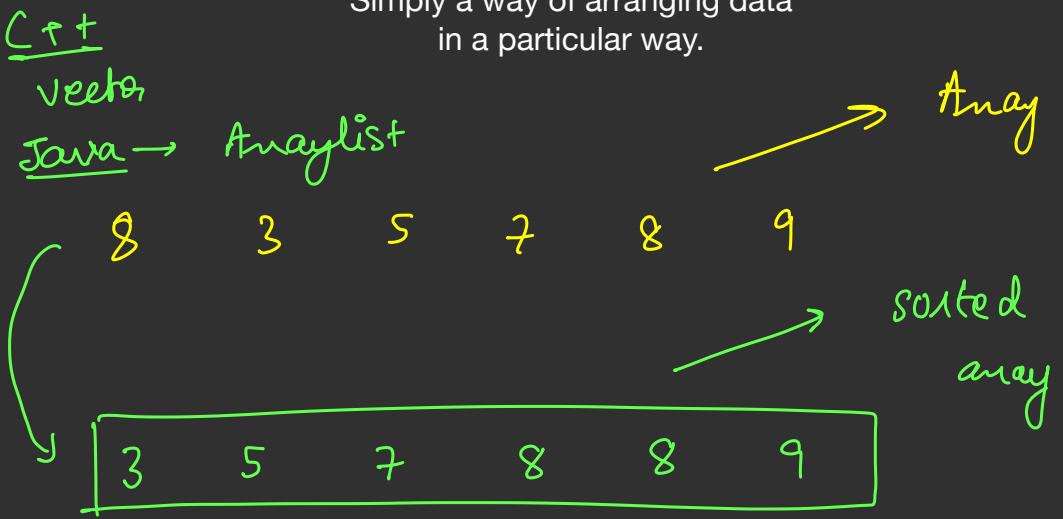
L18
Sorting : Introduction

What is Sorting?

Simply a way of arranging data in a particular way.

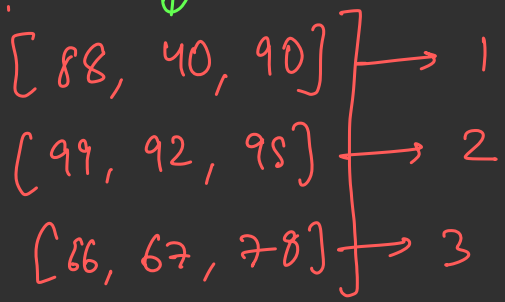


Example

["abc", "aab", "azx", "ef", "cd"]

aab, abc, azx, cd, ef

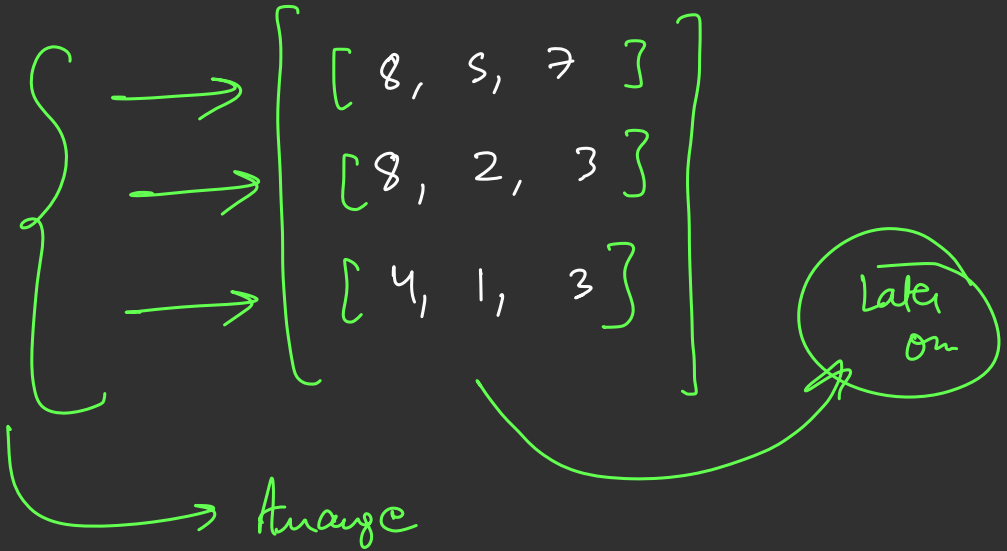
↳ inc / asc



→ Average Chemistry

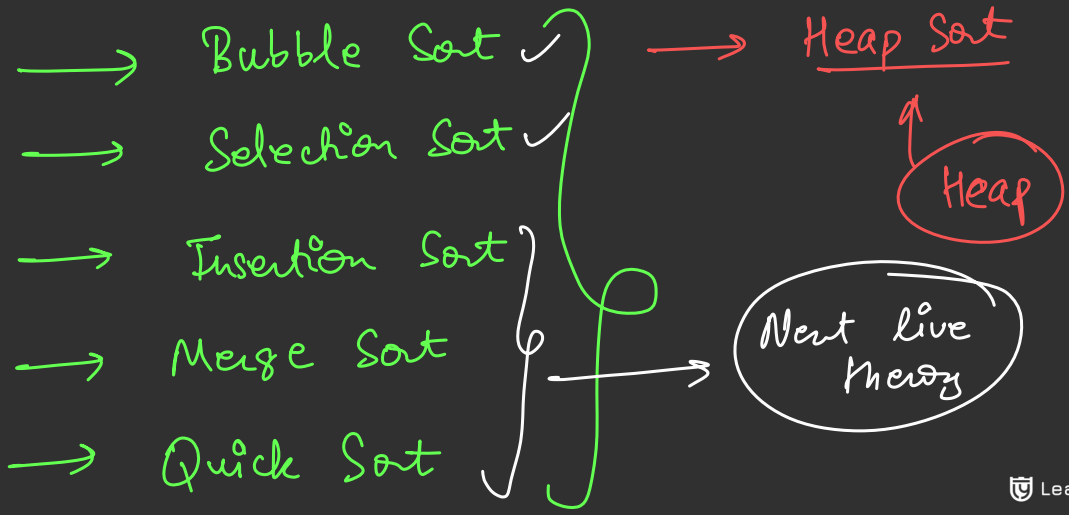
List of cuboids

l, b, h

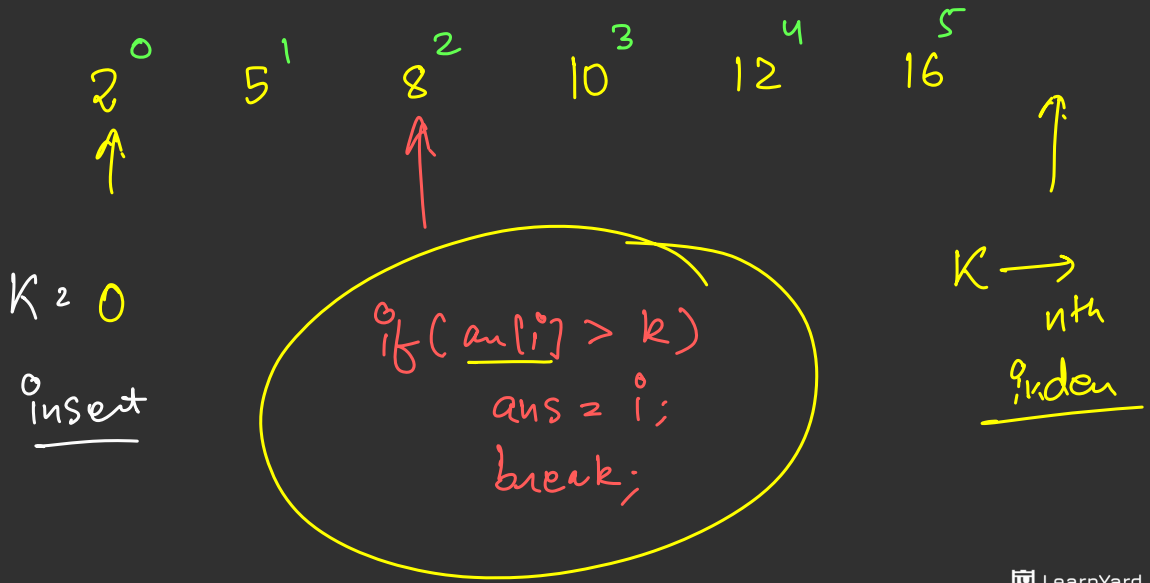


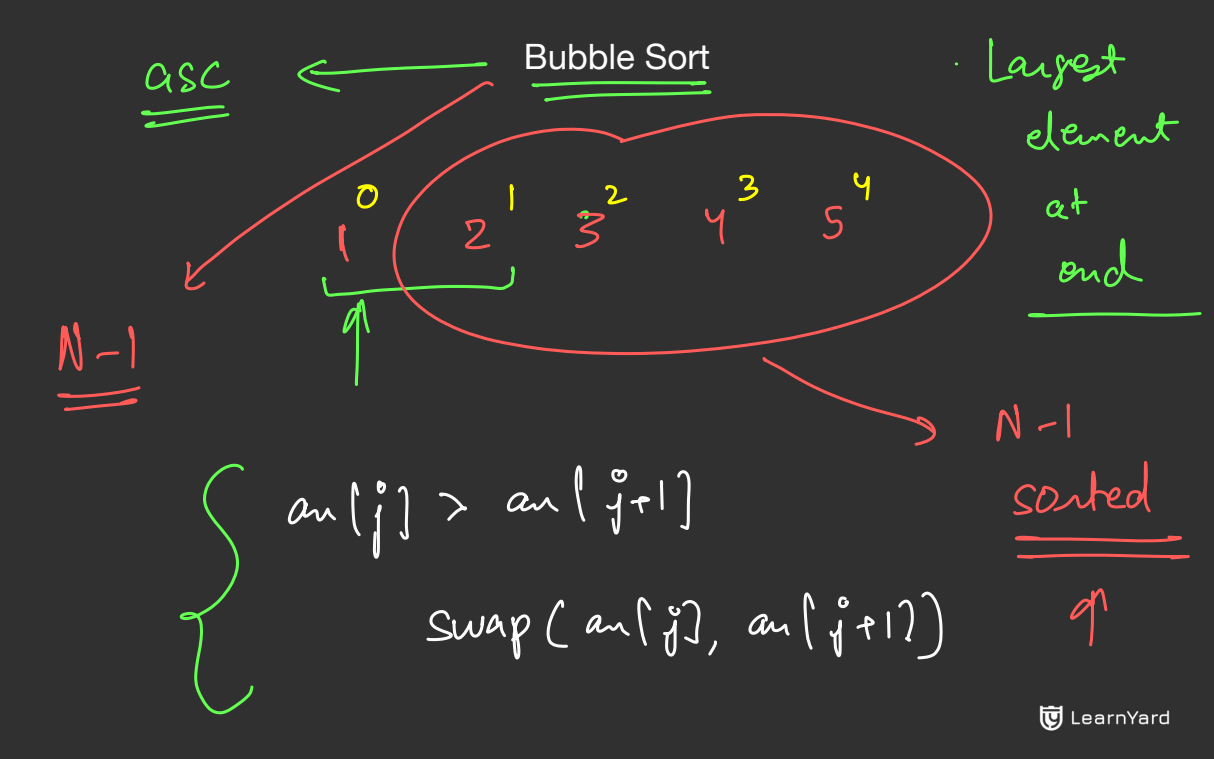
Sorting Algorithms

A lot of them are there in the world.
Few of them are the famous ones.



Inserting an element in an already sorted list




Bubble Sort
asc
Largest element at end
0 1 2 3 4
1 2 3 4 5
N-1
N-1
sorted
arr[j] > arr[j+1]
swap(arr[j], arr[j+1])
LearnYard

I → N-1 logic

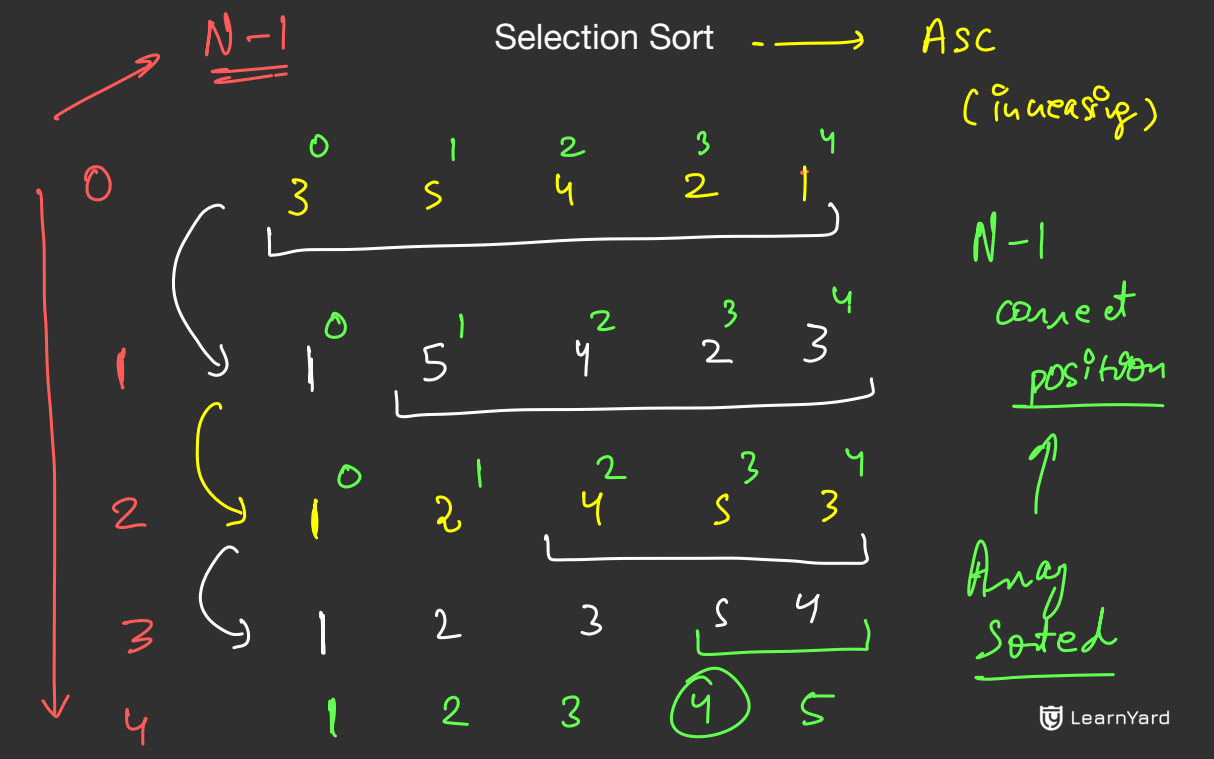
↓

II → { if(arr[j] > arr[j+1])
swap(arr[j], arr[j+1])

Property → N-1 element are in correct or they sorted

Whole Array is Sorted.

3 2 1 4 5


N-1
Selection Sort
Asc
(increasing)
0 1 2 3 4
3 5 4 2 1
0
1 0 5 1 4 2 2 3 3 4
1
1 0 2 1 4 2 5 3 3 4
2
1 2 3 5 4
3
1 2 3 4 5
4
N-1
correct
position
Array
Sorted
LearnYard

1. What is the best auxiliary space complexity a sorting algorithm can have?

A. O (n log n)
B. O (n)
C. O (1)
D. O (n^2)

1. What is the best auxiliary space complexity a sorting algorithm can have?

A. O (n log n)
B. O (n)
C. O (1)
D. O (n^2)

Solution : the discussed selection sort is an **in-place** sorting algorithm, and one can't go better than O(1).

2. Sort [🍎, 🍊, 🍊, 🍎] in ascending order.

A. [🍎, 🍊, 🍊, 🍎]
B. [🍊, 🍊, 🍎, 🍎]
C. [🍊, 🍎, 🍊, 🍎]
D. Can't sort.

apple & orange cannot be compared

2. Sort [🍎, 🍊, 🍊, 🍎] in ascending order.

A. [🍎, 🍊, 🍊, 🍎]
B. [🍊, 🍊, 🍎, 🍎]
C. [🍊, 🍎, 🍊, 🍎]
D. Can't sort. ✓

Solution : one can't compare apple and oranges. More formally, the comparator function is not defined.

3. Sort [🍎, 🍊, 🍊, 🍎] in ascending order, if 🍊 < 🍎

A. [🍎, 🍊, 🍊, 🍎]
B. [🍊, 🍊, 🍎, 🍎]
C. [🍎, 🍎, 🍊, 🍊]
D. [🍊, 🍎, 🍊, 🍎]

Doubt session

↳ Custom Comparator

3. Sort [🍎, 🍊, 🍊, 🍎] in ascending order, if 🍊 < 🍎

A. [🍎, 🍊, 🍊, 🍎]

B. [🍊, 🍊, 🍎, 🍎] ✓

C. [🍎, 🍎, 🍊, 🍊]

D. [🍊, 🍎, 🍊, 🍎]

Solution : The comparator function is now defined.

4. Which of the following best describes a stable sorting algorithm (single answer)?

A. Any algorithm that doesn't crash for all possible inputs.
B. An algorithm that correctly sorts the input array correctly for all possible inputs.
C. An algorithm that sorts the input and maintains relative order of those elements that are equal to each other at the end for all possible inputs.
D. An algorithm that never allows two elements to be equal to each other.

Stable Algo will discuss in doubt session

4. Which of the following best describes a stable sorting algorithm (single answer)?

A. Any algorithm that doesn't crash for all possible inputs.
B. An algorithm that correctly sorts the input array correctly for all possible inputs.
C. An algorithm that sorts the input and maintains relative order of those elements that are equal to each other at the end for all possible inputs.
D. An algorithm that never allows two elements to be equal to each other.

Solution : Definition

5. Is bubble sort a stable sorting algorithm?

A. Yes
B. No

5. Is bubble sort a stable sorting algorithm?

A. Yes
B. No

Solution : The order is not changed when swapping the values since only adjacent elements are swapped when the first element is strictly greater than the second.

6. Is the following sorting algorithm stable?

A. Yes
B. No
C. It is not a correct sorting algorithm

```
for i do in range 0...n-1
    for j do in range 1...n-1-i
        if then A_j ≤ A_{j-1}
            swap(A_{j-1}, A_j)
        end if
    end for
end for
```

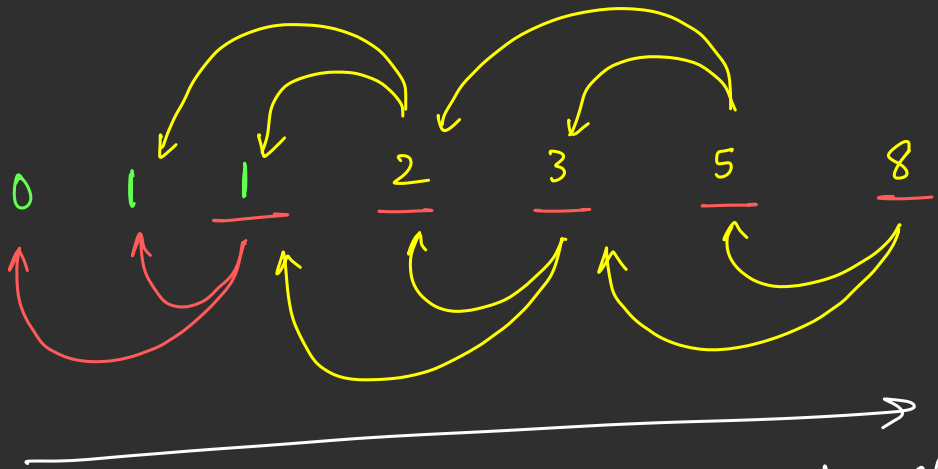
6. Is the following sorting algorithm stable?

```
for i do in range 0...n − 1
    for j do in range 1...n − 1 − i
        if then A_j ≤ A_{j−1}
            swap(A_{j−1}, A_j)
        end if
    end for
end for
```

A. Yes
B. No ✓
C. It is not a correct sorting algorithm

Solution : Consider (1, 1, 3), after the first iteration it becomes (1, 3, 1) first element moves to the end and relative position of the 1's change.

Fibo



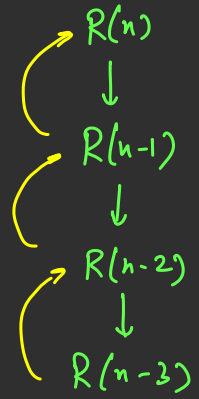
Nth Fibo Number

N2 Sth

Fibo

$N^{th} \rightarrow (N-1)^{th} + (N-2)^{th}$

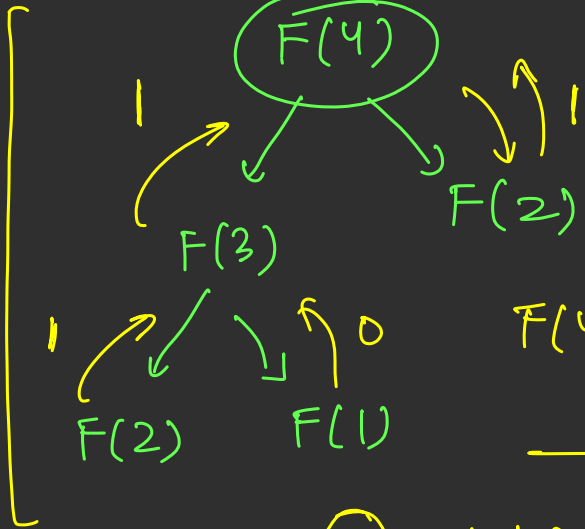
Bigger → Smaller } → Recursion



$F(3) \rightarrow F(1) + F(2) \rightarrow 1 + 1 = 2$

3rd

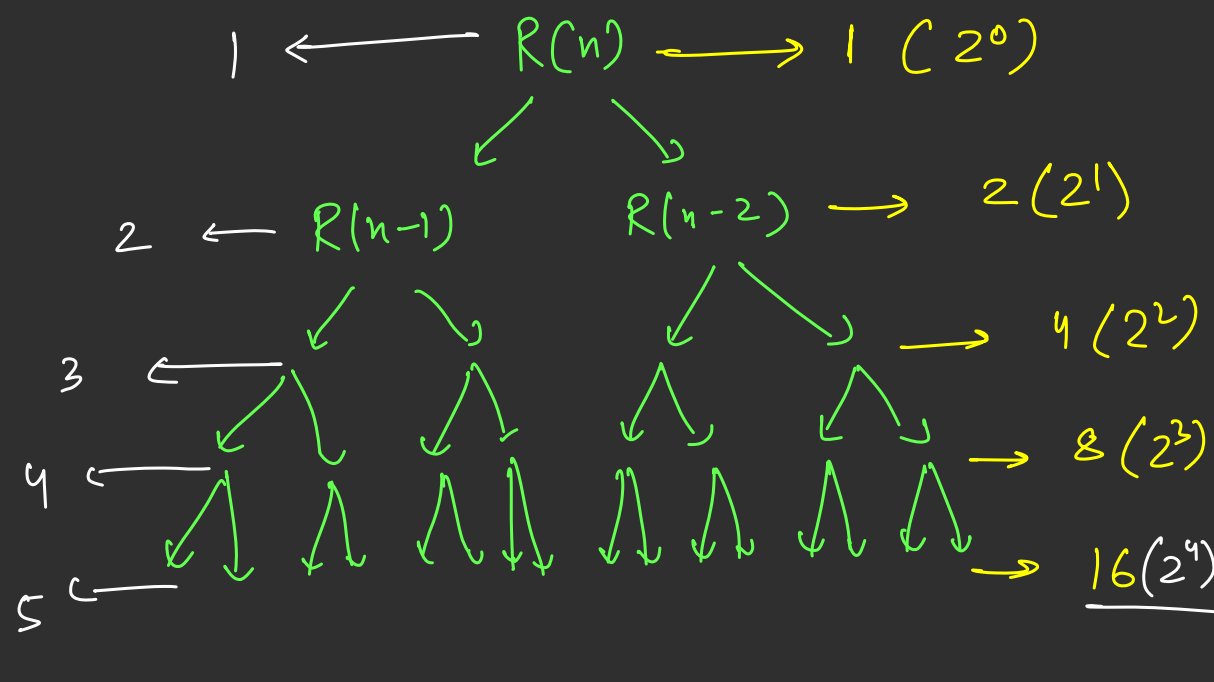
F(1) → 0



$F(4) = 1 + 1$

$= 2$

→ 2 Multiply (Exponential)


1
R(n)
1 (2^0)
2
R(n-1)
R(n-2)
2 (2^1)
3
4 (2^2)
4
8 (2^3)
5
16 (2^4)

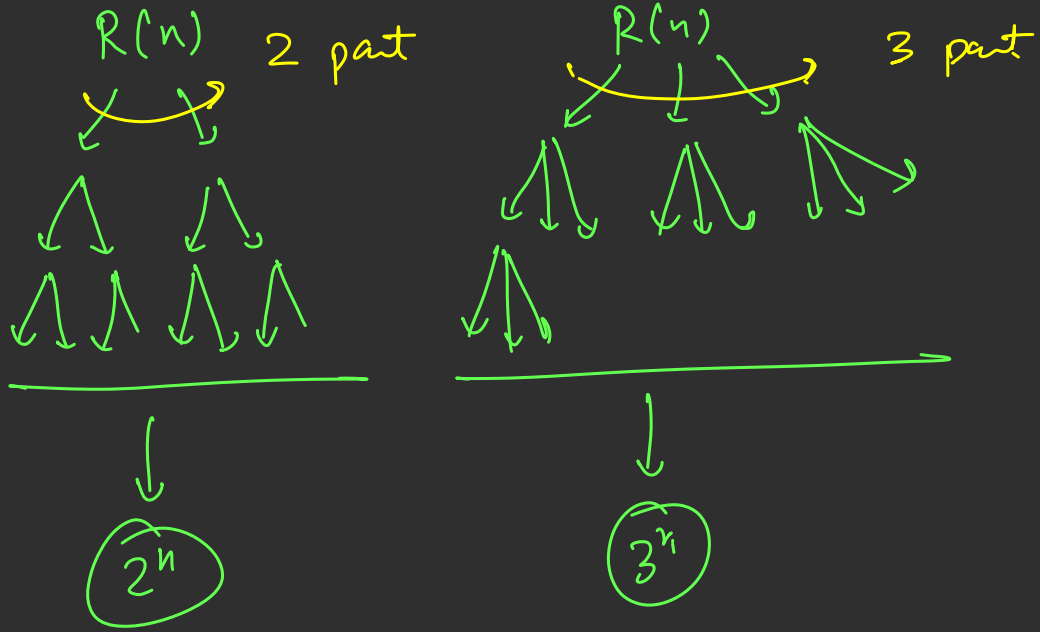
N^{th} level $\rightarrow$ 2^{n-1}

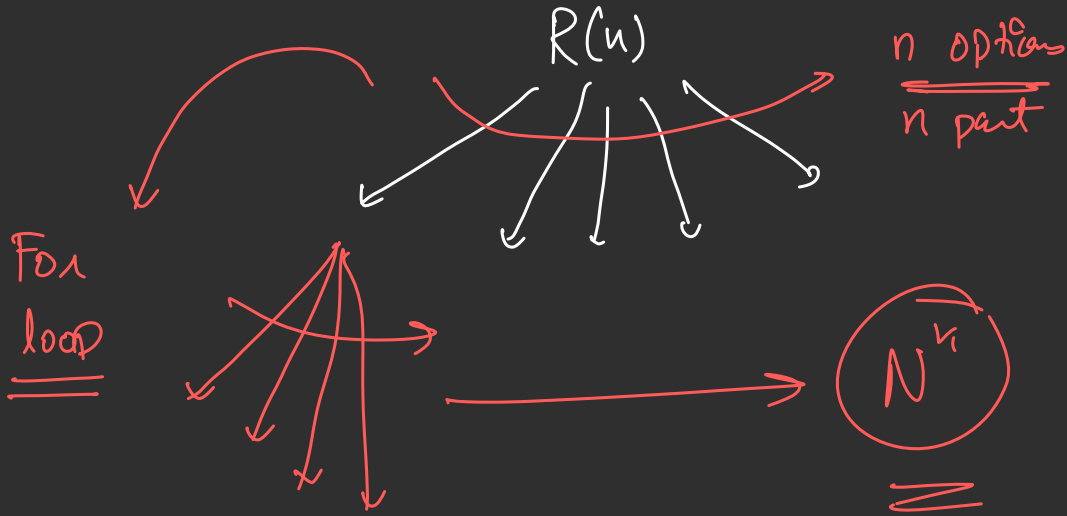
TC $\rightarrow$ 2^n

Sum $\rightarrow$ $2^0 + 2^1 + 2^2 - - - 2^{n-1}$

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

$O(2^n)$


R(n)
2 part
R(n)
3 part
2^n
3^n


R(n)
n options
n part
For loop
N^n