	⋮		