2nd class → Recursion.

Agenda → Problem solving

→ New addition to the pattern of recursive

Q → There are N persons, who want to go to a party. There is a constraint that any person can either go alone or can go in a pair. Calculate the no. of ways in which N persons will go to party.

Ex $N = 3$ ans 4

(A) (B) (C) $\Rightarrow$

all alone

[A] [B] [C] $\rightarrow$ 1 way

a, b makes a pair

[A, B] [C] $\rightarrow$ 1 way

a, c makes a pair

[A, C] [B] $\rightarrow$ 1 way

[B, C] [A] $\rightarrow$ 1 way

B, C makes a pair

4 ways

⇒ A B C - - - - - - - - - - N] N persons

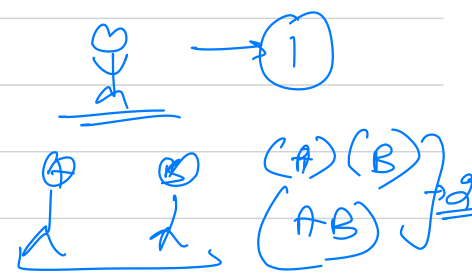
Calculate a function $f(N)$ → returns no. of ways in which N persons can go to a party

We don't know the ans for $f(N)$

→ Base Case

for $N = 1$ → ans $= 1$

$N = 2$ → ans $= 2$



→ Recursion Assumption $f(N)$ → works fine

A B C D - - - - - - - N] n person

alone

or

go in pair

Case1 (A goes alone)

$f(n)$ depends on

$f(n-1)$

$f(N) \rightarrow f(N-1)$

I will go alone

(A) $\overbrace{B\ C\ D}$

(A) (B) (C) (D)

(A) (B C) (D)

(A) (C D) (B)

(A) (B D) (C)

What we are saying is,

if we have N friends and the first one says "I will go alone" then the total no. of ways in which N people can go depends on the no. of ways N-1 people will go. Why?? Because one of them made a decision.

Case 2 Now let's assume the person 'A' said

"I will go in pair"

A B C D

A pairs with B | A pairs with C | A pairs with D

(A B) (C) (D) | (A C) (B) (D) | (A D) (B) (C)

(A B) (C D) | (A C) (B D) | (A D) (B C)

2 | 2 | 2

$f(n-2) + f(n-2) + f(n-2) = 3\,f(n-2)$

6

A can make pair with multiple persons.

$f(N) \rightarrow$ no. of way for N persons

if A make a pair $\rightarrow$ people left for making decision $\Rightarrow N-2$

$f(N)$ depends on (no. of ways in which A makes pair) $* f(N-2)$

$(N-1)$

$f(N) \rightarrow (N-1) * f(N-2)$

→ Self work

Sum (OR)

return $f(N-1) + (N-1) f(N-2)$

$$f(N) = f(N-1) + (N-1) f(N-2)$$

Q) Given two number 'a' and 'b' Calculate a^b recursively

→ $a = 3$ $b = 2$

ans → $3^2 \to \underline{9}$

$$\underline{a^b = (a \times a^{b-1})}$$

$b-1$
$b-2$
$b-3$
$b-4$
⋮
1

b

Base Case → if (b==0) return 1;

Recursive Intuition → $\underline{f(a, b)} \to a^b$ → Anyhow if get $\underline{f(a, b-1)}$

correctly

Self work → return $\underline{a \times f(a, b-1)}$

Determine Time & Space Complexity

TC → O(b)

SC → O(b)

May be we can optimize

$$a^b = a \times a^{b-1}$$

But

$$a^b = a^{b/2} \times a^{b/2}$$ if $b \rightarrow$ even

or

$$a^b = a \times a^{b/2} \times a^{b/2}$$ if $b \rightarrow$ odd

$$a^b = a^{b/2}$$ (multiply with itself)

$$a^{b/2} \rightarrow a^{b/4}$$

$$a^{b/4} \rightarrow a^{b/8}$$

$\vdots \qquad \vdots$

TC → no. of operation → $O(\log_2 b)$

$SC \to O(\log_2 b)$

b
↓
$b/2$
↓
$b/4$
↓
$b/8$
↓
⋮
1

K terms

$\frac{b}{2^k} = 1$ → last term

$2^k = b$

taking log both sides

$\boxed{\log_2 b = k}$

$b = 1024$

$\log_2 1024$

10

$f(a, b)$

$f(a, b/2)$

$f(a, b/4)$

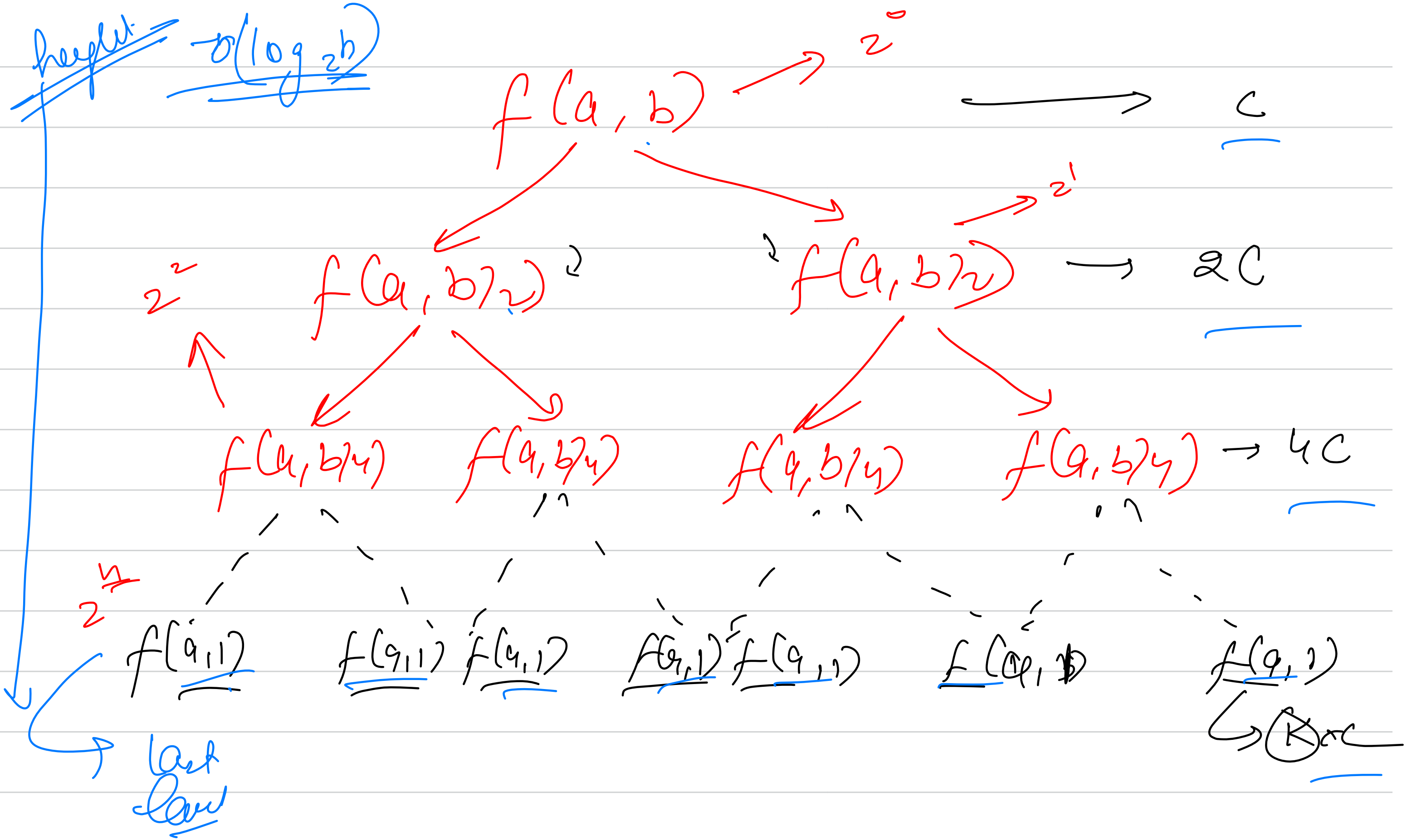
$f(a, b/8)$

$\vdots$

$f(a, 1)$

Recursion tree

$a \log_2 b$



height → $O(\log_2 b)$
$f(a, b)$
2^0
c
$f(a, b/2)$
$f(a, b/2)$
2^1
$2c$
2^2
$f(a, b/4)$
$f(a, b/4)$
$f(a, b/4)$
$f(a, b/4)$
$4c$
2^4
$f(a,1)$
$f(a,1)$
$f(a,1)$
$f(a,1)$
$f(a,1)$
$f(a,1)$
$f(a,1)$
$K \times c$
last level

$$\underbrace{C + 2C + 4C \; - - - - - - \; kC}$$

No of terms $\underline{\log_2 b}$

what is $\underline{k}$

$$c\left(1 + 2 + 4 \; - - - - - - - - \; 2^{\log_2 b}\right)$$

$$c\left(\underbrace{1 + 2 + 4 + 8 \; - - - - - \; b}\right)$$

$$c\left(\frac{1 \times \left(2^{\log_2 b} - 1\right)}{2 - 1}\right) \Rightarrow c\left(b - 1\right)$$

$\underline{\text{const}}$

$\underline{O(b)}$

Q => Given a value of N, print the pattern of N rows recursively.

```
* * * *
* * *
* *
*
```

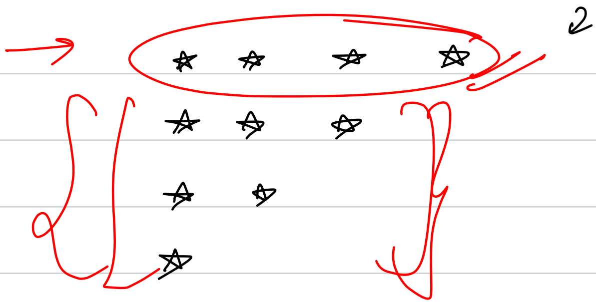
N = 4

SANKETIO

```
* * * * *
* * * *
* * *
* *
*
```

N=5

No loop allowed
Use only one func



→ 4

→ 3

→ 2

→ 1

→ 0

N=4

↳ f(n) → which print N rows of the pattern

Recursion intuition → Anyhow print the N-1 rows

Self work → I will print myself (loop)

(later we will eliminate)

Recursion prints the row

$\boxed{\text{loop}}$ prints the column for the row

elimination

$f(n, i) \rightarrow$ prints the i^{th} column of n^{th} row

Recursive assumption →

if the value of i (column) is less than n, then recursively print all the columns to the right

else if i (column) is equal to n, recursively print the rows below

Self work

if i < n, I will print the current column char

else I will print new line

HW Given a value of N, print this pattern

Recursively

```
*
*   *
*   *   *
*   *   *   *
*   *   *   *   *
```

$\boxed{N=5}$

Qs Given an array, print all the subsets of the array

[1, 2, 3]

→ [ ]
[1]
[1, 2]
[1, 2, 3]
[2, 3]
[2]
[3]
[1, 3]

print

No. of subsets of a given set → 2^n

$$[\overset{2}{1},\ \overset{2}{2},\ \overset{2}{3}] \rightarrow 2^3$$

inc

not inc

Base Case

[ ] → empty

[ ]

$f([\,]) \rightarrow$ subset

Recursion assumption

Anyhow any cost

[1, 2, 3]

Self work

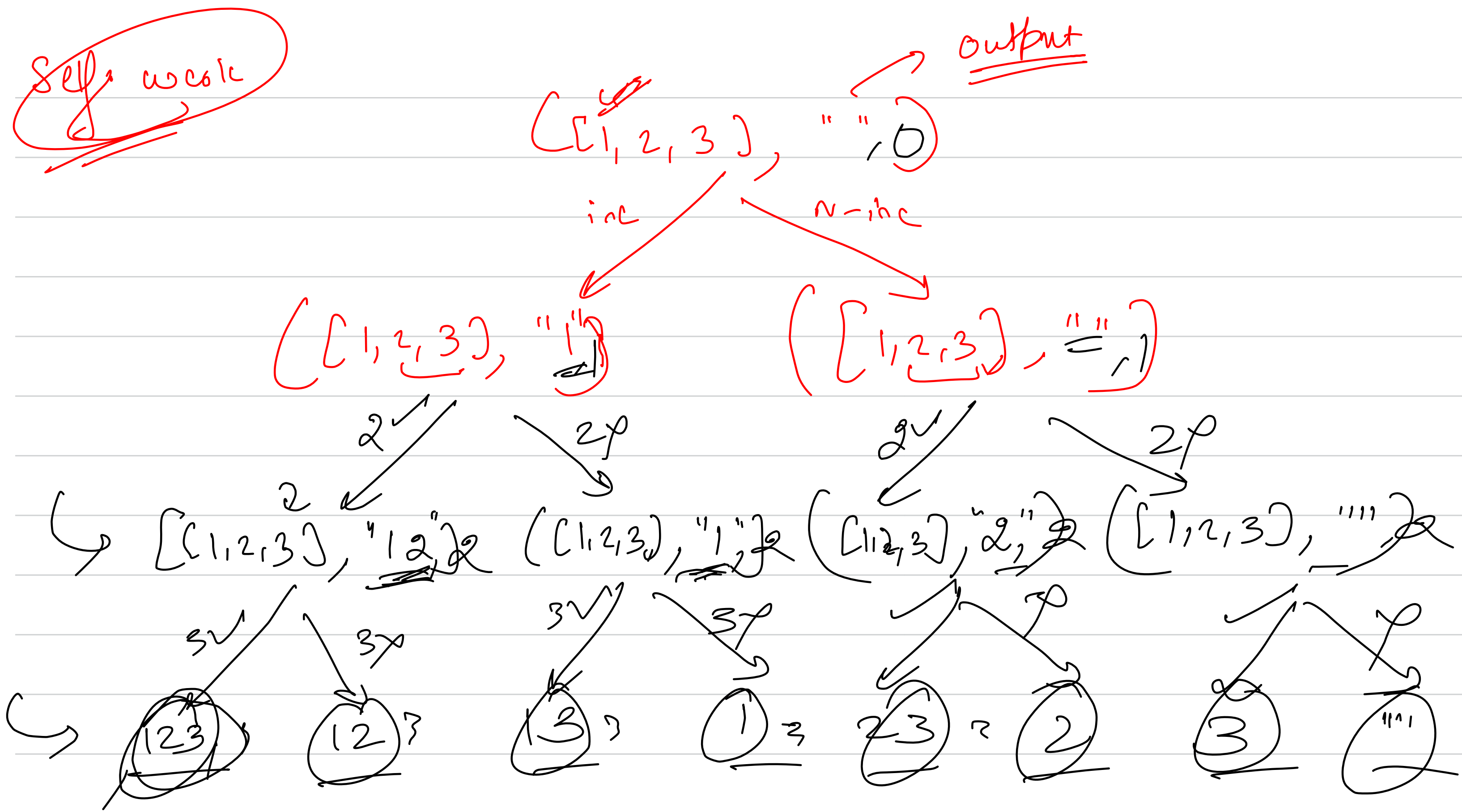
[1]
[2, 1]
[3, 1]
[2, 3, 1]

[ ]
[2]
[3]
[2, 3]

include

not include

[ ]
[2]
[3]
[2, 3]



Self work
output
([1, 2, 3], "", 0)
inc
n-inc
([1, 2, 3], "1", 1)
([1, 2, 3], "", 1)
2✓
2✗
([1,2,3], "12", 2)
([1,2,3], "1", 2)
([1,2,3], "2", 2)
([1,2,3], "", 2)
3✓
3✗
123
12
13
1
23
2
3
""

array

$f(arr, i, osf) \longrightarrow f(arr, i+1, osf + arr[i])$ incl

denotes current value's idx

$\rightarrow f(arr, i+1, osf)$ not incl

HW → Given a value n, print all the binary strings of size 'n' which have no consecutive one.

n = 3 ⟶

0 0 0
0 0 1
0 1 0
1 0 0
1 0 1

print manually

Q-1 Given an array, check if it is sorted or not

[1, 2, 3] → Yes

[3, 2, 1] → NO

[1] → sorted   Base Case

[1, 2, 3]

arr[i] < arr[i+1] → return true

return false