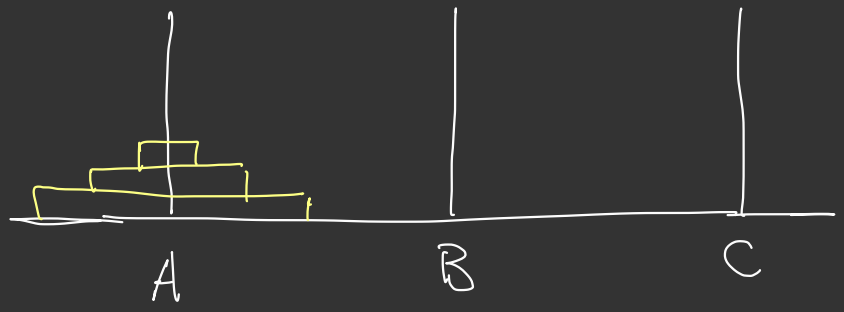
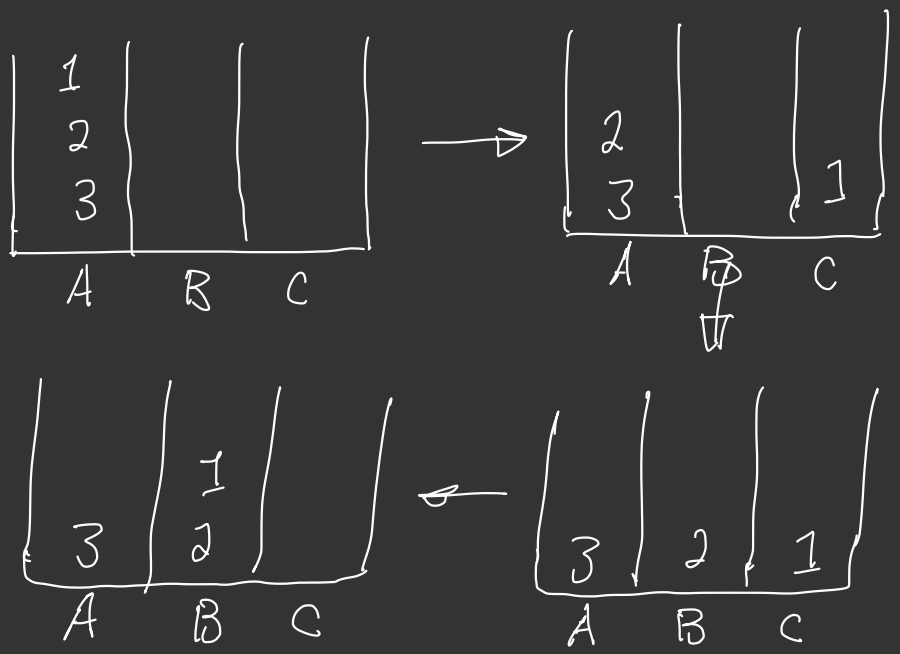
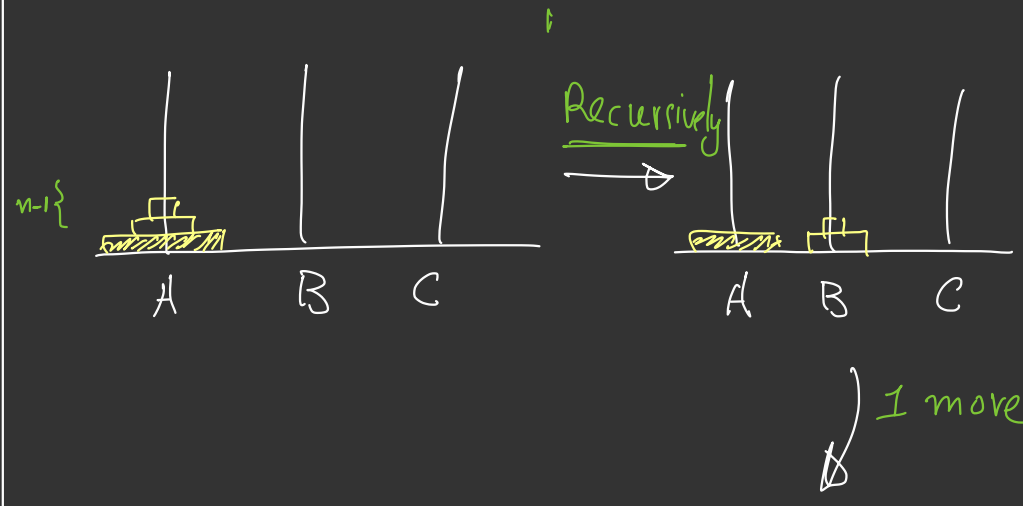
Tower of Hanoi

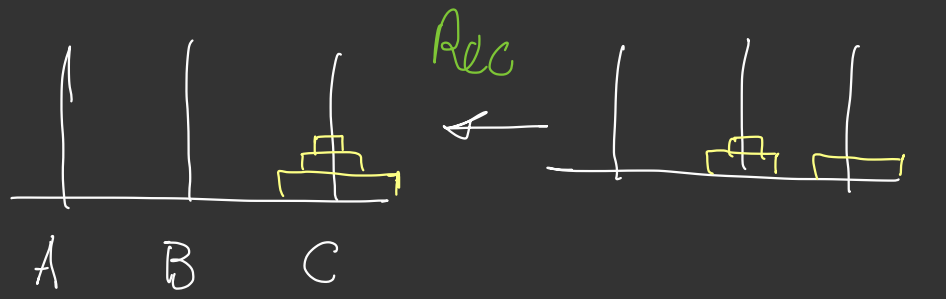


Rules: (1) move one disk at a time.

(2) never place a disk on a smaller disk.







Recursively: possibly multiple moves.

```
Hanoi (n, src, dst, tmp)
    if n > 0:
        Hanoi (n-1, src, tmp, dst)
        move n to dst
        Hanoi (n-1, tmp, dst, src)
```

Hanoi (n, src, dst, tmp)

if n > 0:

- Hanoi (n-1, src, tmp, dst)
- move n to dst
- Hanoi (n-1, tmp, dst, src)

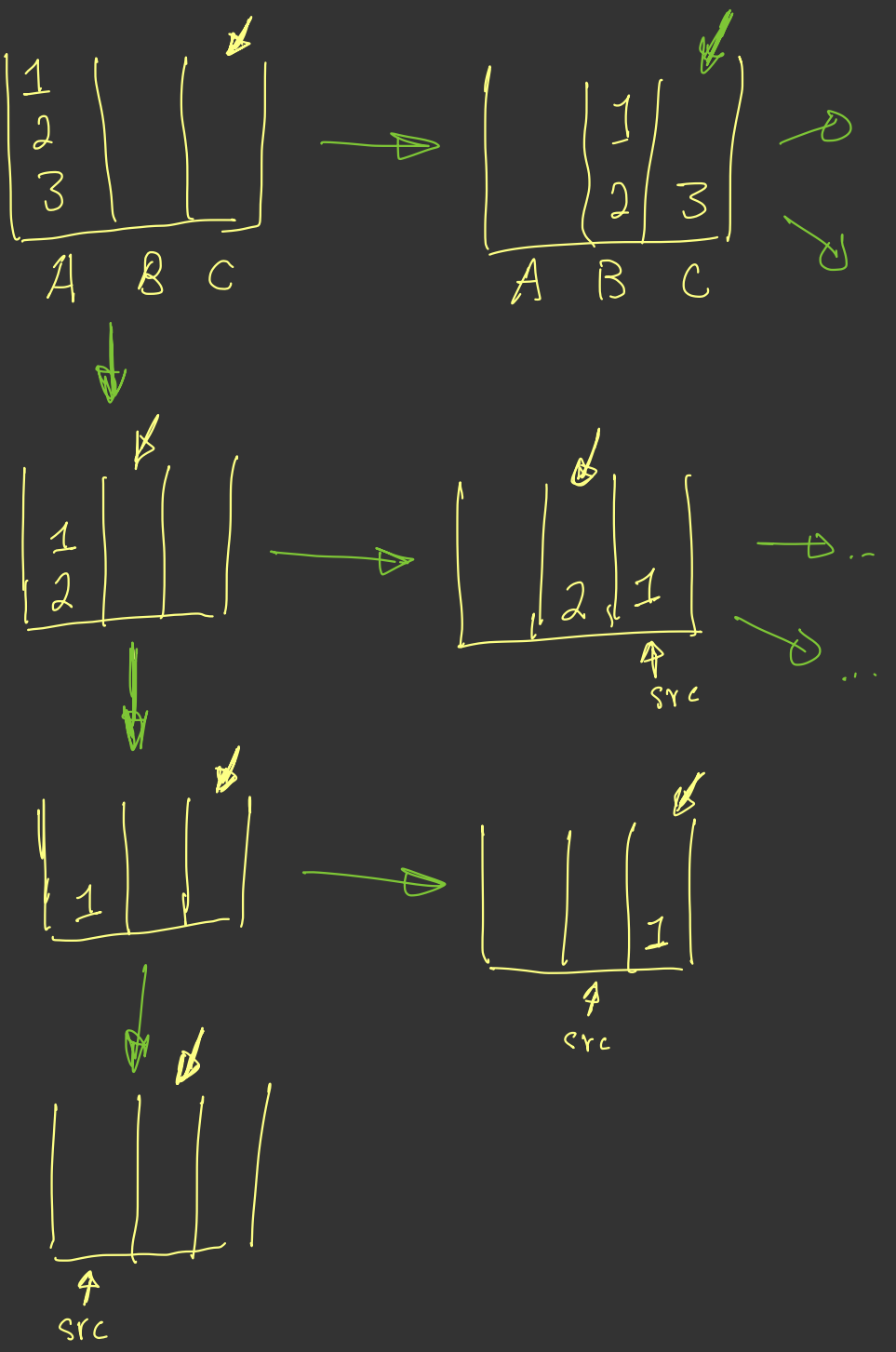
How many moves does the Hanoi alg do to solve The puzzle?

T(n): # of moves for n disks.

$T(0) = 0$

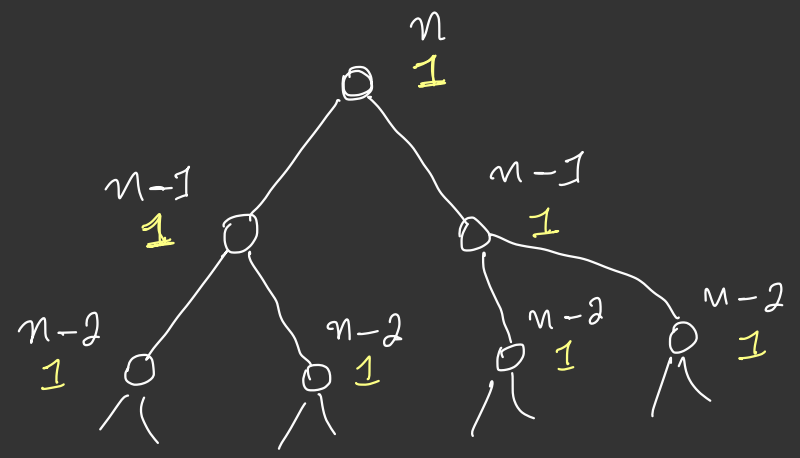
$T(n) = T(n-1) + 1 + T(n-1)$

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



$T(0) = 0$

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



1

2

4

2^{i-1}

2^{n-1}

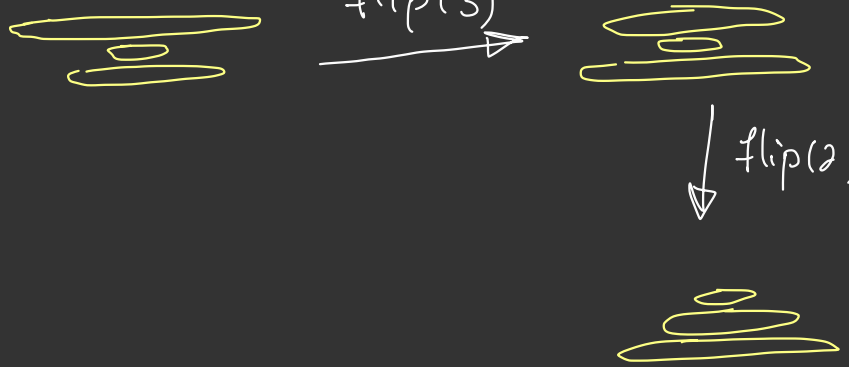
$$T(n) = 1 + 2 + \cdots + 2^{n-1} = 2^n - 1$$

Pancake Sorting

Given a stack of pan-cakes.
(#) they have all diff sizes.

flip operation:

flip(k): flip the order of the top pancakes (using a spatula).

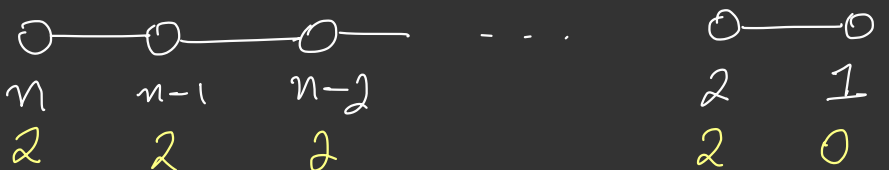


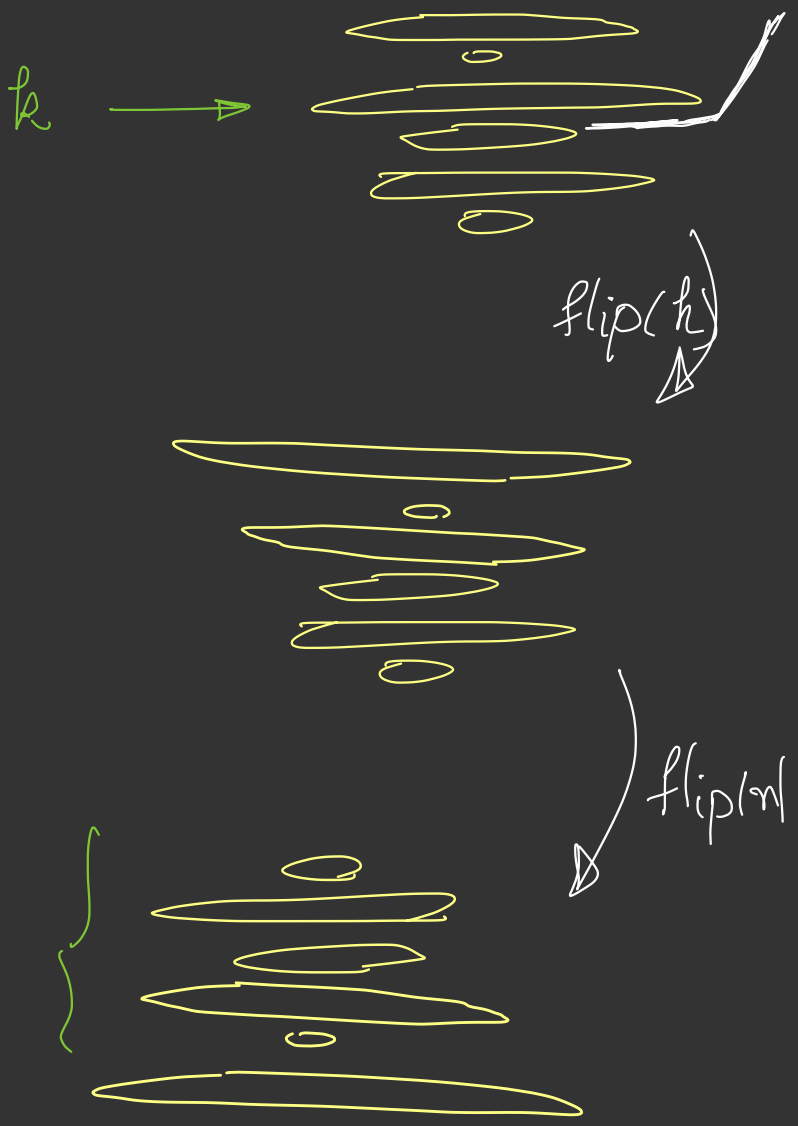
```
Pancake(n):
  if n > 1:
    k ← index_of_the_largest_pancake
    flip(k)  # after this max is on top
    flip(n)  # after this max is at bottom
    Pancake(n-1)
```

How many moves 'Pancake' alg uses to sort pancakes?

$T(n)$: # of moves for a stack of size n.

$$\left.\begin{aligned} T(n) &= 2 + T(n-1) \\ T(1) &= 0 \end{aligned}\right\} \Rightarrow T(n) = 2(n-1)$$





Sorting

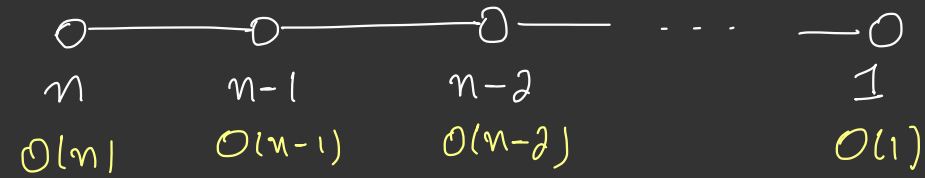
```
InsSort(A[1..n]):
O(1) → if n>1:
T(n-1) → InsSort(A[1..n-1])
O(n) → Insert A[n] to A[1..n-1]
```

Ex: write alg for inserting A[n] into A[1..n-1] that does it in O(n) time.

T(n): RT of InsSort for n numbers.

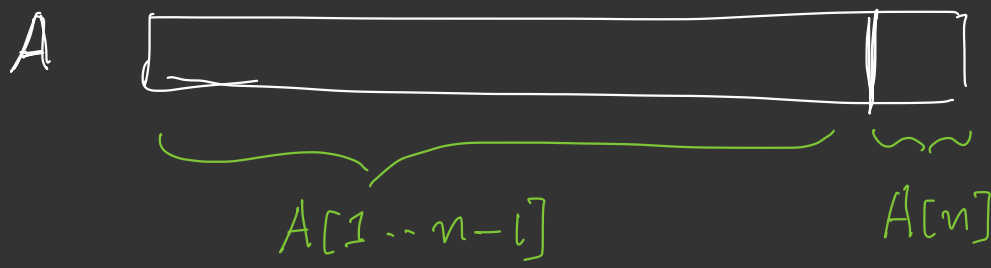
$T(1) = O(1)$

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

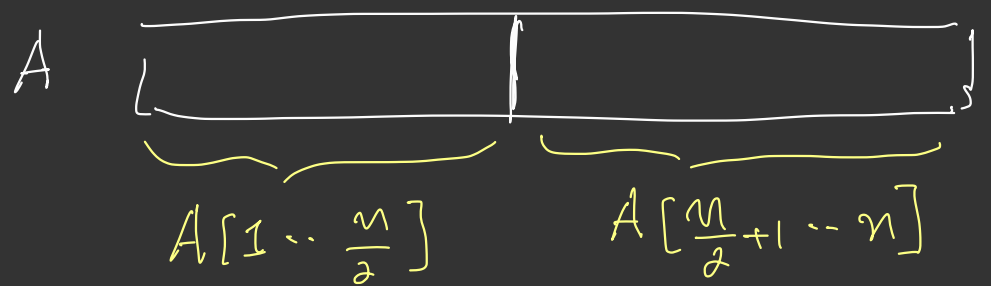


$$T(n) = O(n) + \cdots + O(1)$$
$$= O(n + \cdots + 1)$$
$$= O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

Insertion Sort



Merge Sort



MergeSort (A[1..n]):

O(1) → if $n > 1$:

O(1) → $m = \lfloor \frac{n}{2} \rfloor$

$T(\frac{n}{2})$ → MergeSort (A[1..m])

$T(\frac{n}{2})$ → MergeSort (A[m+1..n])

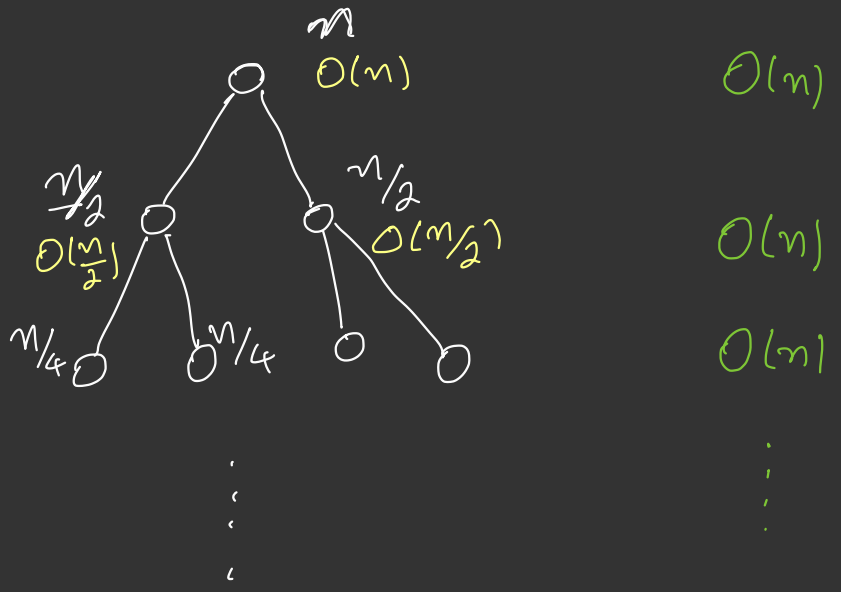
O(n) → Merge (A[1..m], A[m+1..n])

Assuming A[1..m] and A[m+1..n] are sorted merges them into one sorted array.

Ex: implement Merge in O(n) time.

$T(1) = O(1)$

$T(n) = 2T(\frac{n}{2}) + O(n)$



Binary tree has $\lg n$ levels

$T(n) = \lg n \cdot O(n) = O(n \lg n)$

```
Pancake(n):
  if n ≥ 1:
    k ← index_of_the_largest_pancake
    flip(k)  # after this max is on top
    flip(n)  # after this max is at bottom
    Pancake(n-1)
```