Recap: Arrays → 1st op: Sorting → 1) Bubble sort → $O(n^2)$ complexity

Selection / Insertion → Safely Ignore!

2) Faster. Recursive sorting. Pre: Recursion

Remember: Recursion can be understood in two ways → a) Visualize it completely | Entire call stack

b) Think inductively.

Repeat: What is inductive thought?

a) Ensure your base cond^n is correct. | Easy

b) Assume that decomposition step brings you correct results. (You can check this for smaller inputs) → manually

Note: You decomposition call returns you same type what parent call returns.

Recursion:
Base cond^n
Problem Decomposition
Problem Recomposition

Eg.

```
int sum(arr, a, b):
    o/p here is net sum of array + a + b.
```

↓ sum(,.,)

Int

① Diff in datatype

② Diff in operation

```
bool sum(arr, a, b):
    o/p here is check if a+b is present in arr or not.
```

↓ sum(...)

True/False

① What does it do?

② What does it return?

③ If you got the correct results from decomposition, can you combine/recompose these partial results into the correct result that you expected.

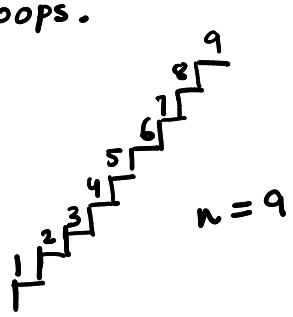
→ My recursion will work. Major thought → Base cond^n Recomposition.

ILLUSTRATIONS.

① Mergesort

② Quicksort. ↓ Template #2 ↑ Many questions are variations.

MYTHBREAKER: Thinking recursive is hard compared to loops.



Imp Q. There are n steps on a staircase.

Allowed_steps = 1 or 2. a, b, c

Number of ways a staircase could be climbed.

Eg: n = 4

(1, 1, 1, 1)
(2, 1, 1)
(1, 2, 1)
(1, 1, 2)
(2, 2)

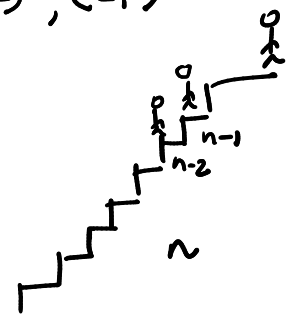
O/p: 5 (∵ 5 different ways)

ways ⇒ Sequence in which you take 1/2 steps.

ways(n) = ways(n-a) + ways(n-b) + ways(n-c) → Generalize!

Sol^n

n = 0 ans = 1
n = 1 ans = 1
n = 2 ans = 2 ∵ (1,1) (2)
n = 3 ans = 3 ∵ (1,1,1) (1,2), (2,1)
⋮ ⋮

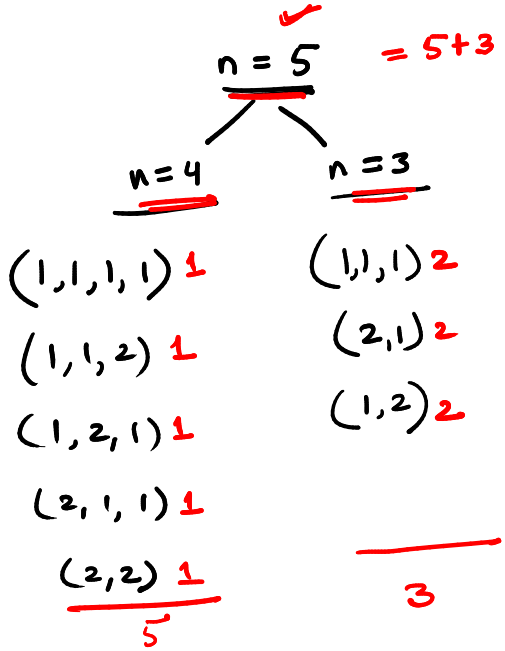


let w_n = no of ways you can reach n steps.

I will be at w_{n-1} or w_{n-2}.

$w_n = w_{n-1} + w_{n-2}$

↳ n^{th} Fibonacci number.




```
int ways(n):
    if n == 0 || n == 1 :
        return 1
    return ways(n-1) + ways(n-2)
```

← Terrible n=47 "Improve."

Debate: Thm: Every iterative code can be written using recursion.
Every recursion code can be written iteratively.

MERGESORT.

arr = [3 5 8 1 7 6 4 2] → sort this!

How can I democratize this?

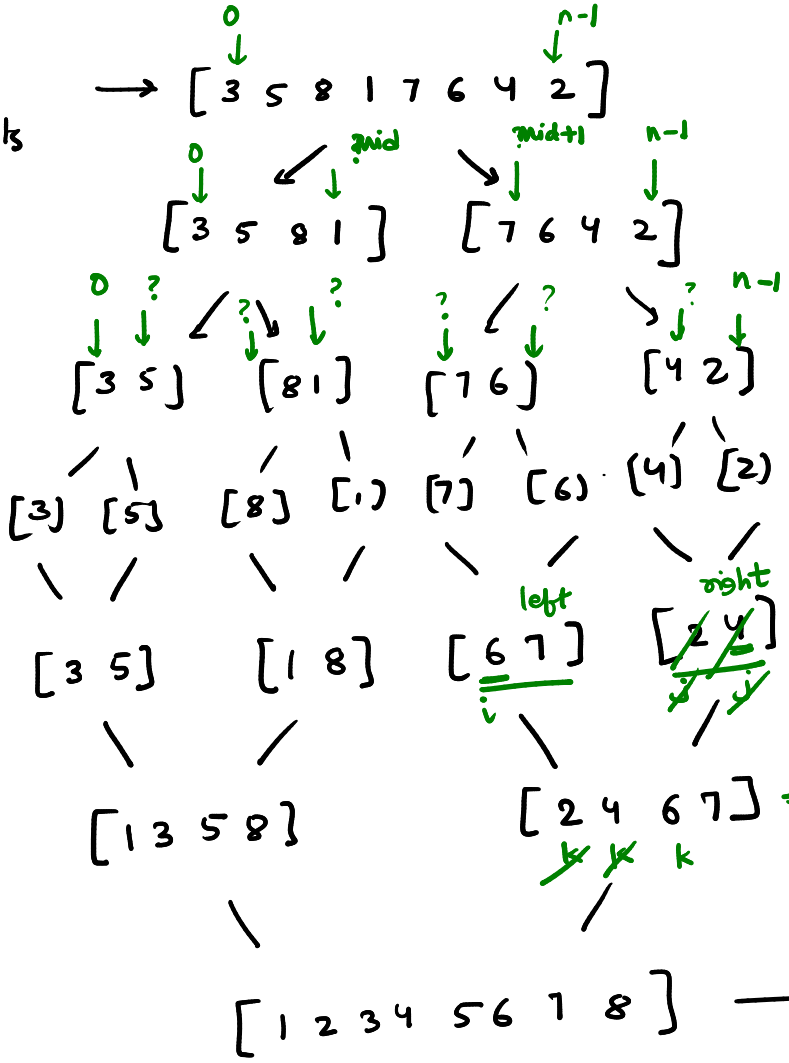
① If all elements are treated separately / individually / exclusively.

1 array
n elements → [3 5 8 1 7 6 4 2]

Idea: Single elements are always sorted.

diff of 1 size is allowed.

Visualization by keeping track of call trace.



→ n sorted arrays of 1 element.

j=0 j=1 j=2/n ∵n=2

[ ]

⇒ arr

sorted arr!!

```
merge_sort (arr, start, end):
    if start >= end :
        return
    mid = (start + end)/2
    left_arr = arr [start : mid]
    right_arr = arr [mid+1 : end]
    merge_sort (left_arr, start, mid)
    merge_sort (right_arr, mid+1, end)
    Merge (left_half, right_arr)
```

Implementation →

```
mergesort (arr, start, end)
    mid = s+e/2
    ms (arr, start, mid)
    ms (arr, mid+1, end)
```

Indices →

Intuitive. ★★

Arrays →

```
mergesort (arr):          ← len == 1 return
    mid = len(arr)/2
    left = arr [s : mid]
    right = arr [mid+1 : e]
    ms(left) ms(right)
```

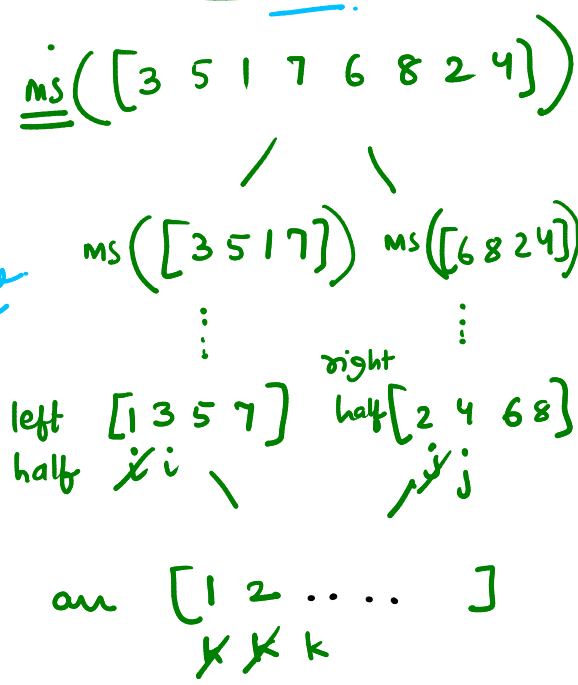
```
merge (arr, left_half, right_half):
    i = j = k = 0
    m = len(left_half)
    n = len(right_half)

    while (i < m and j < n):
        if left[i] < right[j]:
            arr[k] = left[i]
            i++  k++
        else:
            arr[k] = right[j]
            j++  k++

    while (i < m):
        arr[k] = left[i]
        i++  k++

    while (j < n):
        arr[k] = right[j]
        j++  k++
```

There are elements in both arrays

There are some elements in left half which are pending

pending elements are in right half.

Inductive Thought:



Example

Start = 0
end = 8
mid = $\frac{0+8}{2} = 4$

9 elements.

0,4 | 5,8

right has 1 ele less in unequal splits.

0,1,2,3,4 | 5,6,7,8 | 4

5

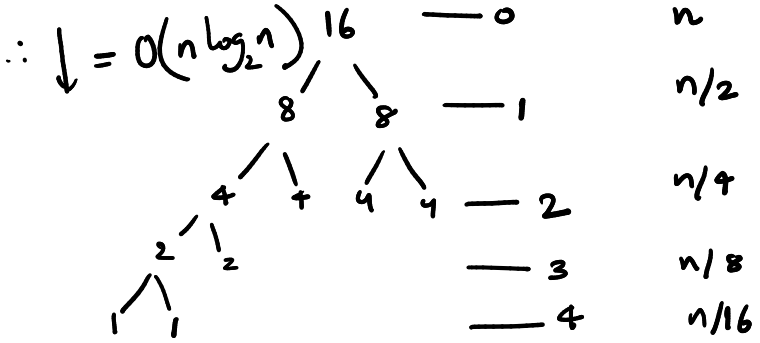
Did we improve on time? / T.C.

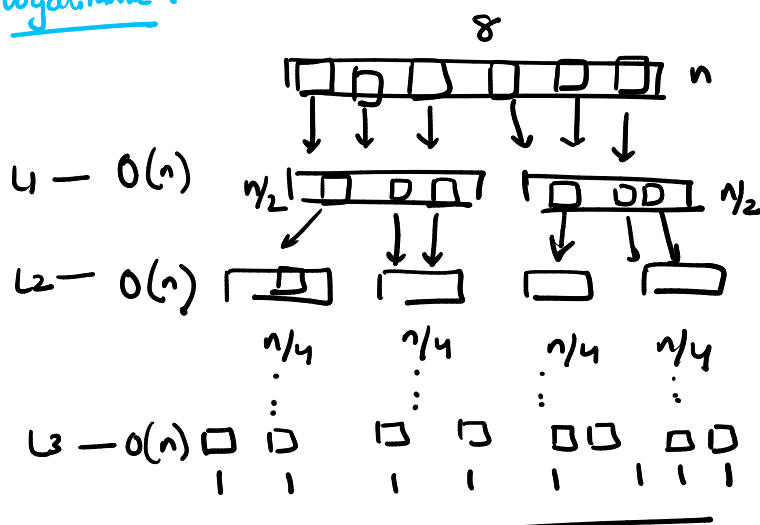
logarithmic.

Work in decomposition = $O(n) * \text{levels}$

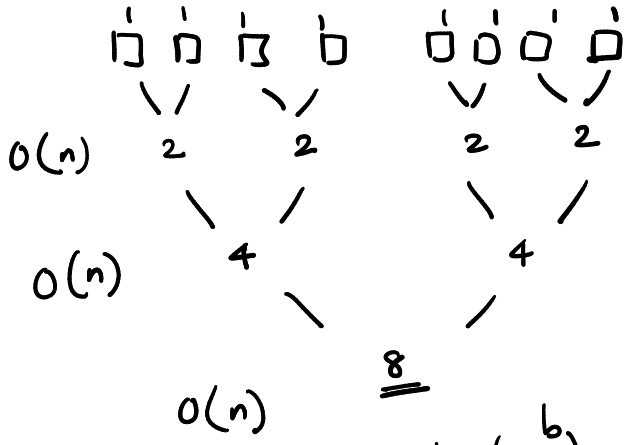
levels = $\log_2 n$

$\therefore T = O(n \log_2 n)$





Recomposition $= O(n \log_2 n)$



Total work $= O(2n \log_2 n)$

$\uparrow$

$\times$

$\log(a^b) = b \log a$

* **TC $\sim O(n \log_2 n)$**

$n \log_2 n < 10^8$

$n \sim 10^5$

$n \sim 10^6$

$n \log_2 n$

$\uparrow$ ~ 30

$10^6 \times 6 \times \log_2 10$ $\sim 1.8 \times 10^8$

$10^5 \times 5 \times \log_2 10$ $\sim 2 \times 10^6$

You are much much better than 10^4 of bubble sort.

NOTE: Nope! $n \log n$ is best general sorting time complexity.

Certain property:

1) All values I get are bet? 0 to 1. $\rightarrow$ $\because$ you are in IP
2) All values are bet? 1 to 30 $\Rightarrow$ $\because$ 30 is max cap on no. of students.

↳ $O(n)$ is possible but only if a certain property is given and can help in ordering.

10^8 size array $\longrightarrow$ Nope! Can't sort in 1s in general setting !!

"100 computers" $\rightarrow$ 10^6 array $\longrightarrow$ sort. — 1s

Combine 100 arrays. — 1s

$\downarrow$

10^8 in 1s.

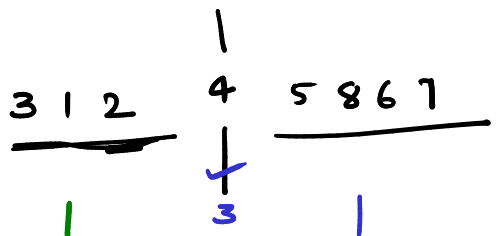
QUICKSORT

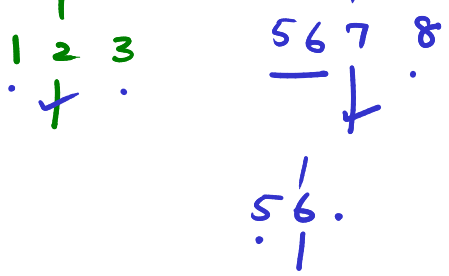
Intuition.

1. Pick the last element.
2. Partition the array across this element. This element finds its correct position.
3. Do this recursively on left half and right half.

[3 1 5 8 6 7 2 4]

How this partition? →





[1 2 3 4 5 6 7 8]

Probabilistic idea.

T.C. is very complex.

$E(TC) \sim O(n \log_2 n)$

Think inductively : qs(arr) ⇒ sort this array.

```
i = partition(arr)
qs(arr[start : i])
qs(arr[i+1 : end])
```

n elements : 0 to n-1 — you never know the last element