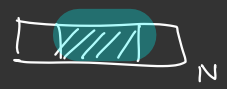
Advanced Batch

卐 Sri Ganesh

→ **ARRAYS**

↳ Subarrays

↳ Subsequence / Subset



Choose 2 indices out of N indices to get a subarray

$^{N}C_{2} \Rightarrow O(N^2)$ Subarrays

#1 Subarray

1, 2, 3, 4

Len = 1	Len = 2	Len = 3	Len = 4
1	1, 2	1, 2, 3	1, 2, 3, 4
2	2, 3	2, 3, 4	
3	3, 4		
4			

$$4 + 3 + 2 + 1$$

$$= 1 + 2 + 3 + \dots N = \frac{N(N+1)}{2} = O(N^2) \text{ subarrays}$$

2

Subsequence | Subset

Smaller set of elements from the array, may or may not be continuous

↳ Ordering doesn't matter

$N = 4$

1	2	3	4

Len = 0	Len = 1	Len = 2	Len = 3	Len = 4
{ }	{1}	{1, 2}	{1, 2, 3}	{1, 2, 3, 4}
1	{2}	{1, 3}	{1, 2, 4}	1
	{3}	{1, 4}	{2, 3, 4}	
	{4}	{2, 3}	{1, 3, 4}	
	4	{2, 4}	4	
		{3, 4}		
		6		

2^N subsets

$= O(2^N)$

Total subsets $= 1 + 4 + 6 + 4 + 1$

$= 16$

$2 \cdot 2 \cdot 2 \cdots \cdots 2 = 2^N$ combinations

1	2	3	.	- -	N

include

exclude

N N N

1	2	3

Y Y Y

{1,2,3}

Y N Y

1	2	3

{ } {1,3}

Y N N

1	2	3

{1}

N = 3

1	2	3

{ } {1} {1,2} {1,2,3}

{2} {2,3}

{3} {1,3}

= 8 subsets

$= 2^3$ 2^N

$= \cancel{3^2}$ $\cancel{N^3}$

Subsequence → String Problems

Subset → Number

Hello

1234

H l o → subset/subseq.

1 3 4 →

Problems

Q1 → Find all subsets → Bitmanipulation (Today) / Backtracking (Later)

Q2 → Find a subset that satisfies a given condition.

↳ Sum of elements is X (one more step)

$N = 3$

Array: | 1 | 2 | 3 |

i = 0	0	0	0	{ }	0
i = 1	0	0	1	{3}	3
i = 2	0	1	0	{2}	2
i = 3	0	1	1	{2,3}	5
i = 4	1	0	0	{1}	1
i = 5	1	0	1	{1,3}	4
i = 6	1	1	0	{1,2}	
i = 7	1	1	1	{1,2,3}	

N

gives all possible bit combination of len = 3

Bitwise ops

Subset $2^3 = 8$ subsets

all 8 subsets

$\boxed{X = 4}$

Apply filter to get the subset

$N = 4$ $\quad 0 \text{ ———— } (2^4 - 1)$

$= 15$

$i = 0$ $\quad 0000$

0001

0010

$\vdots$

1111

$i = 15$ $\quad 1111$

$N = 20$

Subsets $\rightarrow 2^{20}$

$= 2^{10} \cdot 2^{10}$

$\sim (1000)^2$

$\sim 10^6$

$\boxed{1M}$

$N \sim 40$

$= (1000) \times (1000) \times (1000) \times (1000)$

$= 10^{12}$

Practically this is Not possible in given time limit

Execution Time

CPU

Max operations $= \boxed{10^8}$ ops / 1s

$2^N \rightarrow$ exponential complexity

$\downarrow$

value of $N \leq 25$

long $\rightarrow 10^{18}$

Time limit

$N \to 200$

hint

2^N ?

N^3

$(200)(200)(200)$

$= 8 \times 10^6$

$= 10^7$ yes

$< 10^8$

- Problems

Statement

Constraints

Max ops in 1s $\to 10^8$

- $N \le 10^6 \to O(N \log N) \to 10^6 \cdot \log 10^6 = 10^6 \cdot \log_2 (1000)^2 = 10^6 \cdot 2 \log_2 1000 = 2 \times 10^6 \cdot 10 = 2 \times 10^7$
- $N \le 100 \to O(N^4)$ $\quad (100)^4 = 10^8 \le 10^8$
- $N \le 20 \to 2^{20} = 10^6$ $\quad O(2^N)$
- $N \le 10^8 \to O(N)$

$O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(N^3) < O(N!) < O(2^N)$

$O(\sqrt{N})$

log → $\log_2 1000 \approx 10$

Sqrt → $\sqrt{1000} = 30$

$(30^2 = 900)$

$2^{10} \to 1024$

$\log_2 2^{10} = \log \log 4$

$\Rightarrow$ $\log 2024 = 10$

$$\boxed{\log N < \sqrt{N}}$$

$2^{\boxed{9.xx}} \to 1000$

$2^{10} \approx 1024$

$0 - 2^N$

Doubt

while (no > 0){

3

N
1 | 2 | 3

N

Max No of Bits in any No is $\log$ No

$\log_2 2^N$

$= N \log_2 2$

$= N$

Q1 Subarray with Zero Sum

Given an array of size N, return whether the array contains a subarray with sum == 0

Q2 Largest Continuous Sequence with Zero Sum

Find the largest continuous sequence in the array which sums to zero.

arr[] = 1, 4, 9, -1, 2, -10, 6, 5

zero sum

① Brute force

find out all subarrays

↳ find sum of each N

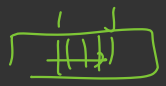
N^2

$O(N^3)$

for(i)

for(j)

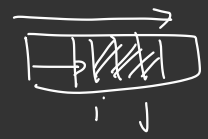
for(k = i — j)



② Prefix Sum

$O(N^2)$

```
for(i)
    for(j)
        sum →
```



$ps[j] - ps[i-1]$

$O(1)$

③ Prefix Sum + Hashing

zero Sum

i → j

1, 4, 9, -1, 2, -10, 6, 5

ps = 1, 5, 14, 13, 15, 5, 11, 16

$ps[j] - ps[i] = 0$ → $i+1$ ——— j = 0

⇒ $ps[j] = ps[i]$

Two indices which have the same value

hs

1

5

14 ✓

13 ✓

15 ✓

Hashset / HashMap

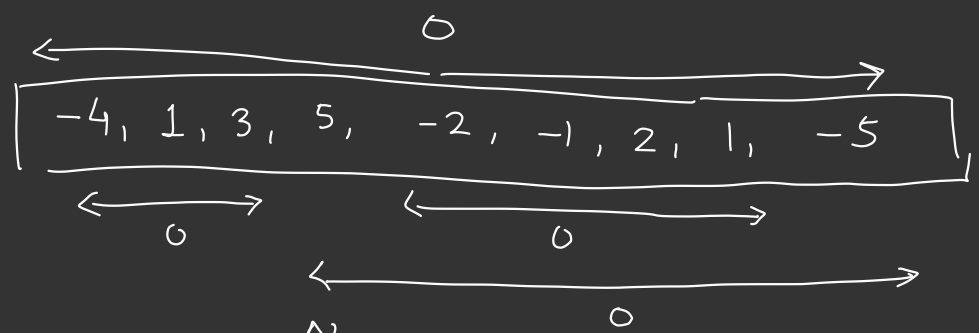
just Key is present / Not

Key - value /
5 - true / index

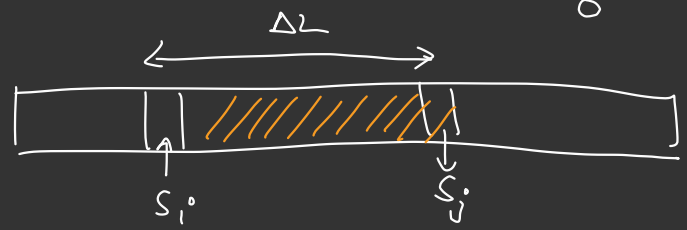
Largest Continuous Sequence with Zero Sum

Find the largest continuous sequence in the array which sums to zero.

Subarray



overlapping subarrays



$j - i$ → Max length

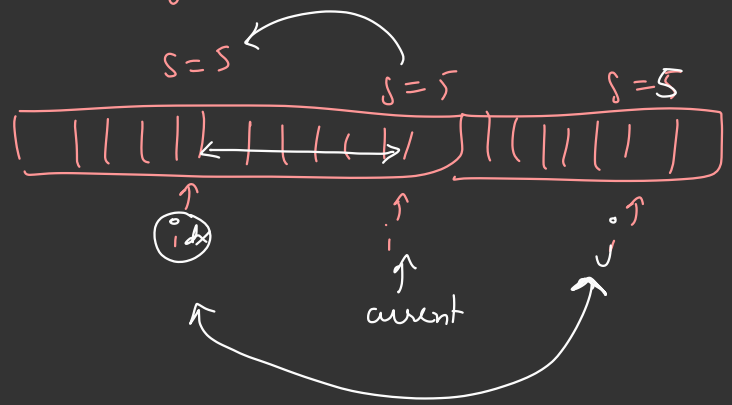
→ lowest possible idx

$$S_j - S_i = 0$$

with same sum

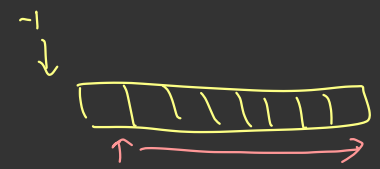
$S_j == S_i$

When ever we generate a new sum, we track idx in a hashmap.



5 → Lowest one

Hashmap <int, int> hm,



csum = 0

arr = 0, 1, 2, -3, -4, 4, 8

hm.put(csum, -1);

len = 0, left = -1, Right = -1;

for (i=0; i<= n-1; i++) {

csum = csum + arr[i];

if (!hm.contain(csum)) {

hm.put(sum, i)

Else{

if(i - hm.get(csum) > len){

len = i - hm.get(csum)

R = i, left = hm.get(csum) + 1,

}

csum = 0

0	1	2	3	4	5	6	7
6,	1,	2,	-3,	-4,	4,	8,	-14

0 → -1

6 → 0

7 → 1

9 → 2

2 → 4

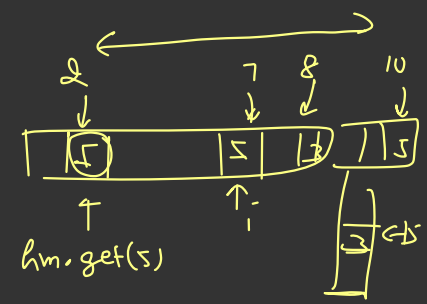
16 → 6

update L

→ 3-0 = 3

→ 5-0 = 5

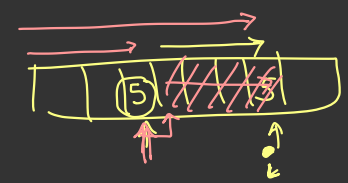
→ 7-(-1) = 8



7-2 = 5

10-2 = 8

15-8 = 7



$\Rightarrow$ print(len), Left, Right

iterate L to R.

$$\left[\begin{array}{l} ps(i) - ps(j) = k \\ \cancel{ps(i) - k = ps(j)} \end{array}\right]$$

$$ps(i) = ps(j) + k$$

N=6

	0	1	2	3	4	5
-1	1	2	3	4	5	-15
0	1	3	6	10	15	0

Csum

Special Case

$5 - (-1) = 6$

$1 - (-1)$

if(csum == 0)

len = i+1

Rainwater Trapping

Extra → Subarray Sort

11:05

Given an array, which was sorted, but a subarray was shuffled. (unsorted)

Find out the smallest subarray that we need to sort so that entire array becomes <u>SORTED</u>

(Array = [1, 2, 3, 4, 5, 7, 10, 16, 20])

<u>Input</u> = [1, 2, 3, 7, 10, 5, 4, 16, 20] // Input

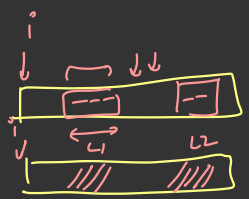
[1, 2, 3, 4, 5, 7, 10, 16, 20]

L ----- R

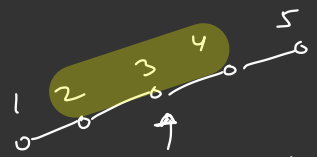
<u>Algorithm-1</u>

- Make a copy
- Sort the complete array → NLogN + N → fact → Quicksort Java / Tim Sort Python
- Compare using Two pointer approach.

= O(NLogN)



Algorithm - 2



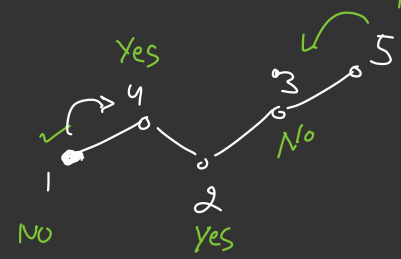
$a[i-1] < a[i] < a[i+1]$

will not satisfy if an element is out of order

Step-1

Largest = 4
Smallest = ~~1~~ 2] 000

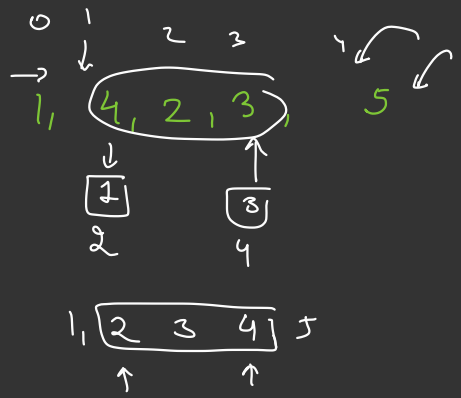
Step-2 find out correct loc of largest smallest



smallest which 000

largest which is 000

find out correct of 2 and 4



time → O(N)
space → O(1)

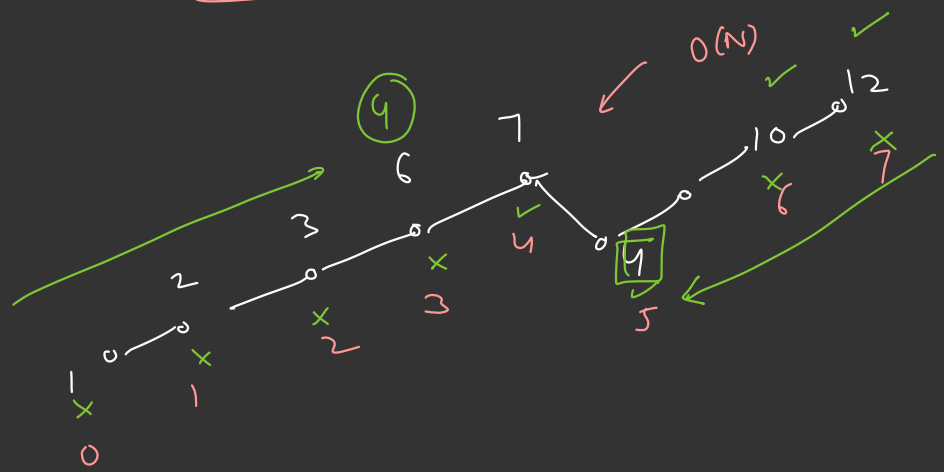
$a(i-1) < a(i) < a(i+1)$

4 7

1, 2, 3, ~~6~~, 7, 4, 10, 12

0 1 2 3 — —

⇒ 1, 2, 3 4, 6, 7, 10, 12



→ O(N)

Largest = 7

Smallest = 4

1, 2, 3, 4

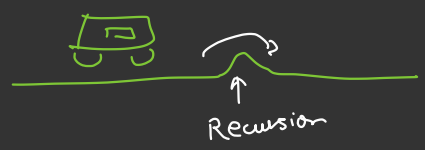
3

7, 10, 12

- Rec-1
- Rec-2
- ~~Rec-3~~

↳ Extra class - Hashing, String Builder, Comparators

Recursion



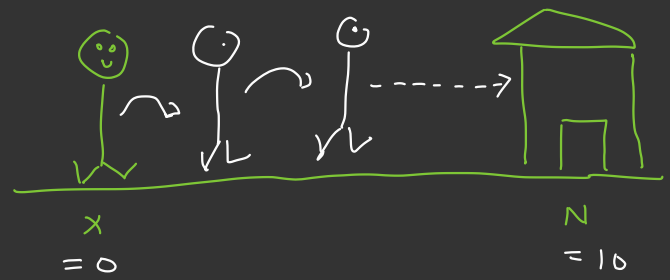
Recursion

Technique in computer science where you can derive solution to given problem by breaking the problem into smaller subproblems of the same type and solving them recursively.

Recursive fn

```
Function () {

    Function();
}
```



Iterative Code

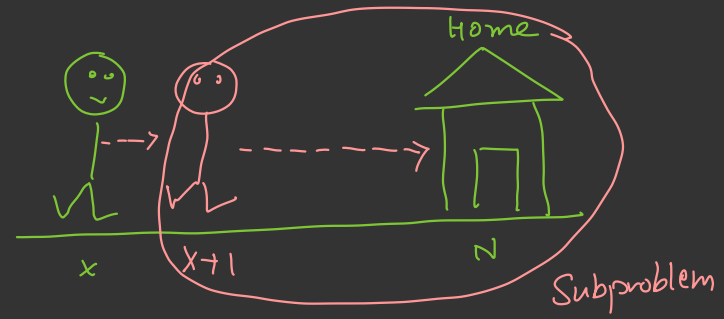
```
void goToHome(x, N){

    while x < N {
        x = x + 1
    }
    print("Reached Home")
}
```

```
void goToHome(x, N) {
    if(x==N){
        print(Reached Home)
        return;
    }
    goToHome(x+1, N);
}
```

Base case → return; → Terminate fn call

Recursive case → goToHome(x+1, N);



$x \longrightarrow N$

$x+1 \longrightarrow N$

x = 0	N = 5
↳ x = 1	N
↳ x = 2	5
↳ x = 3	5
↳ x = 4	5
↳ x = 5	5
↳ x = 6	

(b) Compute Factorial of N.

$N = 5$

output $\rightarrow 1 \times 2 \times 3 \times 4 \times 5 = 120$

Iterative Code

$\downarrow$

better than doing recursion. (Time, Space)

```
ans = 1
for (i=1; i<=N, i++) {
    ans = ans * i
}
print (ans)
```

$ans = 1 \times 1 = 1$

$= 1 \times 2 = 2$

$= 2 \times 3 = 6$

$= 6 \times 4 = 24$

$= 24 \times 5 = 120$

Actual use of Rec $\rightarrow$ in more complex problems where writing iterative can be really hard (TOH, Merge Sort)

Factorial

Recursion

$5! = 5 * 4!$ → Small subproblem

$= 5 \times 24 = 120$

Top to Down

$4 * 3! = 4 \times 6 = 24$

Down To Top

$3 * 2! = 3 \times 2 = 6$

$2 * 1! = 2$

$1 * 0! = 1$ ← smallest possible fact

Code

Magical Rule

factorial

$f(n)$

(1) Find out the Solution to Smallest possible problem.

Base Case

$f(0) = 1$

Hypothesis

(2) Assume for any given $K < N$ you know $f(K)$ is known to you.

$f(5) = 5 * f(4)$

3

Rec case

Find out Recursive relation

Assume that is known

$$f(n) = n * f(n-1)$$

$= 5 * 4!$

$= 5 * 24$

$= 120$

get computed automatically

How

Express f(n) in terms of smaller subproblems of the same type.

Examples

f(n) = 2f(n/2)
f(n) = n * f(n-1)
f(n) = f(n-1) + 2f(n-2)

Code

↳ Space (O(N))

↳ Time Θ(N)

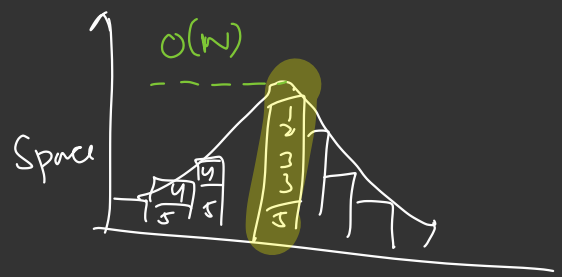
Recursion always takes extra stack memory

Time → No of Functions calls * work in Each call

Space → Max. space utilisation (when stack is having max util → Base Case)

$O(N \times 1)$

$= O(N)$



$5! \rightarrow 5 * 4! \rightarrow 4 * 3! \rightarrow 3 * 2! \rightarrow 2 \times 1! \rightarrow 1 * 0!$

$= O(N)$ calls

Using Magical Rule

Print N numbers in Increasing Order, Decreasing Order

5 Mins

1. Print numbers 1, 2, ... N Inc order
2. " " N, N-1, ... 1 Dec order

Magical Rule -

Do one step and leave the rest.

N=5

5, 4, 3, 2, 1

Dec(N) → print(N)
Dec(N-1)

You: 5 , Rec: 4, 3, 2, 1

Problem

Inc (N)

N = 5

Inc (N-1)

print(N)

1, 2, 3, 4, 5

Rec

You

dec

5, 4, 3, 2, 1

Bottom

0

1

2

3

4

5

main

Top

Bottom

inc

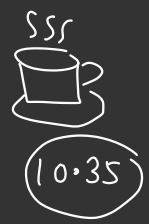
1, 2, 3, 4, 5

Top

15 Mins

Time to Try [Rec Fn's]

1. Given N, find out sum of numbers from 1 to N.
2. Given two numbers a,b multiply them using recursion. (without using *)
3. Given two numbers a,n find power a^n.



1. $N = 5$ $\quad 1+2+3+4+5 = 15$ $\quad f(n) = ?$

2. $a=5, b=3$ $\quad 5*3 = 15$ $\quad f(a,b) = ?$

3. $a=5, n=3$ $\quad 5^3 = 125$ $\quad f(a,n) = ?$

I

Rec Case

- $SUM(N) = SUM(N-1) + N$
- $SUM(0) = 0$ or $SUM(1) = 1$

$1+2+3+\ldots N-1 + N$

$\parallel$

$SUM(N-1) + N$

II

Multiply a, b

$5 * 6 = 30$ Add a b times

$a = 5$

$b = 6$

$f(5,6) = 5 + \overbrace{5 + 5 + 5 + 5 + 5}^{5 \text{ times}}$

$= 5 + f(5,5)$

Rec case $\Rightarrow$ $f(a,b) = a + f(a, b-1)$

Base case

Base case $\Rightarrow$ $f(a,0) = 0$

3, n times $= \underline{3} + \overbrace{3, 3, 3, 3}^{\text{Add } n-1}$

$f(3, 5) = \underline{3} + \overline{3, 3, 3, 3}$

$= 3 + f(3, 4)$

(3) a, n

$5^4 = 5 * 5^3$

Rec case

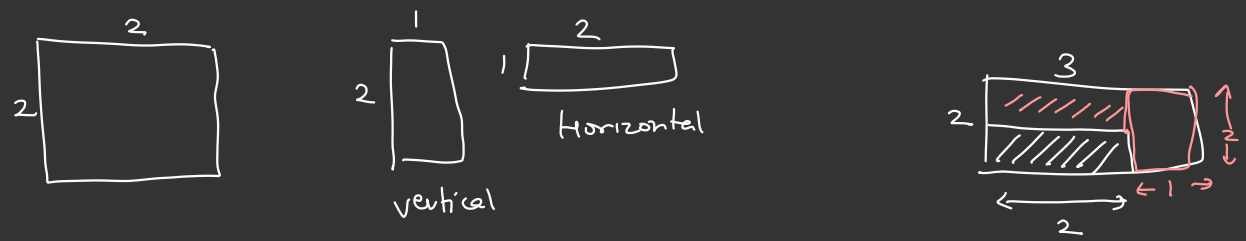
$$\left.\begin{aligned} f(a, n) &= a * f(a, n-1) \\ f(a, 0) &= 1 \end{aligned}\right] \rightarrow \text{Code} \quad \boxed{\text{DIY}}$$

Challenge

Given a floor of size N X 2, which can be filled with tiles of size 1 X 2 or 2 X 1. Find the number of ways to build the floor.

$f(n-1) + f(n-2)$

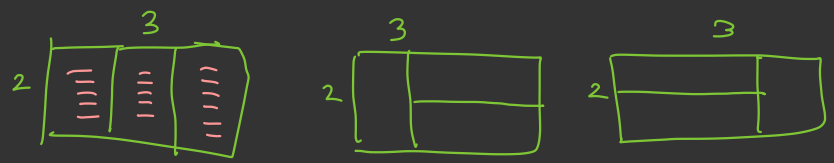
N = 2



ways

N = 2 2 ways

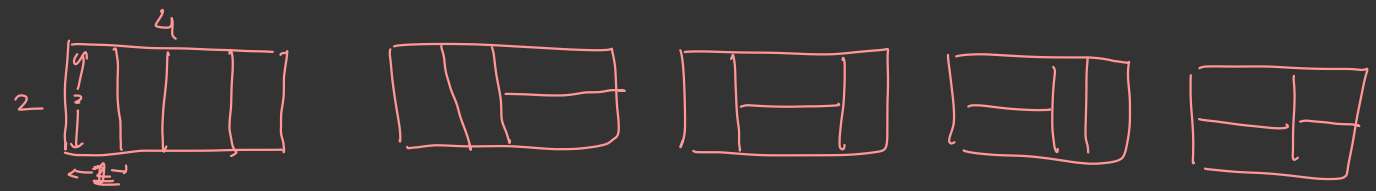
N=3



3 ways

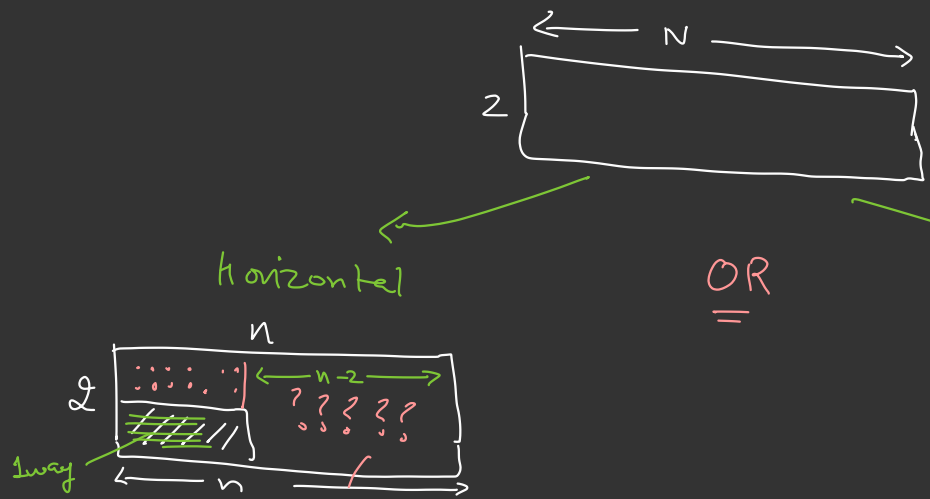
N=4

5 ways



f(N)

f(N)



$$f(n) = x + y$$
$$= f(n-1) + f(n-2)$$

Total ways

1. x ways

$= 1.f(n-2)$

1. y ways

$= 1.f(n-1)$

No floor

$f(0) = 1$

$f(1) = 1$

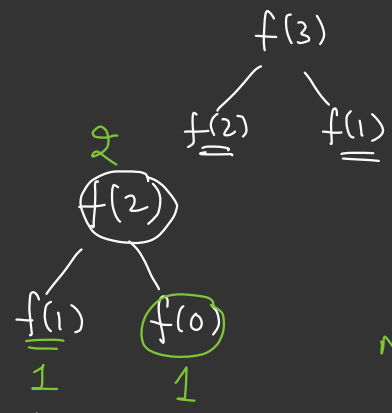
$f(2) = 2$

$f(3) = 3$

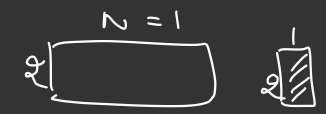
$f(4) = 5$

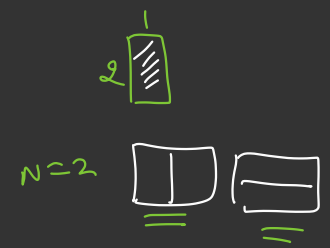
$f(5) = 8$

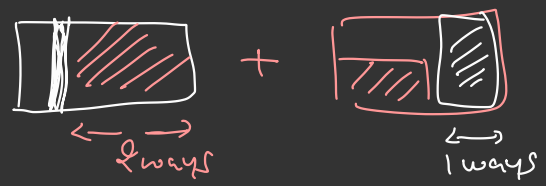
DIY

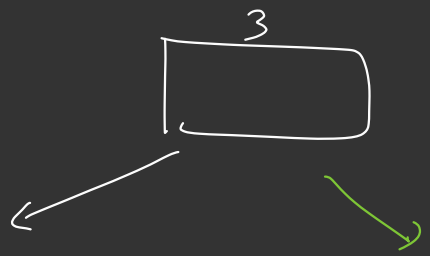


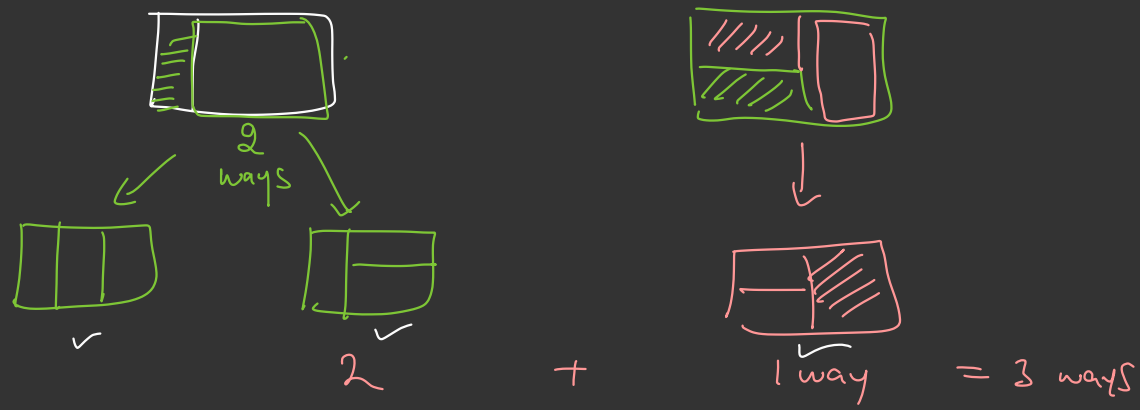
Floor size N×2

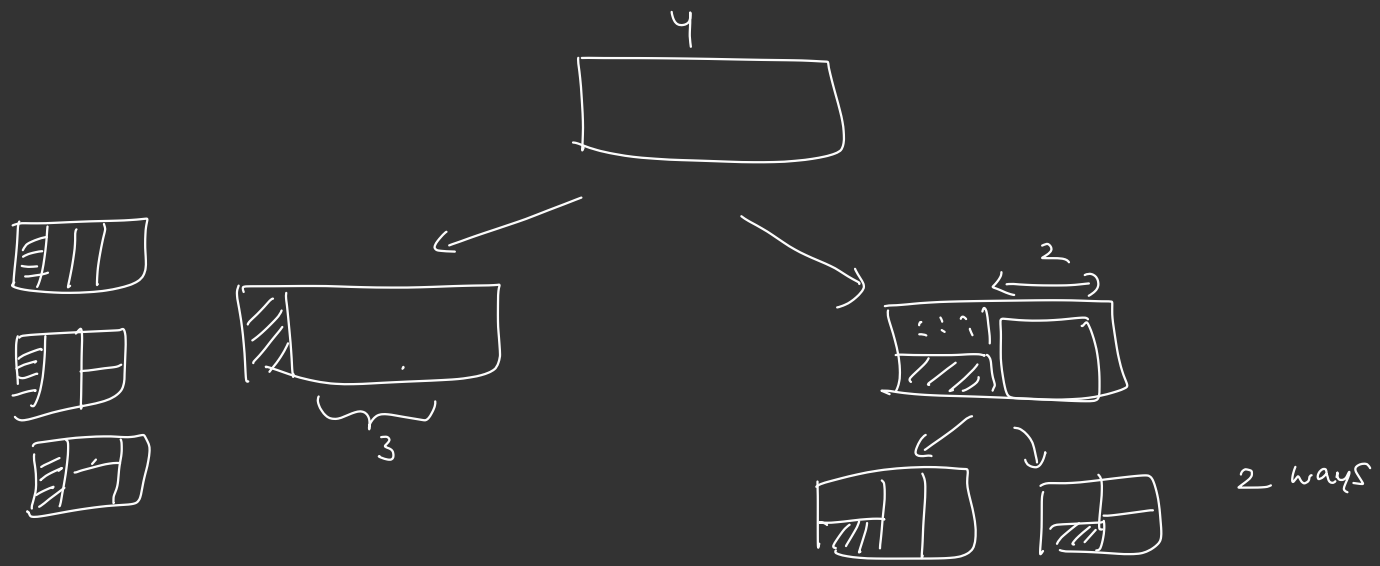




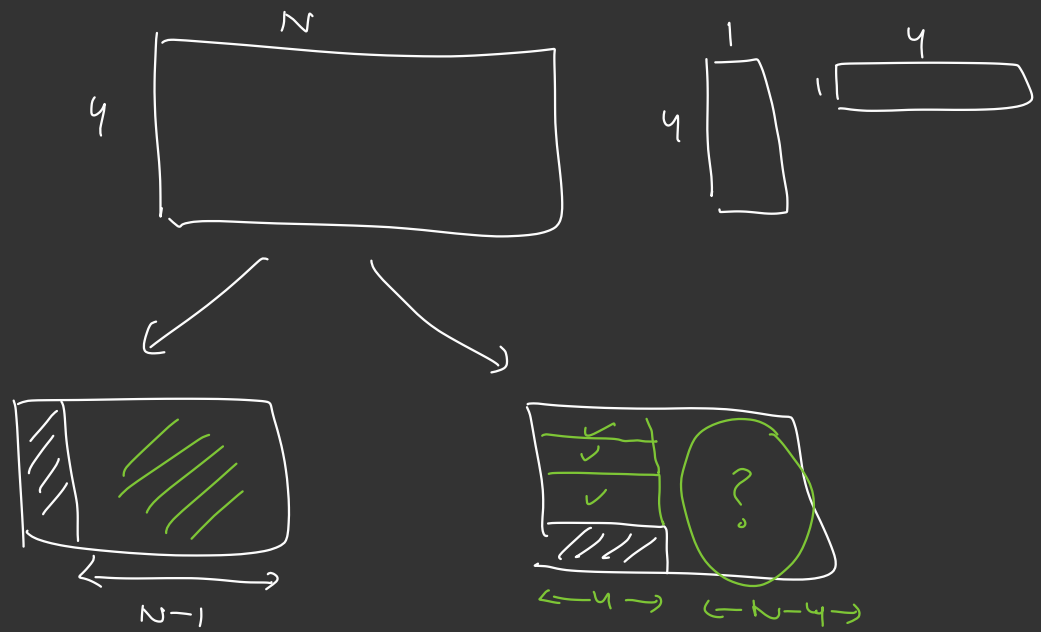





2
ways
2
+
1 way
= 3 ways


4
3
2
2 ways

$N \times 4$



$$f(N) = \underline{f(N-1)} + \underline{f(N-4)}$$

a b c = a+b

1, 1, 2, 3, 5, 8

Better to do iteratively