

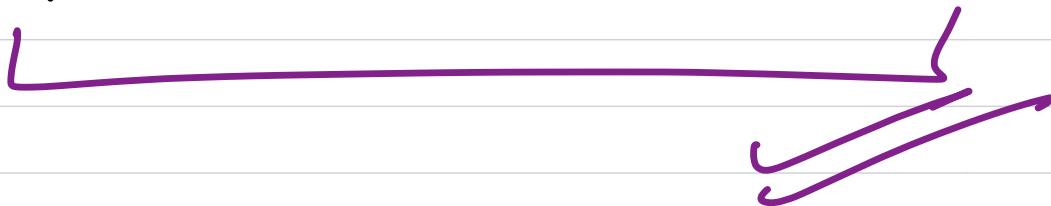
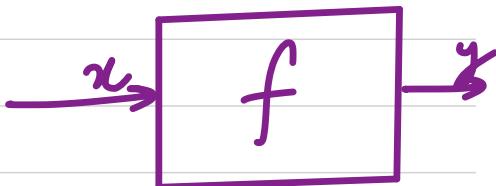
# Recursion

→ Recursion is not just a computer science topic  
It correlates with maths as well:

Mathematically, recursion means a function

calling itself.

$$f(x) = f(f(x'))$$



$$f(a, b) = a \times f(a, b-1)$$

this function

calculates

$$\underline{\underline{a^b}}$$

func<sup>i</sup> definition

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

Ex  $f(2, 3) \rightarrow 2^3 \rightarrow 8$

$$2^4 = 2 \times 2^3$$

$f(2, 4) \rightarrow 2^4 \rightarrow 16$

In computer science, recursion means function

calling itself.

function fun(x) {  
...  
...  
...  
fun(x-1);  
...  
}

```
function gem(x) {  
    console.log(x);  
    y  
}
```

```
function fum(x) {  
    ...  
    ...  
    gem(x-1)  
    fum(x-1);  
    ...  
    ...
```



This is not recursion



This is recursion

}

Example-1

Factorial

factorial of any value  $n \Rightarrow n \times (n-1) \times (n-2) \dots \times 2 \times 1$

$\overbrace{\hspace{10em}}$   $\rightarrow n!$   
 $\overbrace{\hspace{1em}}$

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

$$4! = 4 \times 3 \times 2 \times 1 \rightarrow 24$$

$$3! = 3 \times 2 \times 1 \rightarrow 6$$

$$f(n) = n \times f(n-1)$$

this function

returns  $n!$

The mathematical representation of a recursive function is called Recurrence Relation

$$\begin{aligned}
 5! &\xrightarrow{120} \\
 \downarrow & \quad \downarrow \\
 s \times 4! &\approx 2^4 \\
 \downarrow & \quad \downarrow \\
 4 \times 3! &\approx 6 \\
 \downarrow & \quad \downarrow \\
 3 \times 2! &\approx 2 \\
 \downarrow & \quad \downarrow \\
 2 \times 1! &\approx 1
 \end{aligned}$$

we already  
know them

What is  $11! \Rightarrow 1$

Recursion → In recursion , a function calls itself , where it tries to solve a smaller problem and then calc value of a larger problem

# P M I (principle of Mathematical Induction)

Q. What is the sum of first  $N$  natural numbers ??

$$\text{Ans} \rightarrow \frac{n \times (n+1)}{2}$$

## 3 step process

- 1) Base Case → the smallest value for which we can directly verify the ans-
- 2) Assumption
- 3) Verification / self task

for  $n=1$   $\rightarrow \frac{1 \times (1+1)}{2} \rightarrow$  is correct  
Base Case

for some  $n=k \rightarrow$  the formula is correct

i.e.  $\frac{k \times (k+1)}{2}$

Let's prove ourselves that formula is correct

for  $n = k+1$

$$1 + 2 + 3 + \dots + k + k + 1$$


$$\rightarrow \frac{k \times (k+1)}{2} + (k+1)$$

$$\frac{k \times (k+1) + 2(k+1)}{2} \rightarrow \frac{(k+1)(k+2)}{2}$$

By formula

$$\rightarrow n = (k+1)$$

$$\frac{n \times (n+1)}{2} \rightarrow \frac{(k+1) (k+1+1)}{2}$$

$$= \frac{(k+1) (k+2)}{2}$$

# # Components Of Recursion

- 1) Base Case
- 2) Assumption
- 3) Self work

Write a recursive sol' for Factorial of  $n$

We will write a func<sup>i</sup>

$f(n)$



returns  $n!$

Base Case  $\rightarrow (n == 1)$        $f(1) \Rightarrow \underline{\underline{1}}$

if  $(n == 1)$       return 1

Assumption  $\rightarrow$  assume the function works  
fine for  $n-1$

$f(n-1)$  correctly calls  $(n-1)!$

~~Self work~~  $\rightarrow f(n) = n \times f(n-1)$

Power function  $\underline{\underline{a^b}}$

$f(a, b)$



returns  $\underline{\underline{a^b}}$

Base Case → if ( $b == 0$ ) then ans is 1

Assumption → assume func<sup>n</sup> works fine for  $b-1$   
i.e.  $f(a, b-1)$  is correct.

Self work  $\rightarrow$   $a^b = \underbrace{a \times a^{b-1}}$

$$f(a, b) \leftarrow a \times f(a, b-1)$$

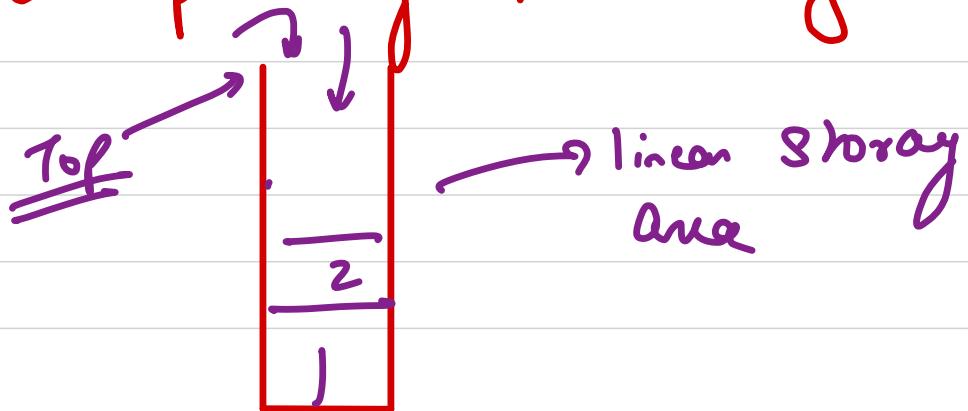
↓  
Conut

$$\underline{\underline{27}}^{51} \quad \underline{\underline{27}} \times \underline{\underline{27}}^{50} \Rightarrow y$$

Q What happens when we call a function ??

In our memory, we have a lot of things going on.

One part of the memory is call stack



1 function fun(x)

2 let a = x + 0;

3 return a;

4 }

5 function gen(y)

6 let b = y + 20

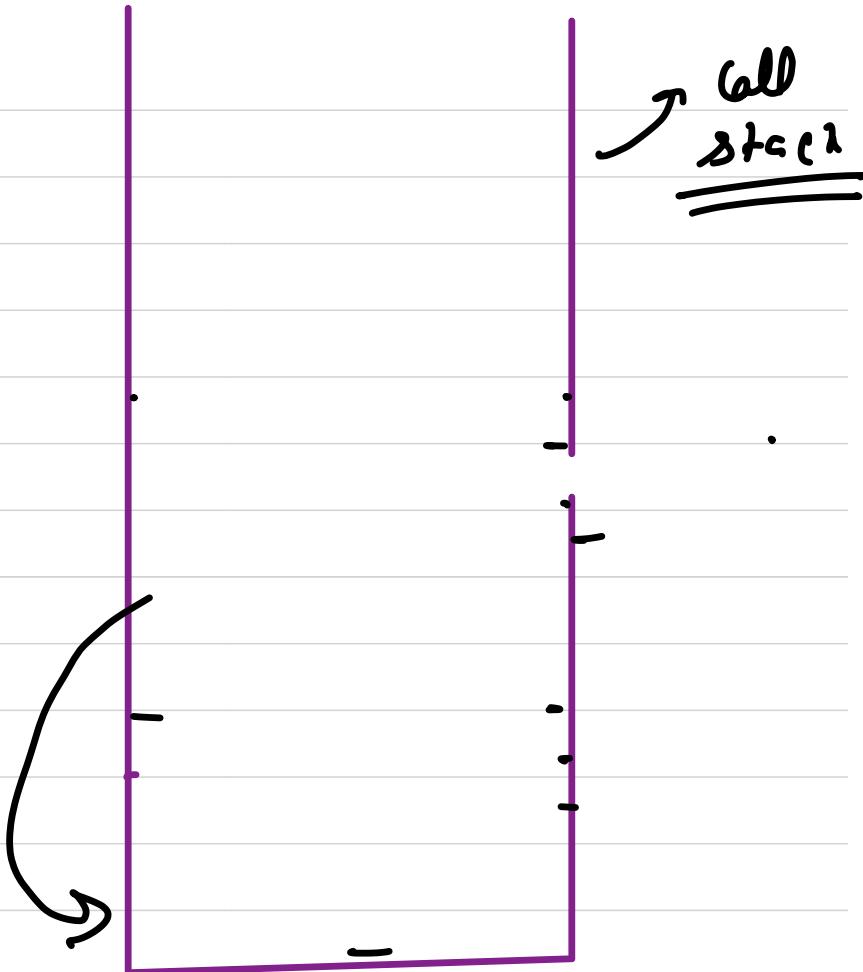
7 let z = fun(b);

8 return z;

9 }

10 console.log(gen(10))

→ call stack



Whenever we call a new function it adds a new entry in the stack.

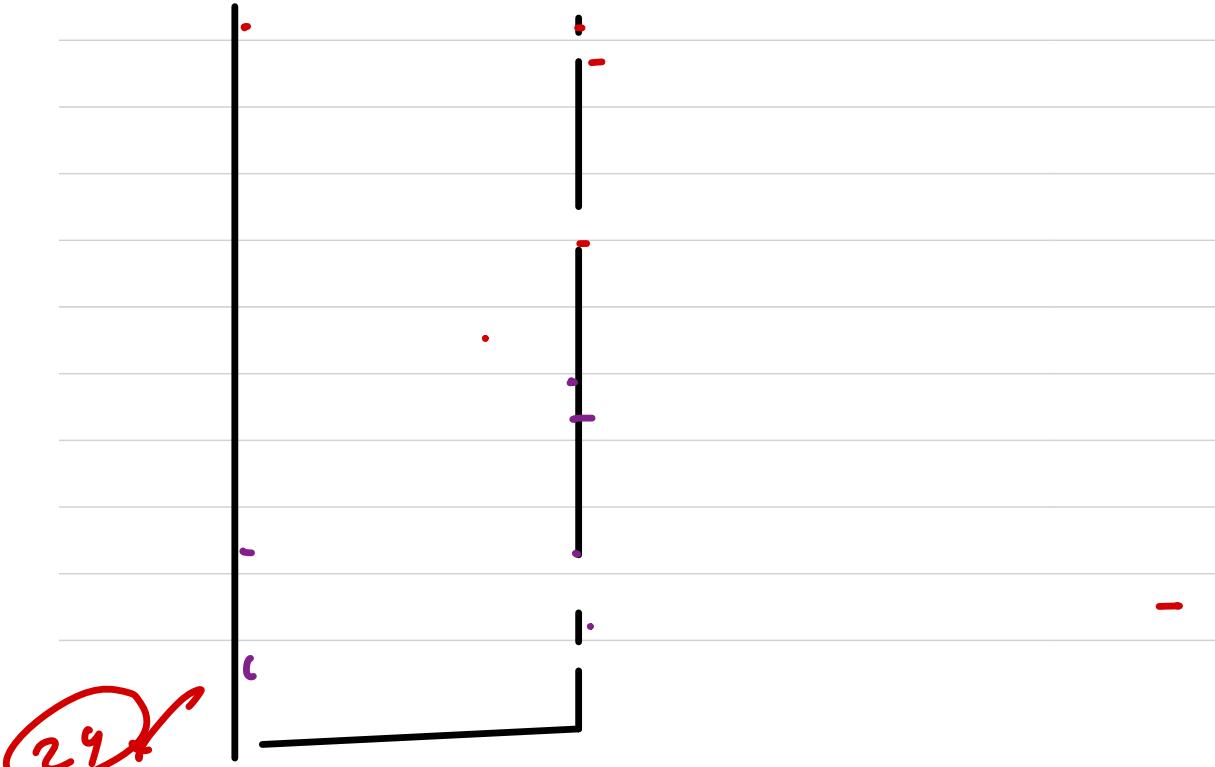
This new entry is called stack trace.

Inside a trace we store all the variables and more data like current line of execution of function.

When we hit a return , the stack trace is removed & value is given back to the caller.

```
1 function fact(n) { // this function should calculate n!
2     // Base case
3     if(n == 1) return 1;
4     let assume = fact(n-1); // that fact of n-1 is correct
5     return n*assume;; // self work
6 }
```

fact(4)



~~$\phi^n$~~  Given a value  $n$ , calc  $n^{\text{th}}$  fibonacci.

starting value

→ 0, 1, 1, 2, 3, 5, 8, 13, 21 ...  
0<sup>th</sup> fib 1<sup>st</sup> fib 2<sup>nd</sup> fib 3<sup>rd</sup> fib 4<sup>th</sup> fib 5<sup>th</sup> fib 6<sup>th</sup> fib . - - - - -

$n = ?$

ans = 13

Solve it recursively

$n^{\text{th}}$  fib

$f(n)$



return  $n^{\text{th}}$

fib

Base Case

$$\begin{aligned}f(0) &\rightarrow 0 \\f(1) &\rightarrow 1\end{aligned}\quad \left.\right\} \xrightarrow{\text{start}} \underline{\text{start}}$$

Assumption → we assume  $f^{(n-1)}$  works  
fine &  $f^{(n-2)}$  works fine

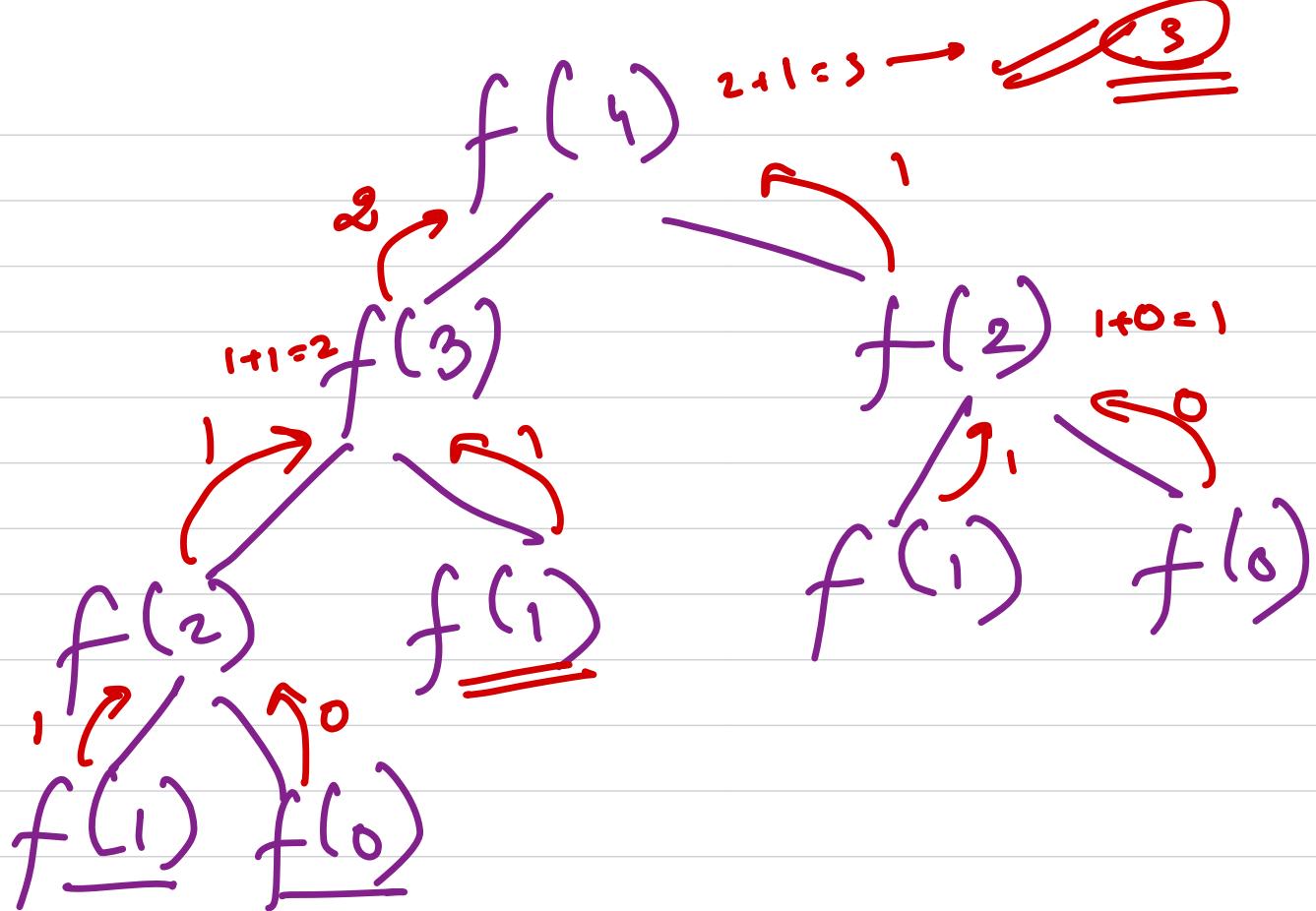
Self work →  $f^{(n-1)} + f^{(n-2)}$

$$\boxed{f(n) = f(n-1) + f(n-2)}$$

```
1 function fib(n) {  
2     // base case  
3     if(n == 0) return 0;  
4     if(n == 1) return 1;  
5     // assumption  
6     let a = fib(n-1);  
7     → let b = fib(n-2);  
8     // self work  
9     return a+b;  
10 }  
11  
12 console.log(fib(3))
```

Recursion takes  
Extra Space  
in memory

~~tree~~



$$\underline{f(n)} + f(n+1) = f(n+2)$$

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

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

Say  $n+2 = x$

$$\underline{f(x-2) + f(x-1)} = f(x)$$

~~Q.~~ Given a number  $n$ , print the first  $N$  natural no. in increasing order, recursively

$$n = 5$$

Base  $\rightarrow n = 1 \rightarrow \text{console.log}(1)$

Ans  $\rightarrow$  1  
2  
3  
4  
5

assume  $\rightarrow$  if someone can give me the first  $4$  natural no.'s then we can do something

Self  $\rightarrow$  point 5

$f(n)$  $f(n-1)$ 

console.log(n)

which print

first  $n$  natural

no.s

Assume  $f(n-1)$  works correctly

s  
↓  
→

s  
4  
3  
2  
1  
↓  
/ /

f(n)

= console.log(n)

f(n-1)

which prints  
first n natural no  
in dec order

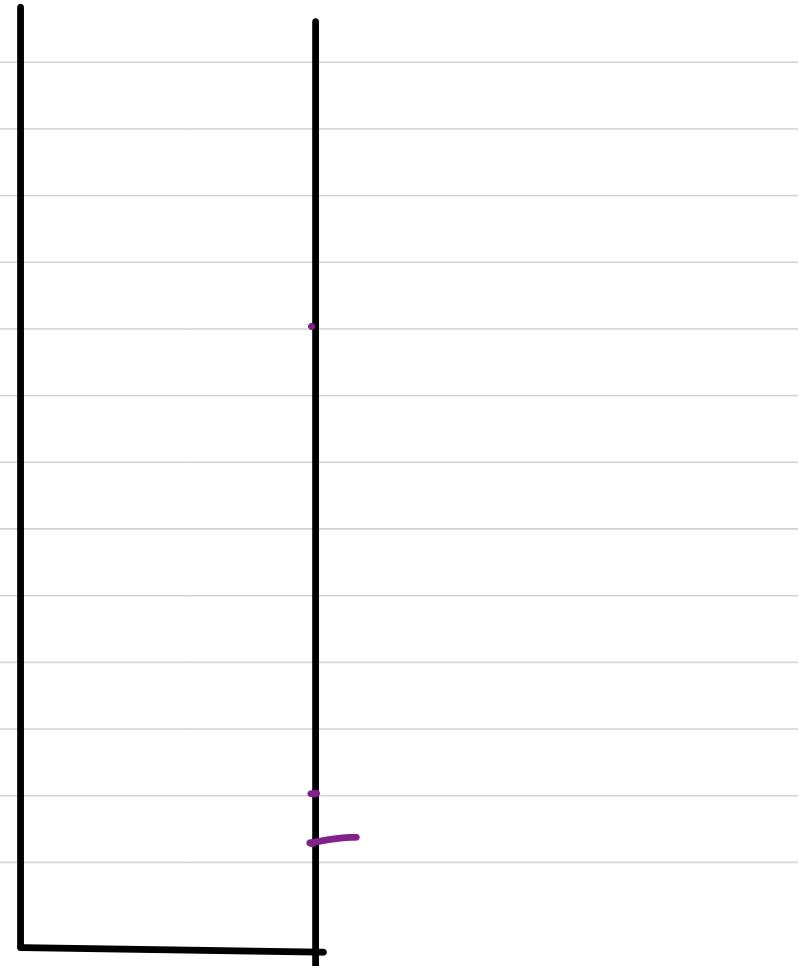
$f(n)$

5  
4  
3  
2  
1 }  
  

`console.log( $f_n$ )`  
 $f(n-1) \rightarrow$  assume it works few

```
1  function print_inc(n) {  
2      // base case  
3      if(n == 1) {  
4          console.log(1);  
5          return;  
6      }  
7      print_inc(n-1);  
8      console.log(n);  
9  }  
10  
11 → print_inc[3]||
```

1  
2  
3



```
1 function print_dec(n) {  
2     // base case  
3     if(n == 1) {  
4         console.log(1);  
5         return;  
6     }  
7     console.log(n); // Self task  
8     print_dec(n-1); // recursive  
9 }  
10  
11 print_dec(3)
```

3  
2  
1

Q. There are  $n$  friends, who want to go to a party. They can either go alone or go in a pair. Calc the no. of ways in which  $n$  friends can go to party. (Try Recursivity)

Ex  $\underline{n=3}$

ans  $\rightarrow 4$

- |             |     |
|-------------|-----|
| (A) (B) (C) | → ✓ |
| (A B) (C)   | → ✓ |
| (A C) (B)   | → ✓ |
| (A) (B C)   | → ✓ |

$n=9$

A

B

C

D

Base Case ] →  $n \geq 1$  → Ans → 1  
                        ] →  $n = \leq 2$  → Ans → 2 ←  
A B → (A) (B) ]  
                        (A B) ]

$n=9$

A      B      C      D

$$f(n) = f(n-1) + (n-1) \times f(n-2)$$

return the  
no. of ways

in which  
 $n$  friends can go

$n$  friend  
said to go  
alone

$n$  friend  
said to  
make a pair

$$\overline{\overline{n=9}}$$

A

B

C

D

n<sup>th</sup> foiled

alone fair

what if D goes alone ??

- |      |     |     |     |
|------|-----|-----|-----|
| (A)  | (B) | (c) | (D) |
| (AB) | (c) | (D) |     |
| (AC) | (B) | (D) |     |
| (BC) | (A) | (D) |     |

What if  $n^{\text{th}}$  friend asked to make a pair ??

$$n=4$$

A

B

C

D

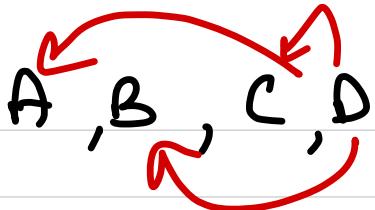
$\rightarrow \underline{n^{\text{th}} \text{ friend}}$

In how many ways  $n^{\text{th}}$  friend  
can make a pair ??  $\rightarrow (n-1)$

$$\underline{(n-1) \times f(n-2)}$$

$$3 \times 2 \\ = 6$$

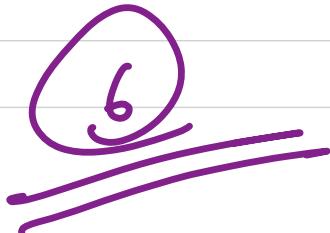
After making a pair how many elements left ??  $\rightarrow (n-2)$



$$\begin{bmatrix} (D)(C) & (A)(B) \\ (D)(C) & (A)(B) \end{bmatrix}$$

$$\begin{bmatrix} (D)(B) & (A)(C) \\ (D)(B) & (A)(C) \end{bmatrix}$$

$$\begin{bmatrix} (A)(D) & (B)(C) \\ (A)(D) & (B)(C) \end{bmatrix}$$



$f^{(n-2)} + f^{(n-2)}$   
 $+ f^{(n-2)}$   
 $(n-1) \times \underline{\underline{f^{(n-2)}}}$

$\Omega^n$  Given a number  $n$ , calculate the no. of  
binary strings of length  $n$  with no  
Consecutive One.

$$\underline{\underline{n=3}}$$

$$\text{Ans} \rightarrow \underline{\underline{5}}$$

000  
001  
010  
100  
101



$n$

1

2

3

4

5

ans

2

3

5

8

13

fibonacci

strings  
0, 1

00, 01, 10

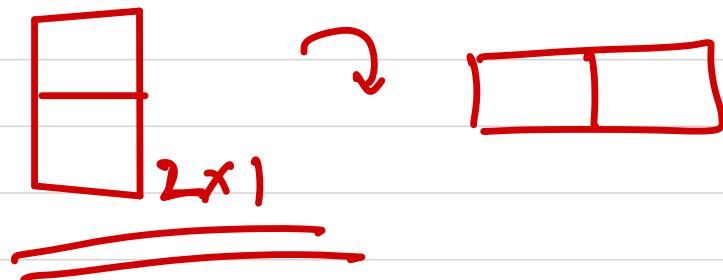
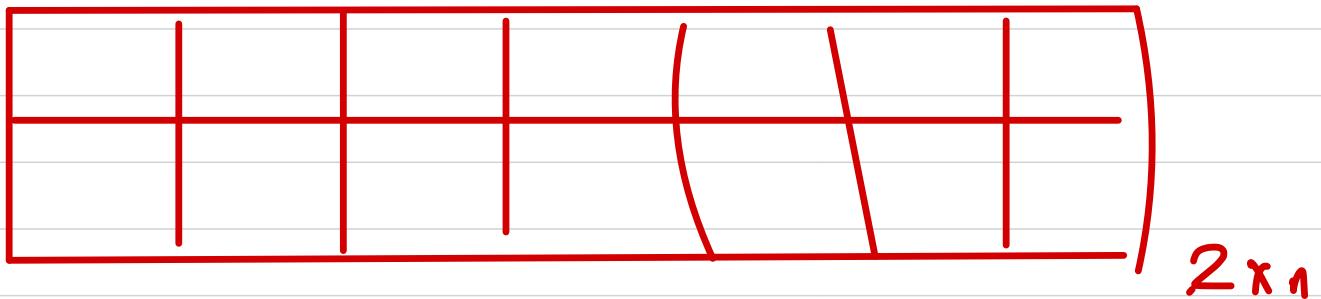
000, 001, 010,  
100, 101

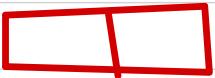
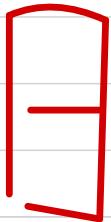
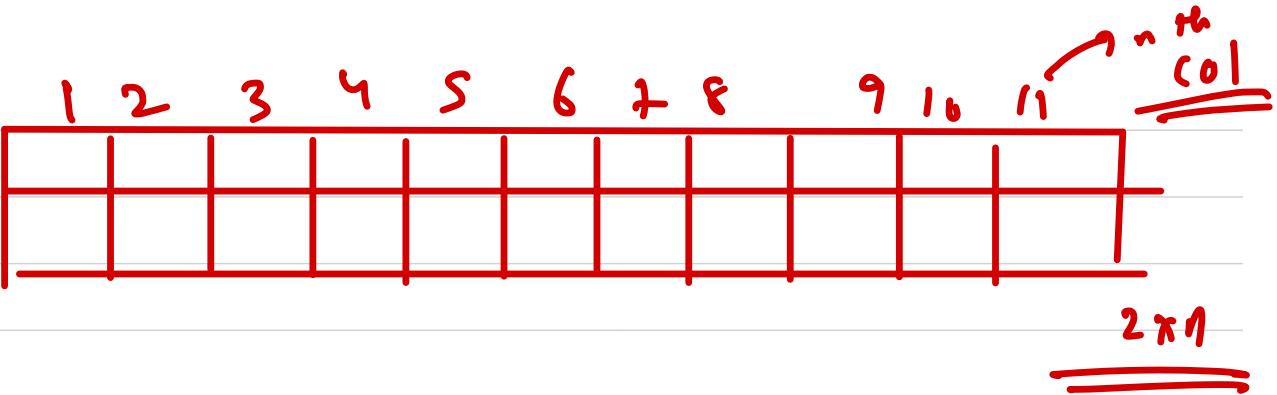
0000, 0001, 0010  
0100, 1000, 1001  
1010, 0101

Base Case  $\rightarrow$

if ( $n \geq 1$ ) return 2  
if ( $n \leq 2$ ) return 3

$$f(n) = \underline{\underline{f(n-1)}} + \underline{\underline{f(n-2)}}$$





On the  $n^{\text{th}}$  column you can put a tile either vertically or horizontally

or horizontally

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

value of  
no. of ways to  
put tiles on

$2 \times n$  board

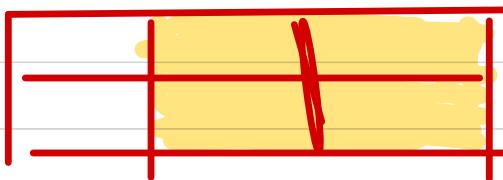
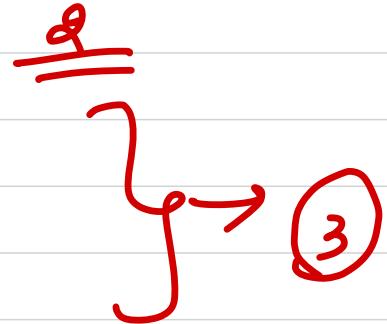
$n^{\text{th}}$  board  
via putting  
vertical  
tile

+

$n^{\text{th}}$  board with  
 $(n-1)^{\text{th}}$  board filled  
with horizontal  
tile

Base Case

$$\begin{aligned} n &= 1 \rightarrow 1 \\ n &= 2 \rightarrow 2 \end{aligned}$$



~~Q~~

Given an array , print all subsets of  
the array .

→ [1, 2, 3]

1

2

3

1 2

1 3

2 3

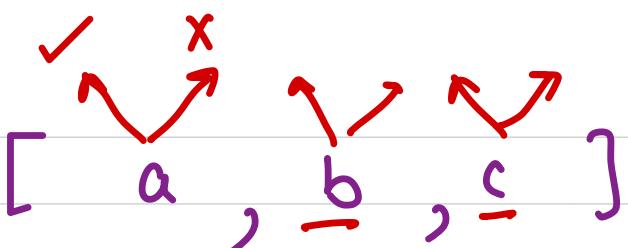
1 2 3

empty

if there is a set of length  $n$ ,  
how many subsets are possible ??

$$\underline{\underline{2^n}}$$

Permutation Combination

 [ a , b , c ]

$2 \times 2 \times 2 \dots$         $\rightarrow b c$

  $\begin{matrix} \checkmark & x & \checkmark \\ x & \checkmark & x \end{matrix} \rightarrow a c$

$\begin{matrix} & \checkmark & \\ x & \checkmark & \end{matrix} \rightarrow b$

$\begin{matrix} \checkmark & \checkmark & \checkmark \end{matrix} \rightarrow a b c$

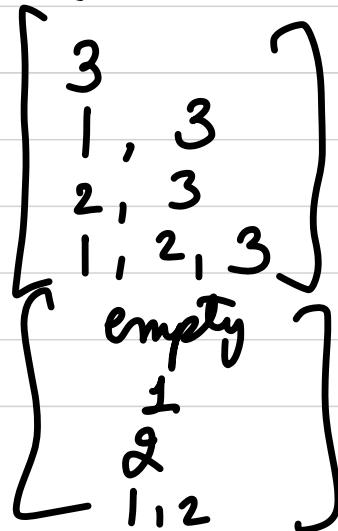
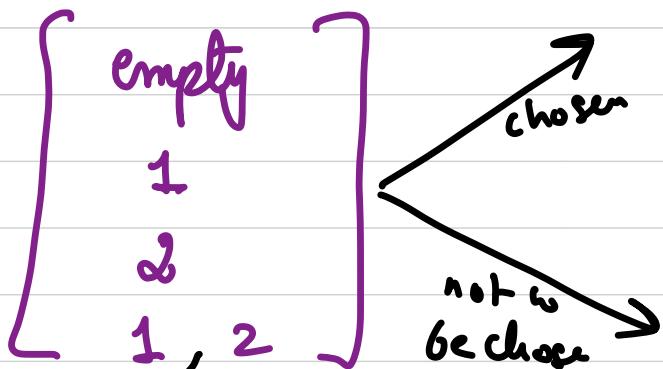
$\begin{matrix} x & x & x \end{matrix} \rightarrow \text{empty}$

$\begin{matrix} \checkmark & \checkmark & x \\ x & x & \checkmark \end{matrix} \rightarrow a b$   
 $\therefore c$

We can use the same formula to build a  
recursion solution.

$$[1, 2, 3]_n$$

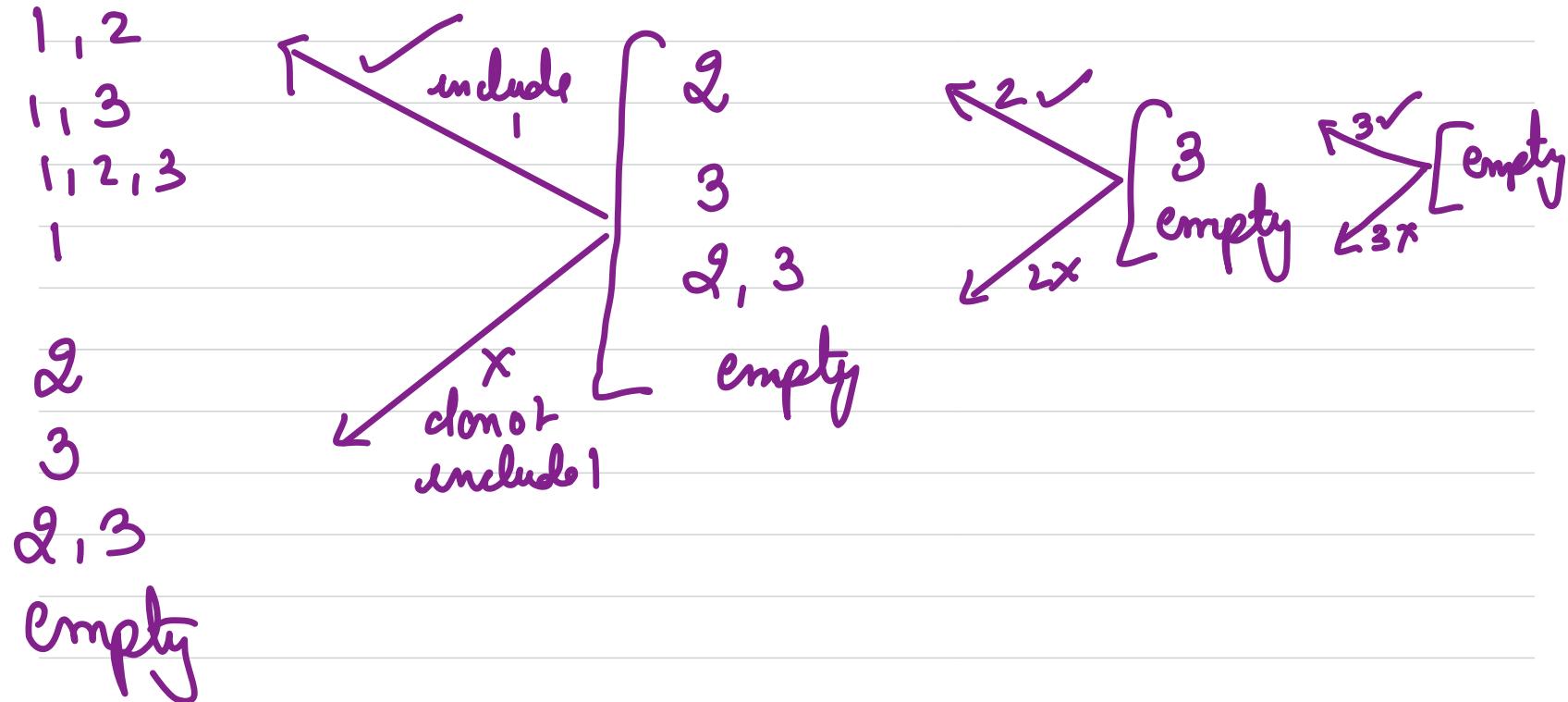
→ we can take the  
choices of  $n^{\text{th}}$   
element.

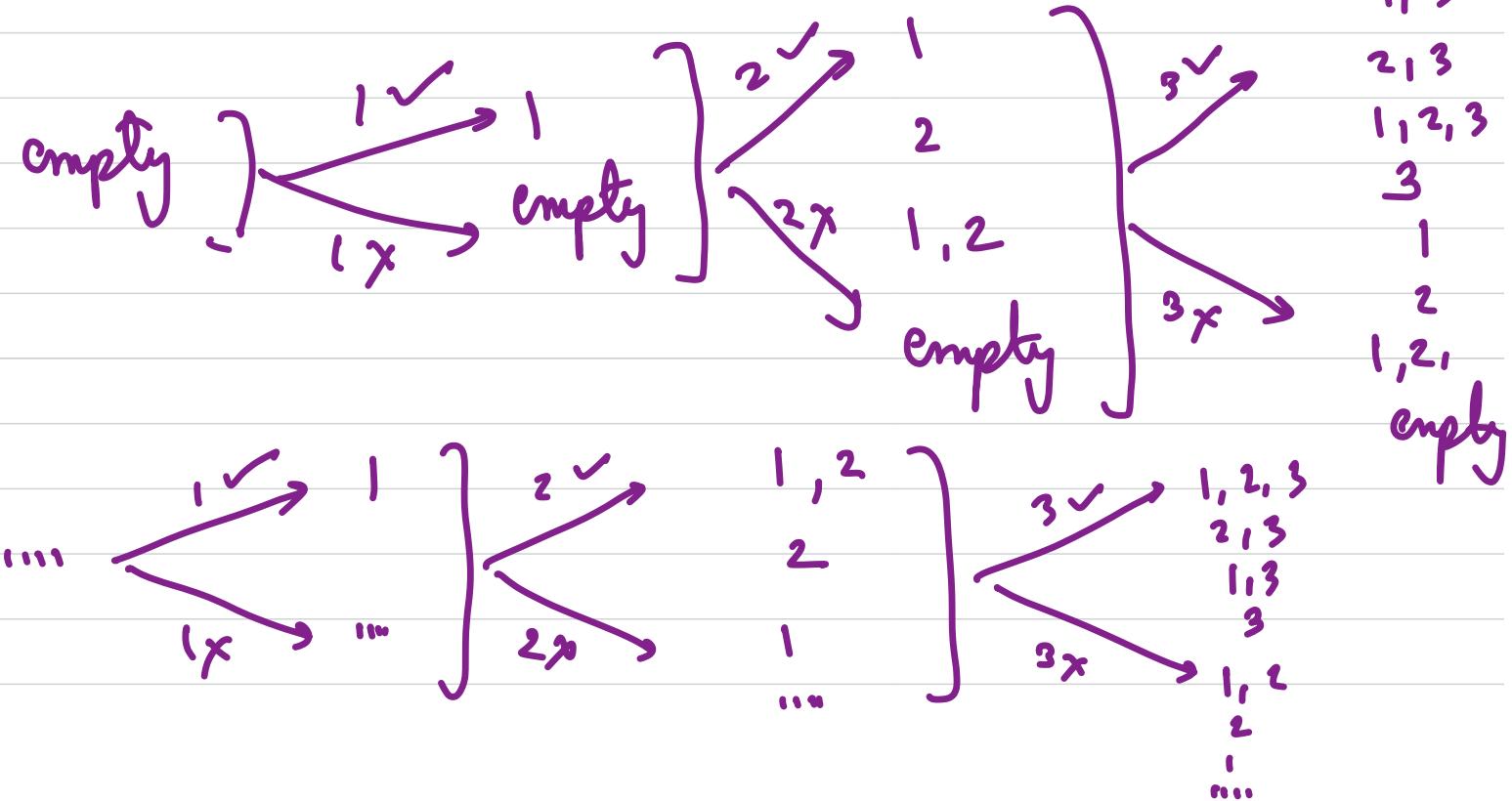
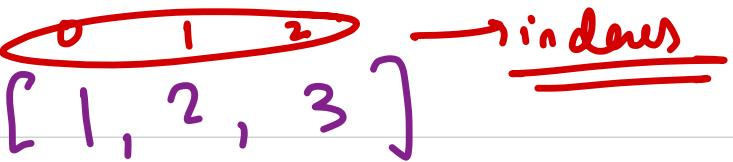


1st client

[ 1, 2, 3 ]

  └┘





$$f(\text{arr}, \text{res}, i) = \begin{cases} f(\text{arr}, \text{res} + \overset{\text{append}}{\text{arr}[0]}, i+1) \\ f(\text{arr}, \text{res}, i+1) \end{cases}$$

two functions

prints all the  
Subsets of  
arr.

Console.log(f([1, 2, 3], ""))

•  
"3"  
"52"

( [S,6,7] , " " , 0 )

( [S,6,7] , "S" , 1 )

( [S,6,7] , " " , 1 )

( [S,6,7] , "S 6" , 2 )

( [S,6,7] , "S" , 2 )

( [S,6,7] , "6" , 2 )

( [S,6,7] , " " , 2 )

$\alpha\in [S, b, ?]$

$(\alpha, "", 0)$

$(\alpha, "S"., 1)$

$1v$

$1x$

$(\alpha, "S 6", 2)$

$(\alpha, "S", 2)$

$2v$

$2x$

$(\alpha, "S 6 7", 3)$

$(\alpha, "S 7", 3)$

$(\alpha, S, 3)$

$(\alpha, 6 7, 3)$

$(\alpha, 6, 3)$

$(\alpha, "", 1)$

$1v$

$1x$

$(\alpha, "6", 2)$

$(\alpha, "", 2)$

$(\alpha, "7", 3)$

$(\alpha, "3", 3)$

Q:- Given an array , check if it is

Sorted or not ? ? (Recursively)

[1, 2, 3] → True

Sorted means arranged  
in ascending order

[5, 9, 2] → false

or not

Try successively



Given a number  $n$ , you can do  $\geq 3$  operations  
on it any number of times

- 1) if  $n$  is divisible by 2, then divide it by 2
- 2) if  $n$  is divisible by 3, then divide it by 3
- 3) if  $n$  is greater than 1, then subtract 1 from it

Calc the minimum no. of operations required to  
reduce  $n$  to 1.

Ex  $n = 9$

ans  $\rightarrow \underline{\underline{2}}$

Say,

$f(n)$

returns the min

no. of ops reqd to

reduce  $n \rightarrow 1$  with

our given ops

$$10 \xrightarrow{-1} 5 \xrightarrow{-1} 4 \xrightarrow{-1} 2 \xrightarrow{-1} 1 \Rightarrow \underline{\underline{4}}$$

$$10 \xrightarrow{-1} 9 \xrightarrow{-3} 3 \xrightarrow{-3} 1$$

$\xrightarrow{-3}$  3

Wrong Ans

Instead of reducing no. with any one op.  
try out all the possibilities.

Say,

$$f(n) =$$

$\downarrow$   
returns the min

$$1 + \min \left\{ \begin{array}{l} f(n/3) \\ f(n/2) \\ f(n-1) \end{array} \right\}$$

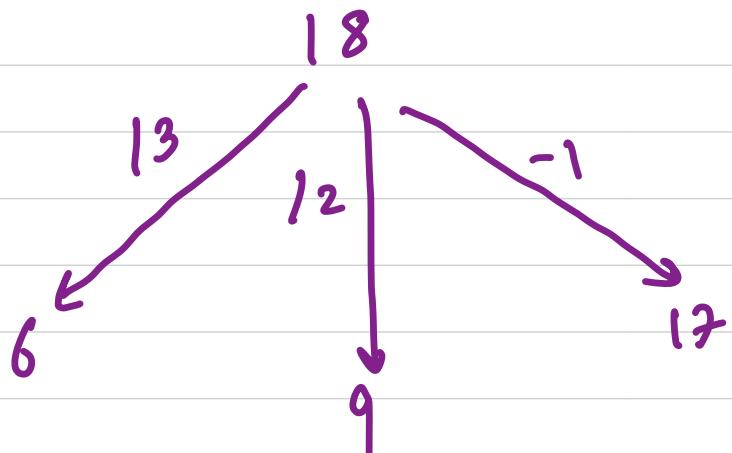
No. of ops reqd to

reduce  $n \rightarrow 1$  with

our given ops

Base Case

$$\begin{array}{lll} n = 1 & \xrightarrow{0} & \} \\ n = 2 & \xrightarrow{1} & \} \\ n = 3 & \xrightarrow{1} & \} \end{array}$$

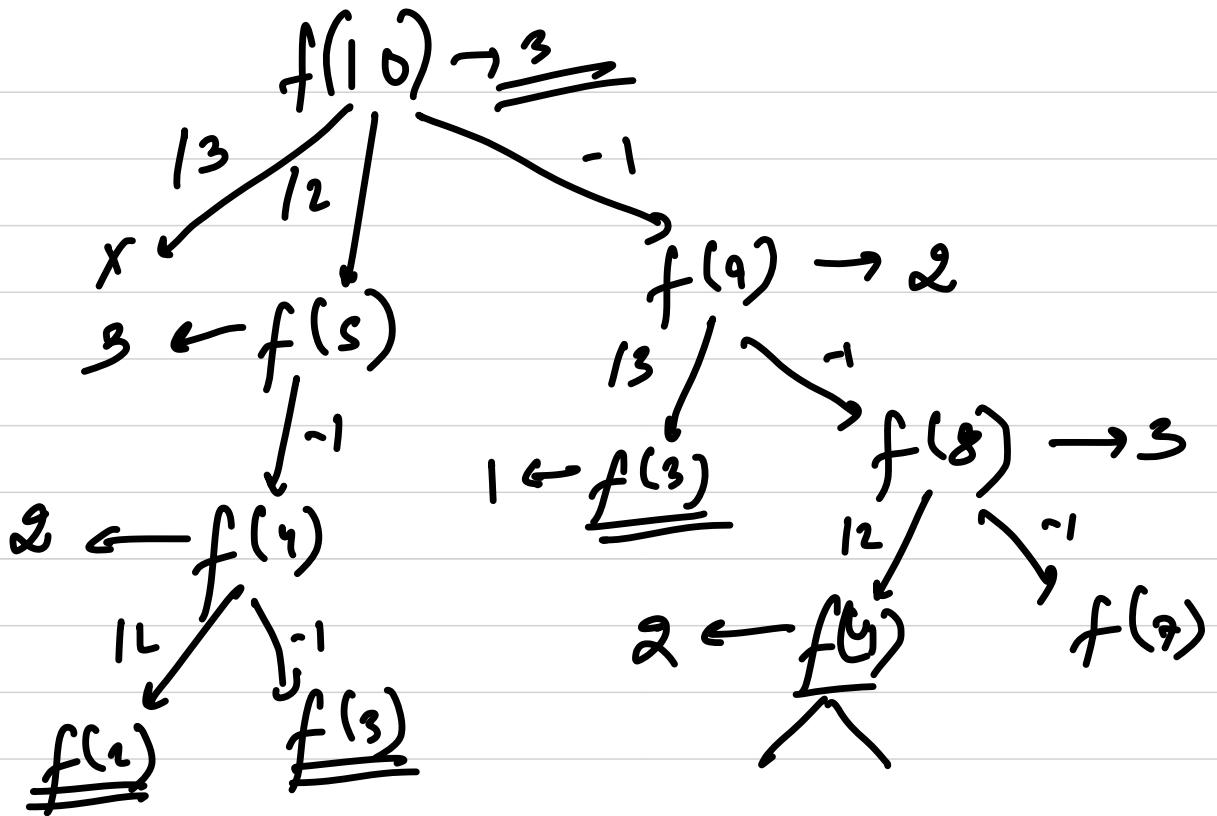


$6 \rightarrow 1 \rightarrow \textcircled{x}$

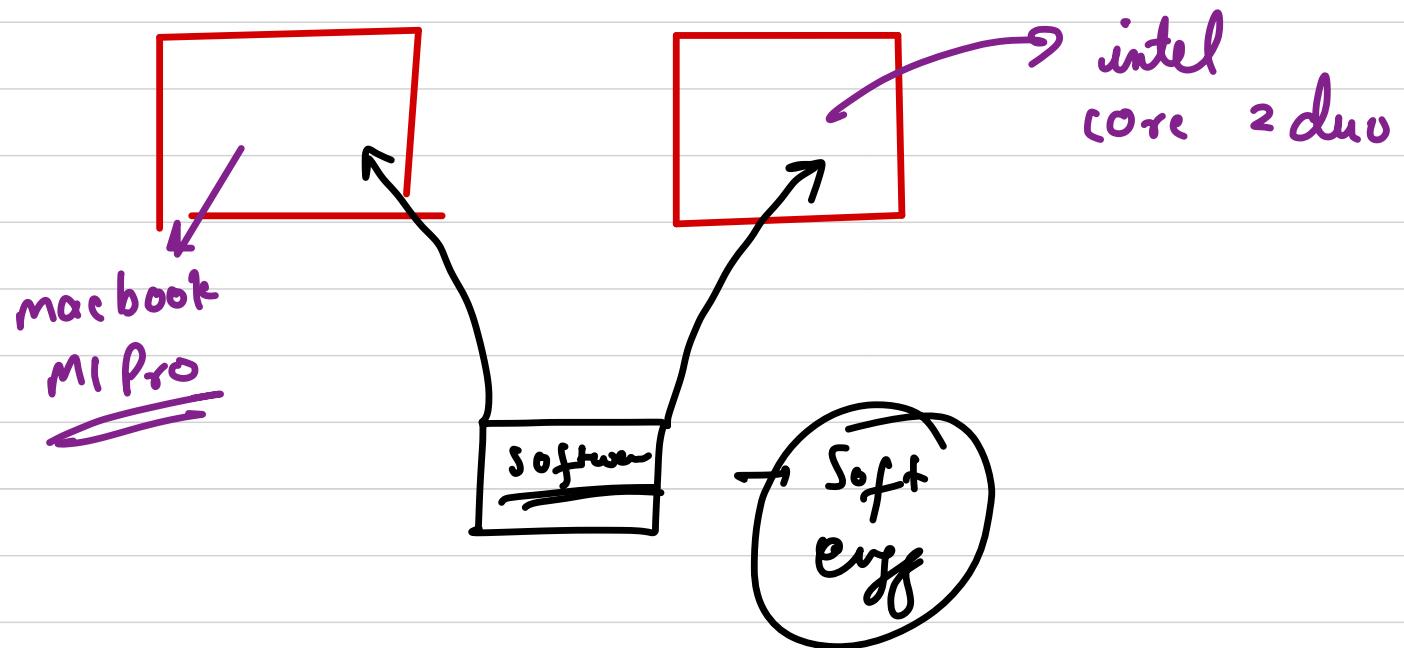
$9 \rightarrow 1 \rightarrow \textcircled{y}$

$17 \rightarrow 1 \rightarrow \textcircled{z}$

$$1 + \min(x, y, z)$$

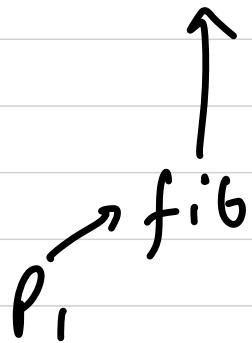


$$\min \left( \underbrace{+\infty}_{\downarrow}, 3, 2 \right)$$

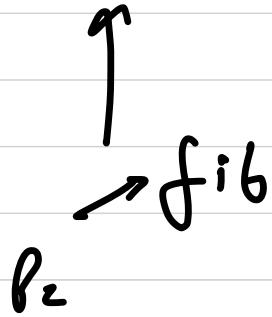


# Experimental analysis

macbook  
m1 pro



i3



In order to compare algorithms we need a mechanism which is machine agnostic

## Asymptotic Analysis

→ input size dependent

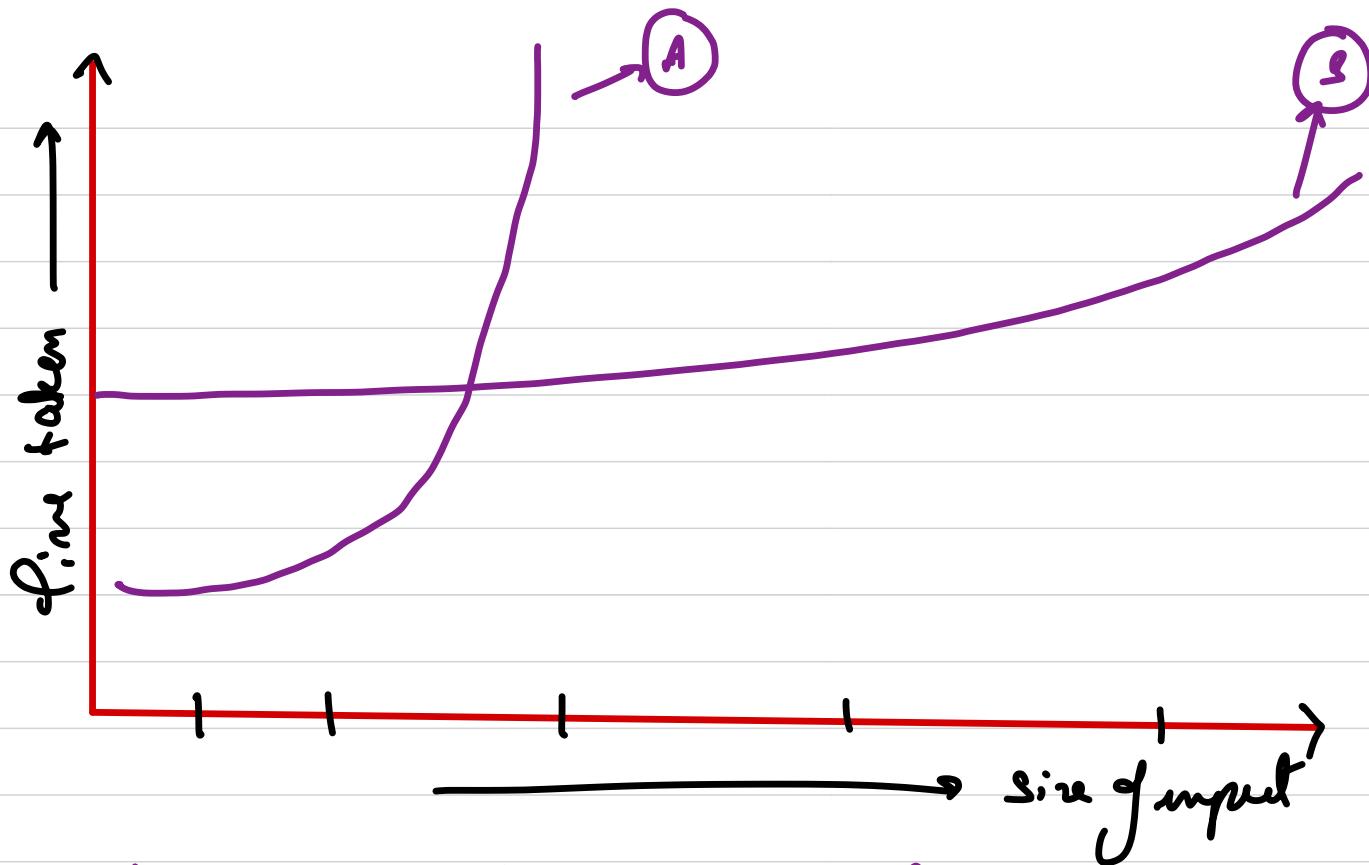
Asymptote

↳ On very big values of your input how the algo behaves is called asymptotic analysis

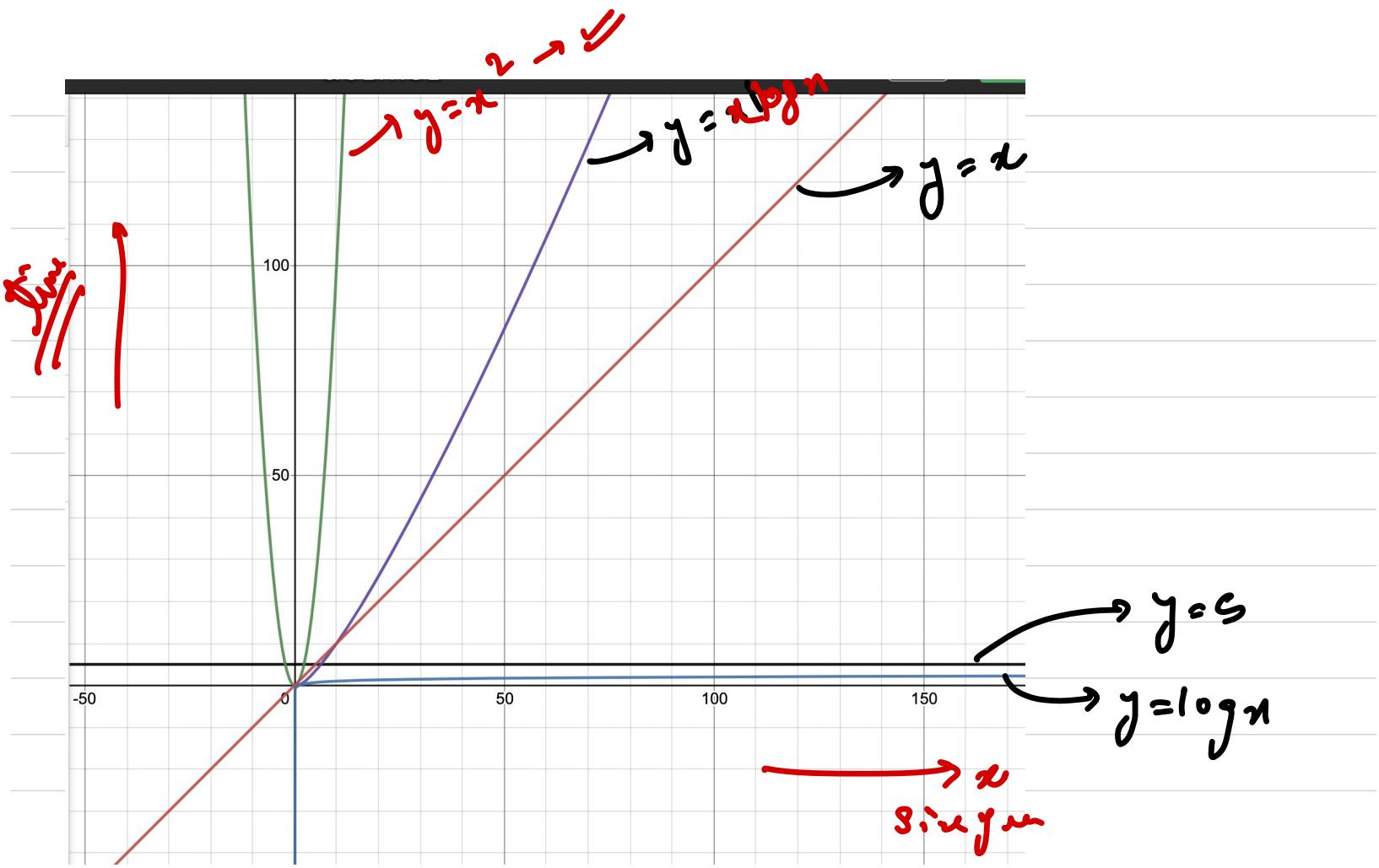
$$2^3 \rightarrow 8$$



$$\cancel{2^{10^6}} \rightarrow \cancel{\text{ }}$$



The rate of growth of B is less



$$\rightarrow y = x^2 \rightarrow \frac{dy}{dx} \rightarrow 2x$$

$$y = x \log x \rightarrow \frac{dy}{dx} \rightarrow 1 + \log x$$

$$y = x \rightarrow \frac{dy}{dx} \rightarrow 1$$

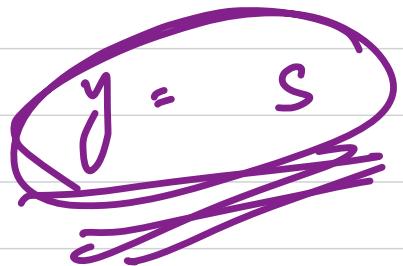
$$y = \log x \rightarrow \frac{dy}{dx} \rightarrow \frac{1}{x}$$

$$y = 5 \rightarrow \frac{dy}{dx} \rightarrow 0$$

$$y = \log_2 x \rightarrow \log_2 10^6$$

$\approx 20$

$$x = 10^6$$



$$x = 10^6$$

Rate of  
change in growth

$x^1$   
 $x^2$   
 $\vdots$   
 $x^3$   
 $x^2$   
 $x\sqrt{x}$   
 $x \log x$   
 $x$   
 $\sqrt{x}$   
 $\log x$   
const

```
for( i=1 ; i <= x ; i++) {  
    console.log(i);  
}
```

2

```
for (i=1; i<=x; i++) {
    for (j=1; j<=x; j++)
        console.log(i, j);
```

$$\underbrace{x+x+x+\dots+x}_{x \text{ times}} = \underline{\underline{x^2}}$$

Asymptotic analysis says that, for very high input value we can avoid lower degree

terms and constants · We will judge rate of growth based on

$$y = \cancel{3x^3} + \cancel{x^2} + \cancel{10x} + \cancel{3}$$

$$\begin{aligned} x &\xrightarrow{\text{---}} 10^9 \\ &\Rightarrow 3\pi(10^9)^3 + (10^9)^2 + 10 \times 10^9 + 3 \end{aligned}$$

# 3 major cases of algo analysis

1) Worst Case

→ Priority hybrid

2) Avg Case

3) Best Case



element =  $x$

$$\underline{n \rightarrow 10^6}$$

Best Case  $\rightarrow \underline{\underline{\Omega(1)}}$

Worst Case  $\rightarrow \underline{\underline{O(n)}}$

Avg Case  $\rightarrow \underline{\underline{O(c \times n)}} \Rightarrow \underline{\underline{O(n)}}$

3 Cases  $\rightarrow$  3 notations

tightest upper bound

1) Worst Case  $\rightarrow$  Big O notation

2) Avg Case  $\rightarrow$  Big O notation  $\xrightarrow{\text{theta}}$  avg time

3) Best Case  $\rightarrow$  Big Omega notation

tightest lower bound

$i = 1$   
 $i = 2$   
 $i = 4$   
8  
16  
32  
 $\vdots$   
 $2^k$

K operator

$$K \approx \log_2 n$$

$$\begin{aligned}2^k &> n \\ \log_2 2^k &> \log_2 n \\ k \log_2 2 &> \log_2 n \\ k &> \log_2 n\end{aligned}$$

$$\log_3^n = \cancel{\log_3^n}$$

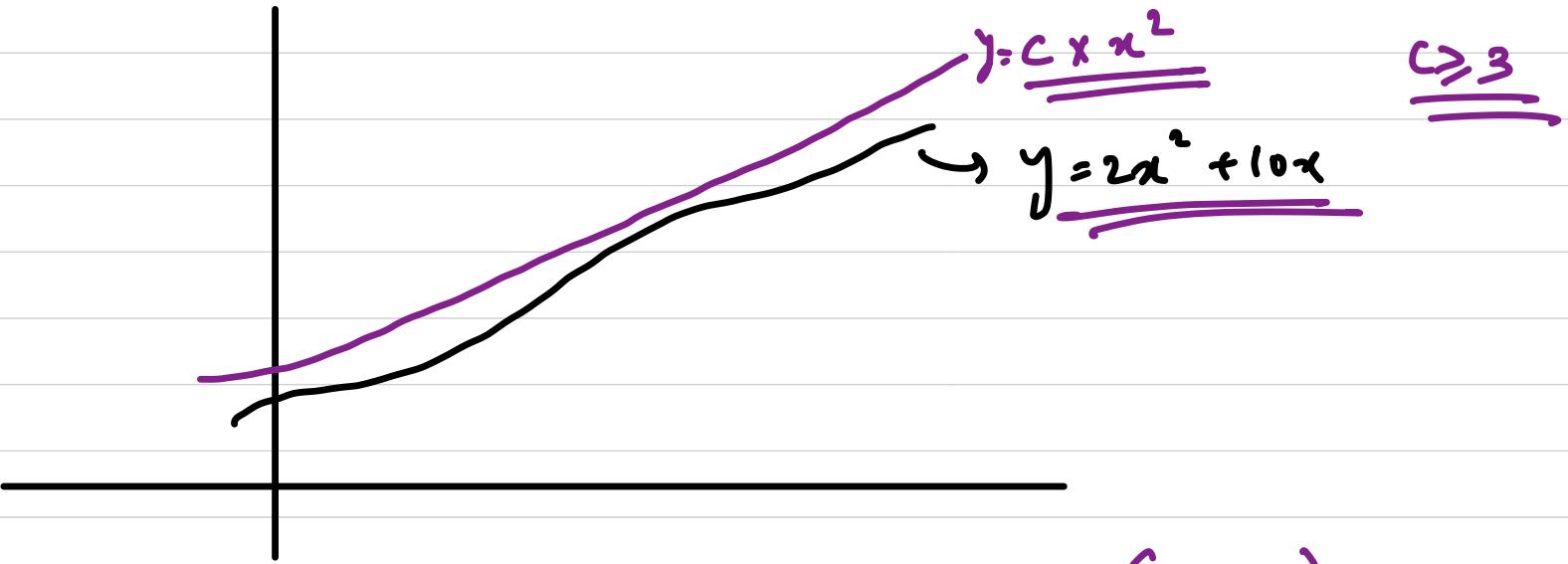
→ last

$$i^* i \leq n$$

$$i^2 \leq n$$

$$i \leq \sqrt{n}$$

for (i=1 ; i  $\leq \sqrt{n}$  ; i++)



$$\mathcal{O}(cx^2)$$

$$\approx \mathcal{\underline{O}}(x^2)$$

# Space Complexity

During the execution of algorithm apart from input and output space , how much space the algorithm took.

function sum(arr)  $\rightarrow$   $\underline{n} = 105$

let ans = 0;

for (let i = 0; i < arr.length; i++) {

ans += arr[i];

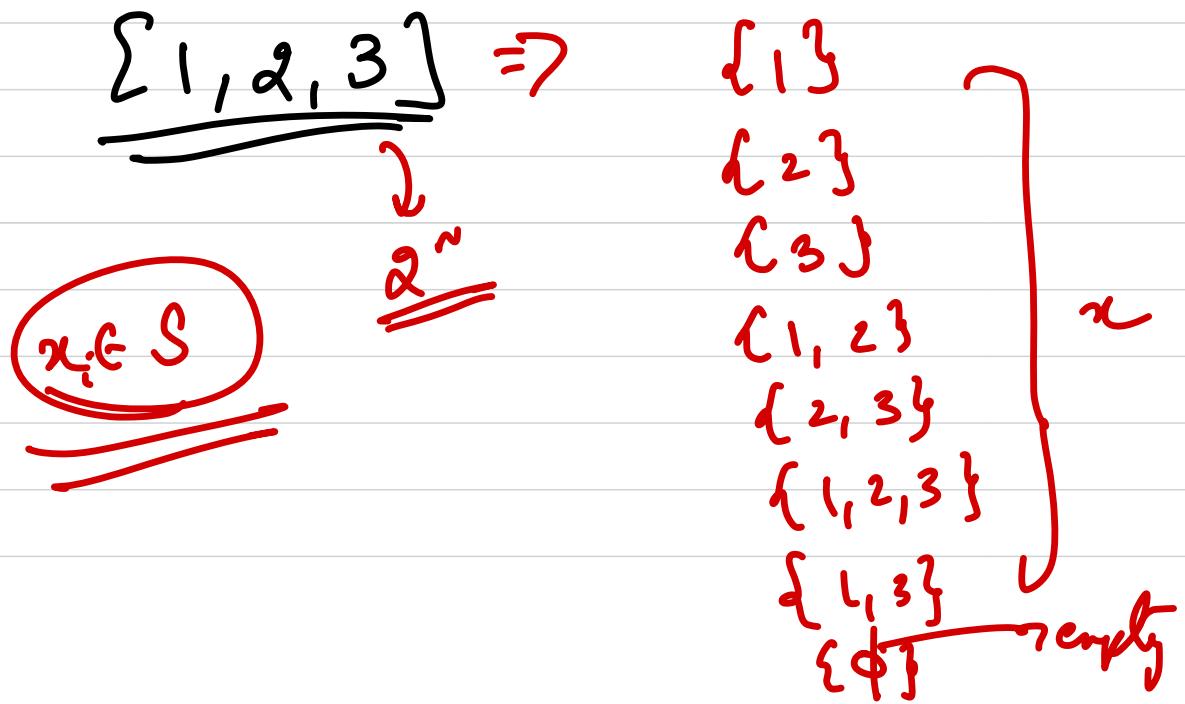
} return<sup>3</sup> ans;

staircase → call stack → will be considered  
in space  
Coupling

## Doubt Session

- \* Dry run of subset
- \* Space complexity
- \* Is array sorted

Subset  $\Rightarrow$  a subset of a set is a collection of elements extracted from the set.



[1, 2, 3]

one choice  $\rightarrow$  included

another choice  $\rightarrow$  excluded

$$2 \times 2 \times 2 \rightarrow \underline{\underline{2^3}}$$

{1, 2, 3}

2^n

✓	✓	✓	$\rightarrow \{1, 2, 3\}$
X	X	X	$\rightarrow \{3\}$
✓	X	X	$\rightarrow \{1\}$
X	✓	X	$\rightarrow \{2\}$
X	X	✓	$\rightarrow \{3\}$
✓	✓	X	$\rightarrow \{1, 2\}$
✓	X	✓	$\rightarrow \{1, 3\}$
X	✓	✓	$\rightarrow \{2, 3\}$

print

index (stry)  
↓  
 $f(\text{arr}, i, \text{output})$

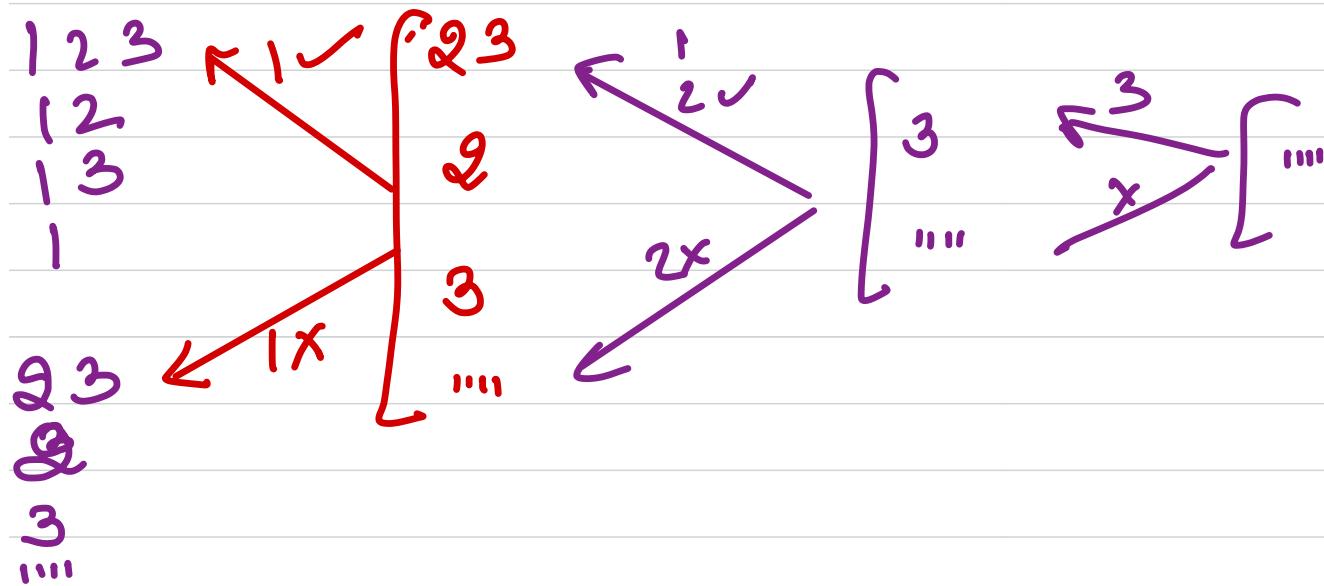
Points the subset  
of arr starting from

Index i

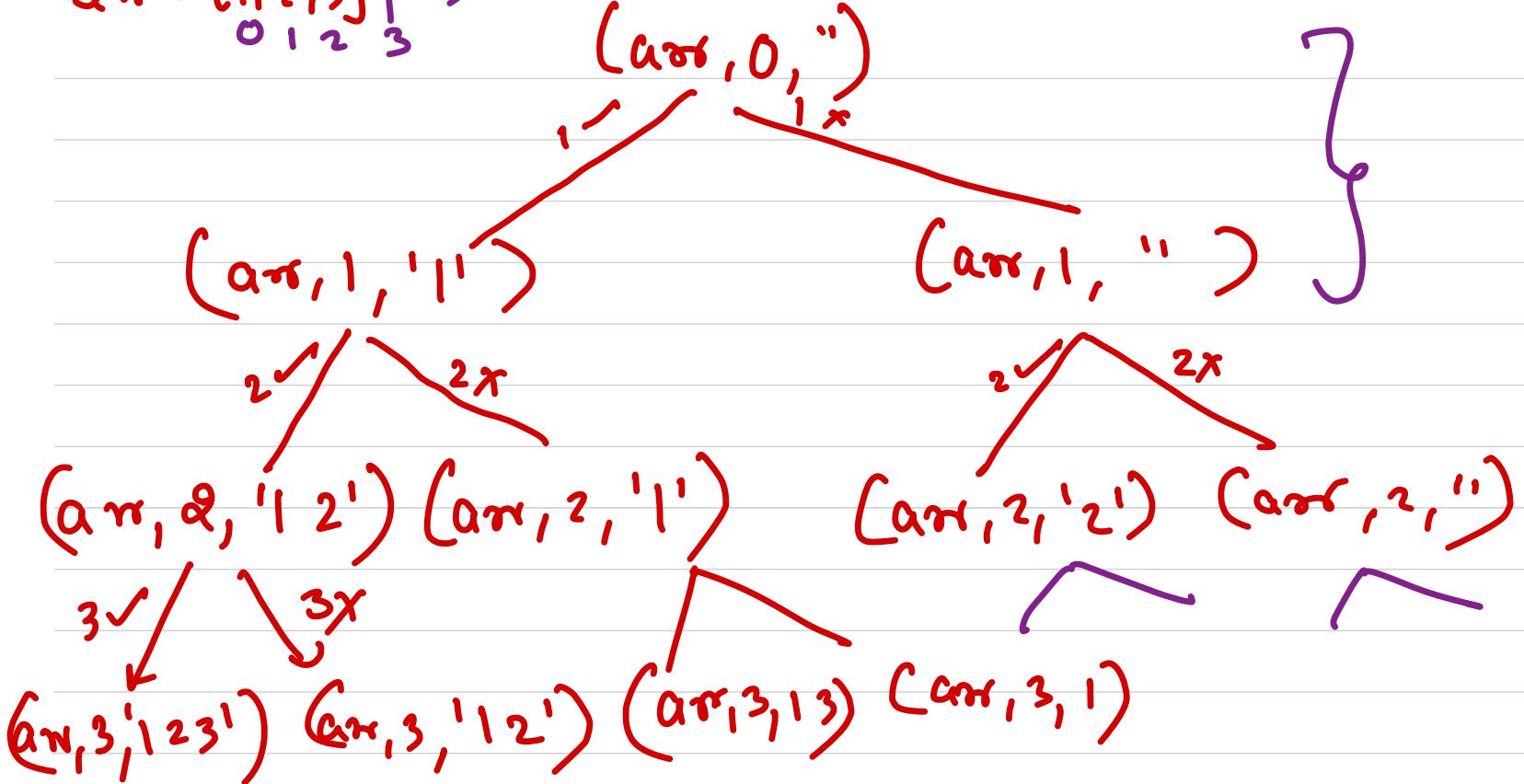
arr

$f(\text{arr}, 0, "") \rightarrow \underline{\underline{\text{arr}}}$

$\{1, 2, 3\}$



$Q \rightarrow [1, 2, 3]$



```
1 function subset(arr, res, i) {  
2     if(i == arr.length) {  
3         console.log(res);  
4         return;  
5     }  
6     subset(arr, res + " " + arr[i], i+1);  
7     subset(arr, res, i+1);  
8 }  
9  
10 subset([4, 9], "", 0);
```

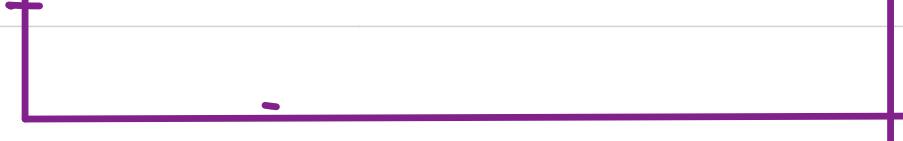
| 4 9 |

| 4 |

| 9 |

| |

Call stack



Q → Recursively → array is asc  
Check if array is sorted or not.

[1, 2, 3] → True

[3, 2, 1] → False

f(arr, idx)  
funcn return whether  
array from index idx  
is sorted or not

```
1 function isSorted(arr, idx) {  
2     if(idx == arr.length - 1) {  
3         return true;  
4     }  
5     let assume = isSorted(arr, idx+1);  
6     return assume && (arr[idx] <= arr[idx+1]);  
7 }  
8  
9 console.log(isSorted([1,12,3,4,5], 0));
```

$\text{arr} \rightarrow [1, 4, 2, 3]$        $10^3$       index  
 $0 \quad 1 \quad 2 \quad 3$

$\text{arr}[1] \rightarrow 4$

$\text{Space} \rightarrow O(n)$

false

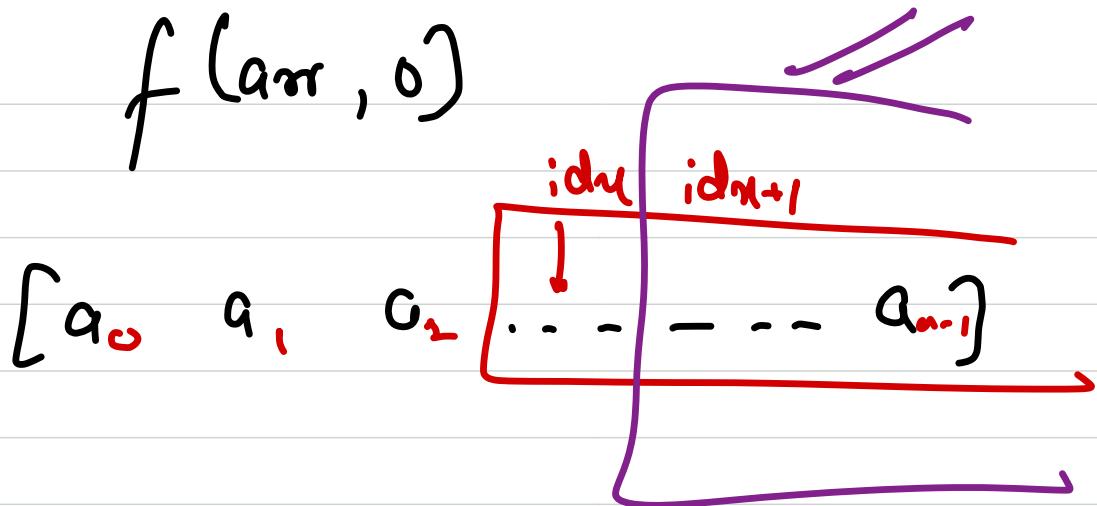
isSorted  
 $\text{arr} \rightarrow [1, 4, 2, 3]$   
 $\text{idx} = 0$   
assume = false

lines

↑ false

ans →

$f(\text{arr}, 0)$



Base Case → if there is only single element → it  
is always sorted.

$$f(\text{arr}, 0) = f(\text{arr}, 1) \text{ and } (\text{arr}[0] \leq \text{arr}[1])$$

$$f(\text{arr}, ; \text{idx}) = f(\text{arr}, \text{idx}+1) \text{ and } (\text{arr}[\text{idx}] \leq \text{arr}[\text{idx}+1])$$

assume this  
works  
correctly

and →

X  
F  
T  
F  
T

Y  
F  
F  
T  
T

X and Y  
F  
F  
F  
T

Y

1, 2, 3, 4, 5

```
function f5(n) {  
    for(let i = n; i > 0; i--) {  
        for(j = 0; j < i; j++) {  
            console.log(i, j);  
        }  
    }  
}
```

Total ops

$$\rightarrow n + \frac{n}{2} + \frac{n}{4} \dots$$

$i = n \rightarrow \text{ops} \rightarrow n$

$i = \frac{n}{2} \rightarrow \text{ops} \rightarrow \frac{n}{2}$

$i = \frac{n}{4} \rightarrow \text{ops} \rightarrow \frac{1}{4}$

$$n \left( -\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots - \frac{1}{n} \right) \Rightarrow \cancel{\alpha_1}$$

What is the sum of the series  $\Rightarrow 2$

geometric  
progression

$$S = \frac{a \times (1 - \gamma^{n+1})}{1 - \gamma} \rightarrow \frac{\frac{a}{\gamma}}{1 - \gamma}$$

$$\frac{1}{1-\gamma_2} \rightarrow \left(\frac{1}{2-\gamma}\right)_{\gamma_2} \stackrel{\text{def}}{=} q$$

```
function f6(n) {  
    let ans = 0;  
    while(n > 0) {  
        ans += n;  
        n /= 2;  
    }  
    return ans;  
}
```

no. of iterations

```
function f5(n) {  
    for(let i = n; i > 0; i/=2) {  
        for(j = 0; j < i; j++) {  
            console.log(i, j);  
        }  
    }  
}
```

$$1 + \frac{1}{2} + \frac{1}{4} + \dots \rightarrow O(n)$$

no. of iterations  $\times$  d = no. of ops

$$2 + 2 + 2 + 2 + 2 + \dots \xrightarrow{k \text{ times}} 2^k$$

```

function f6(n) {
    let ans = 0;
    while(n > 0) {
        ans += n;
        n /= 2;
    }
    return ans;
}

```

$2 + 2 + 2 + 2 \dots \dots \dots 2$   
 $\underbrace{2}_{\text{2}} \quad \underbrace{\dots}_{\text{k times}}$

last value of  $n$   
 $n \geq 1$

$K \rightarrow ?$   
 no. of iteration

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{2^2} \rightarrow \frac{n}{2^3} \rightarrow \frac{n}{2^4} \dots \dots \dots \rightarrow \frac{n}{2^K}$

$$\boxed{1 \leq \frac{n}{2^K} \leq 1}$$

total no. of ops =  $2^K$   
 last time

$$n \geq 2^k$$

Taking log both sides

$$\log_2 n \geq \log_2 2^k$$

$$\log_2 n \geq k \log_2 2$$

$$\hookrightarrow \boxed{k \leq \log_2 n}$$

Total ops  $\rightarrow 2^k \rightarrow 2 \log_2 n$   
 $\rightarrow \underline{\underline{O(\log n)}}$

```
function f7(n) {  
    for(let j = 1; j <= n; j++) {  
        ↗ for(let i = 0; i < n; i = i + j) {  
            console.log(i, j);  
        }  
    }  
}
```

*Total ops*

$j=1 \quad i=0 \rightsquigarrow 1 \rightsquigarrow 2 \rightsquigarrow 3 \dots \dots \dots n-1 \longrightarrow n$

$j=2 \quad i=0 \rightsquigarrow 2 \rightsquigarrow 4 \rightsquigarrow 6 \dots \dots \dots n-1 \longrightarrow n/2$

$j=3 \quad i=0 \rightsquigarrow 3 \rightsquigarrow 6 \rightsquigarrow 9 \dots \dots \dots n-1 \longrightarrow n/3$

$j=4 \quad i=0 \rightsquigarrow 4 \rightsquigarrow 8 \rightsquigarrow 12 \dots \dots \dots n-1 \longrightarrow n/4$

$\vdots$

$j=n \quad i=0 \rightsquigarrow n \longrightarrow 1$

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots \dots \dots |$$

$$n \left( 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots \dots \dots \frac{1}{n} \right)$$

Sum of the series ??

$$\log n$$

$$O(n \log n)$$

$$\sum_{k=1}^n \frac{1}{k}$$

$$\Rightarrow \int_1^n \frac{1}{k} dk = [\log k]_1^n \\ \log n - \log 1 \\ \rightarrow \log n$$

ans

```
function f8(n) {  
    let ans = 0;  
    for(let i = 1; i <= n; i*=2) {  
        ans += 1;  
    }  
    return ans;  
}
```

iterations

loop

Total iterations  $\times$  1 = Total ops

$$j = 2^0$$

$$j = 2^1$$

$$j = 2^2$$

$$j = 2^3$$

$$j = 2^4$$

⋮

$$j = 2^k$$

$$2^k \leq n$$

Takey log both sides

$$\log_2 2^k \leq \log_2^n$$

$$k \leq \log_2^n$$

$$k \leq \log_2^n$$

Time complexity  $\Rightarrow \underline{\underline{O(\log n)}}$

Time = # of operations in  
one function  
call  $\times$  # of function calls =

$\text{fact}(n) \rightarrow \text{fact}(n-1) \rightarrow \text{fact}(n-2) \rightarrow \text{fact}(n-3) \dots$   
 $\dots \rightarrow \text{fact}(0)$

$\text{fact}(5) \rightarrow \text{fact}(4) \rightarrow \text{fact}(3) \rightarrow \text{fact}(2) \rightarrow \text{fact}(1) \rightarrow \text{fact}(0)$

Time  $\rightarrow O(1) \times n$

$\rightarrow \underline{\underline{O(n)}}$

Prob → # of ops in one func call → constant

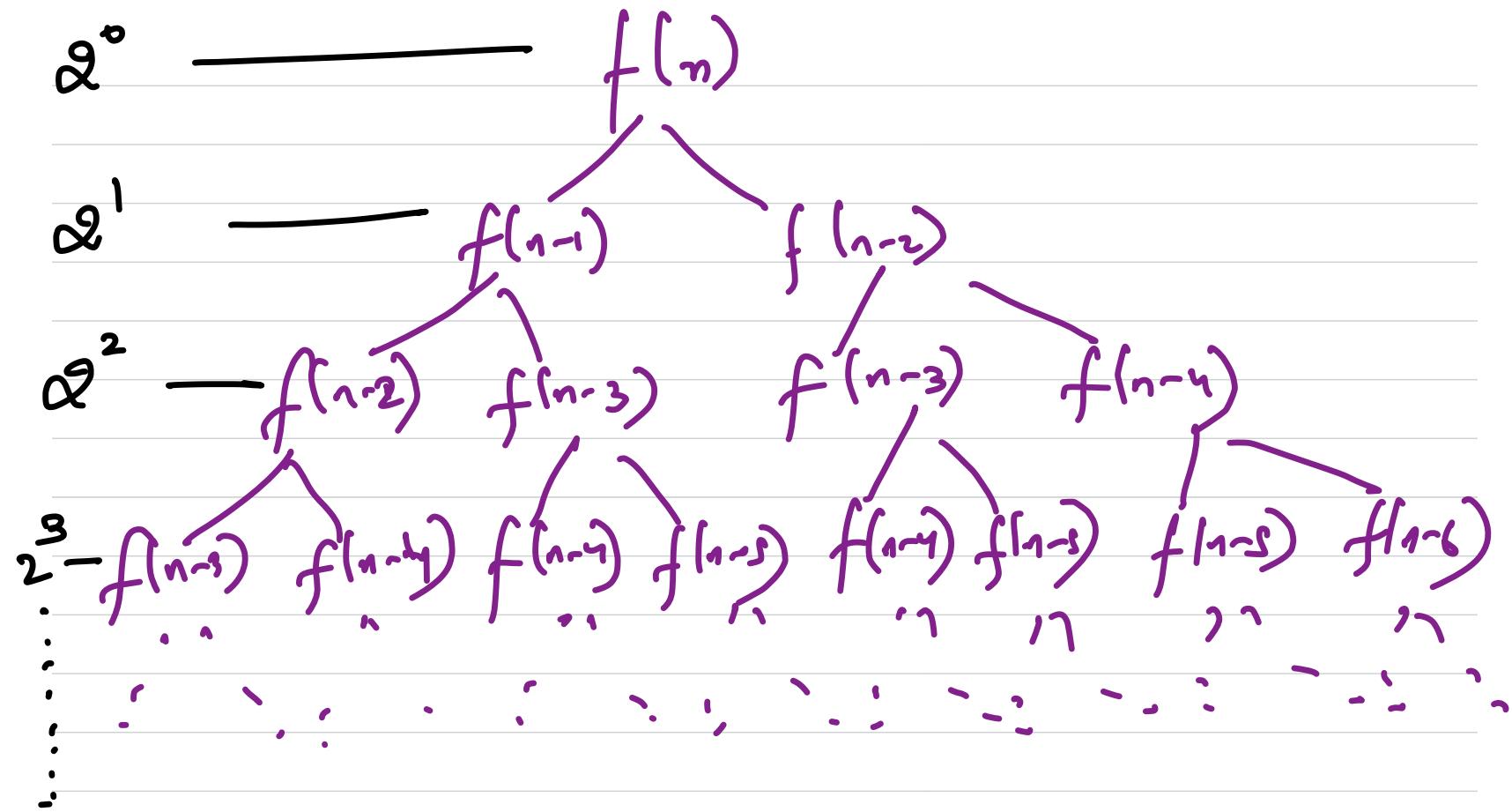
how many function calls:

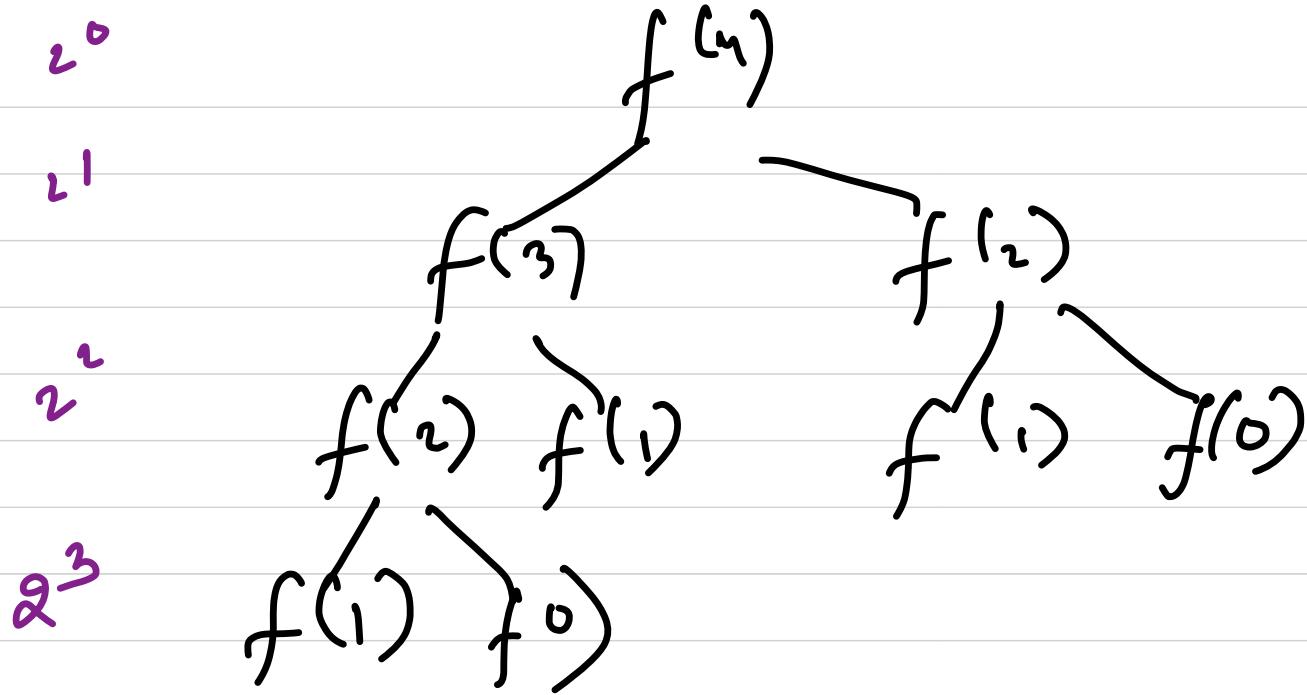
$$2^0 + 2^1 + 2^2 + \dots + 2^n$$

$$\Rightarrow S = \frac{2^0(2^{n+1} - 1)}{2 - 1}$$

$$S \rightarrow \underline{2^{n+1} - 1}$$

$$\text{Time} \rightarrow O(1) \times (2^{n+1} - 1) \Rightarrow \underline{O(2^n)}$$



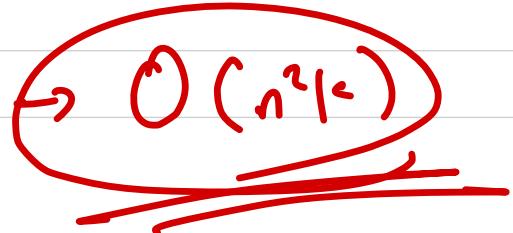


$$1 \text{ func call} \rightarrow 1 + \underbrace{1 + nk + 1 + 1}_{\text{brace}} \rightarrow \underline{\underline{\Theta(nk)}}$$

How many times the func is called ??  $\underline{\underline{n}}$

$$f(n) \rightarrow f(n-1) \rightarrow f(n-2) \dots f(0)$$

$$\text{func} \rightarrow n \times \Theta(nk)$$



In JS ~~strings~~ are immutable

$s = "abc"$

*new  
stru  
geek* ←  $s + "d"$

$"abc" + "d" \rightarrow "abcd"$

$S = "a"$

$S + "b"$

$"ab"$

$S + "d"$

$S = "abc-----" \text{ in char}$

$S + d$

$(abc-----d)$

$O(n)$

$S += "def"$



$S = S + "def"$

$S = "...def"$

# # Dice Combinations

$$f(n) \rightarrow f(n-1) + f(n-2) + f(n-3) + f(n-4) + f(n-5) + f(n-6)$$

return the no. of  
ways to divide  $n$   
with dice nos

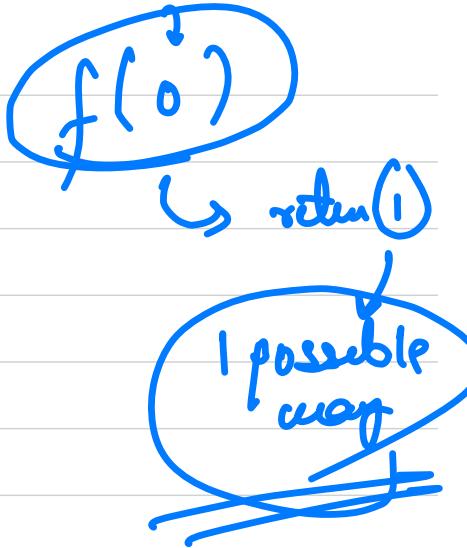
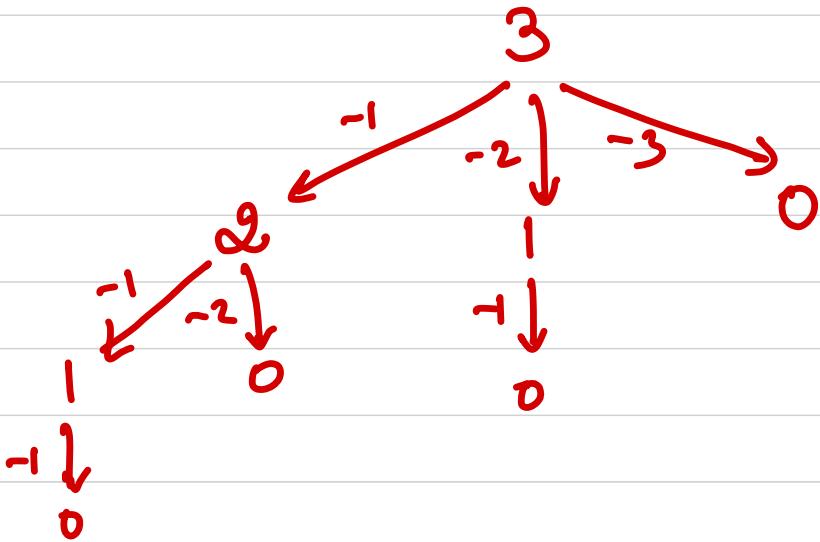
if ( $n == 1$ )  $\rightarrow$  return 1; X Better base case

if ( $n == 0$ )  $\rightarrow$  return 1;

$n \rightarrow 0$

$f(n)$   $\rightarrow$   $f(0)$

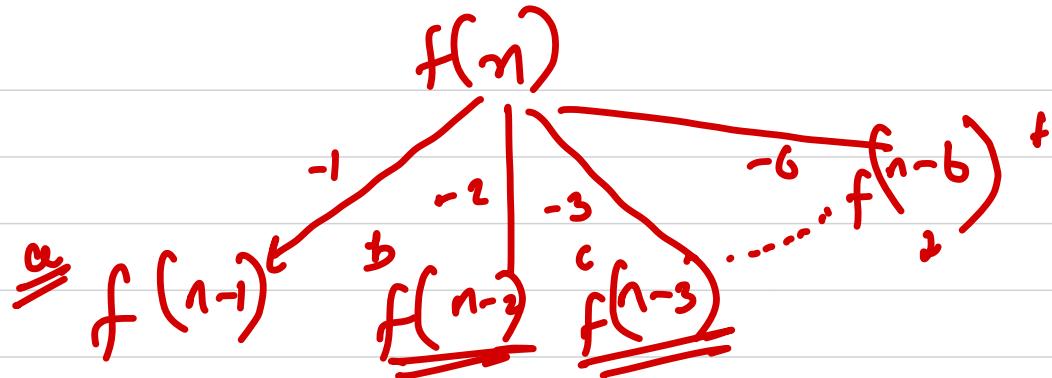
1, 2, 3, 4, 5, 6



Whenever we reach zero by reducing  $n$ , we say we found one possible way.

one possible way

*General Case*



$$\cong f(n-1)$$

$f(n)$

-1

-2

-3

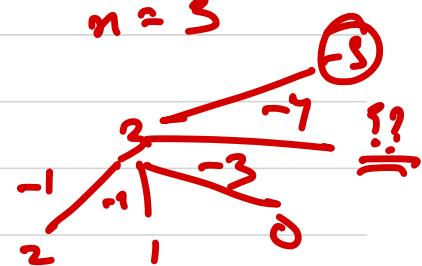
c

-6

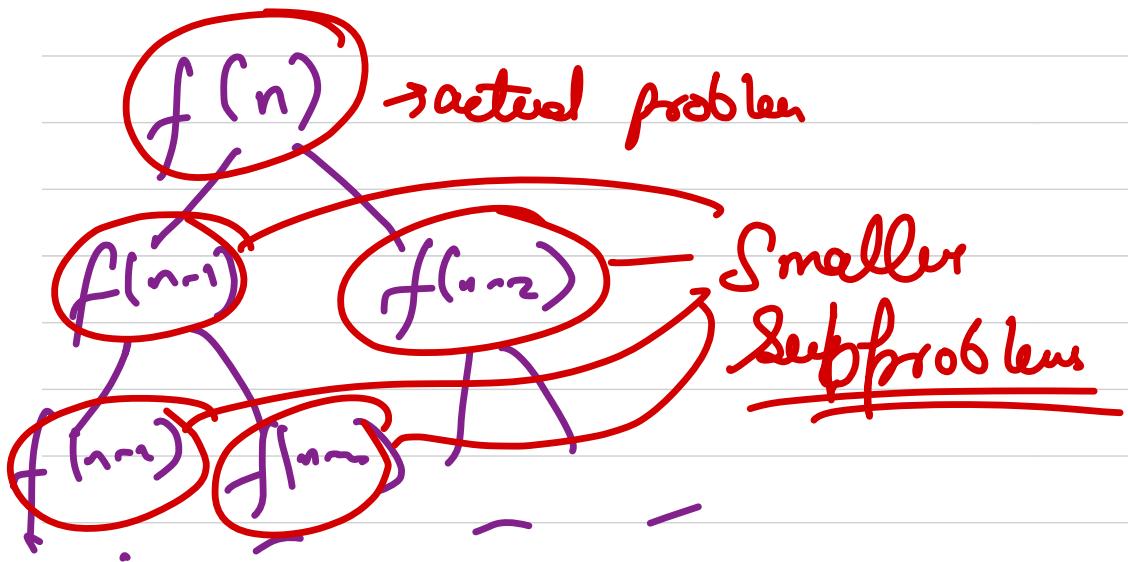
$f(n-b)$

+

$n = 5$



\* State of a problem → It is a subproblem only

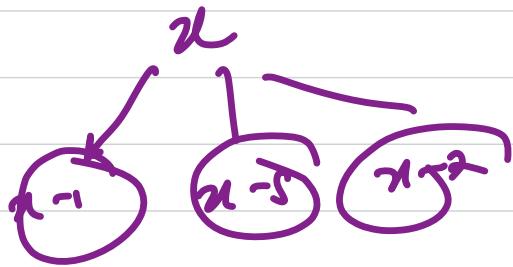


$f(x, \text{coins})$

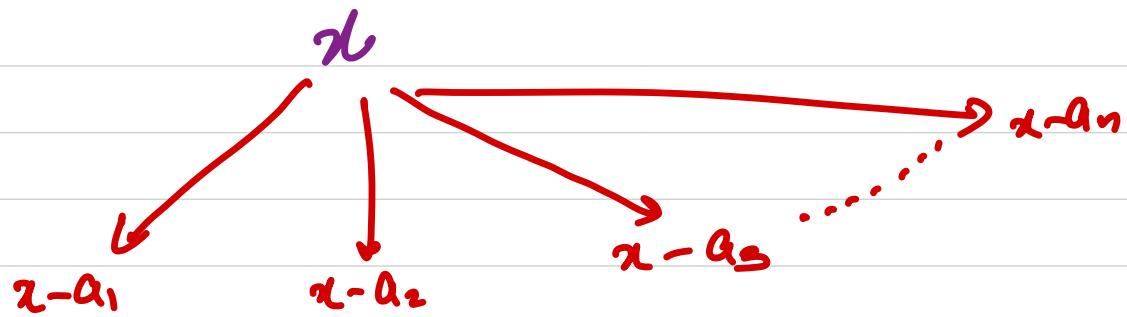
coins any

returns min no. of  
coins to make  
a value  $x$

(1, 5, 7)



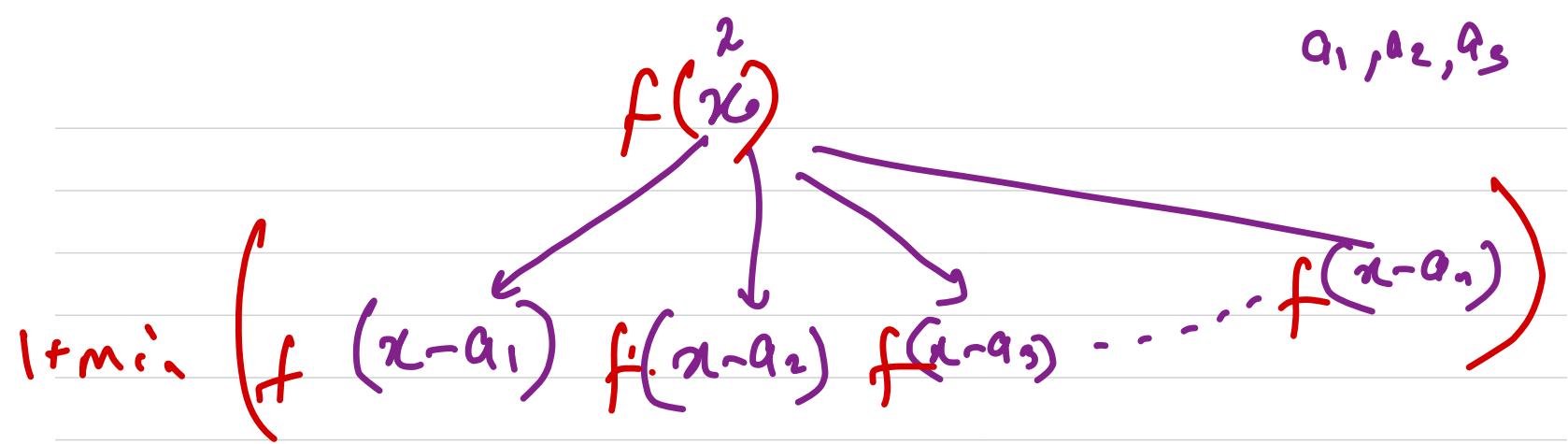
coins  $\rightarrow \underline{\{a_1, a_2, a_3, \dots, a_n\}}$



$$\underline{\underline{f(x, \text{coins})}} = 1 + \min(f(x - \text{coins}[0]), f(x - \text{coins}[1]), \\ f(x - \text{coins}[2]) \dots \dots \dots)$$

when the min no of  
coins to make x



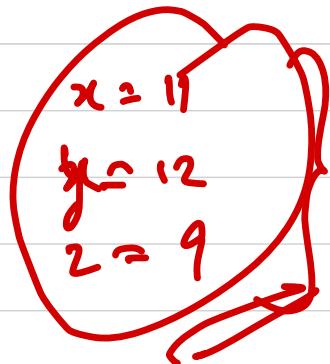


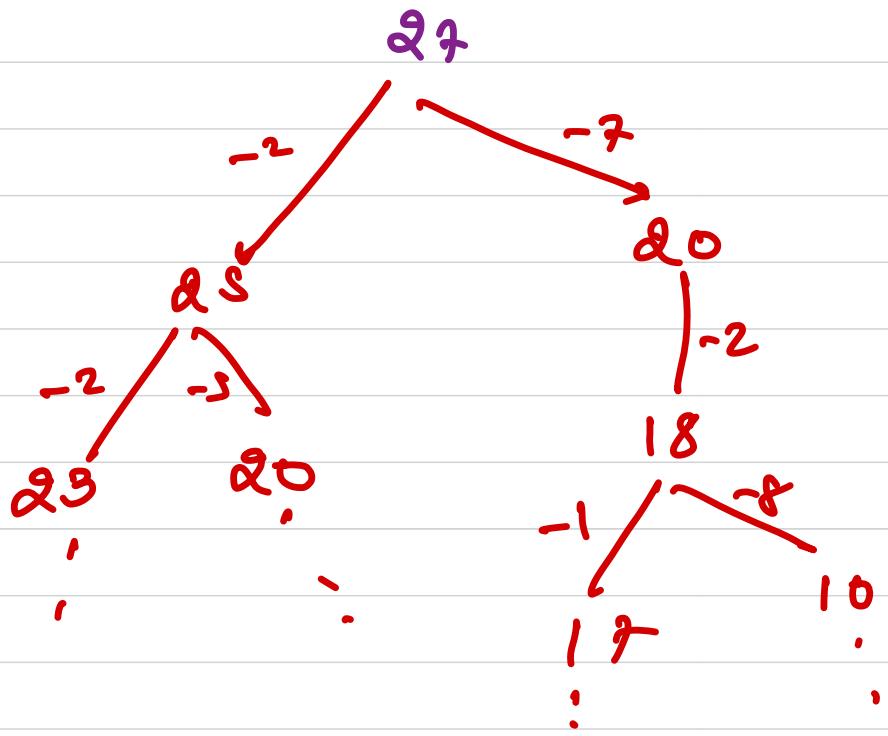
$$ans = \frac{x+5}{2} \text{ if } 9$$

$$ans = \min(ans, x)$$

$$ans = \min(ans, y)$$

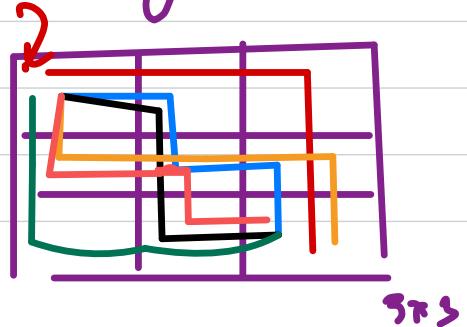
$$ans = \min(ans, z)$$





~~Qn~~ Given a grid of  $n \times m$  dimensions - You're standing at  $0,0$  (i.e. top left corner of grid). From any cell of the grid you can either go right or go downwards. Find the total no. of ways to reach the bottom-right corner of the grid from the top-left corner.

Corner:



Ans  $\rightarrow 6$

$$f(i, j) = f(j, j+1) + f(i+1, j)$$

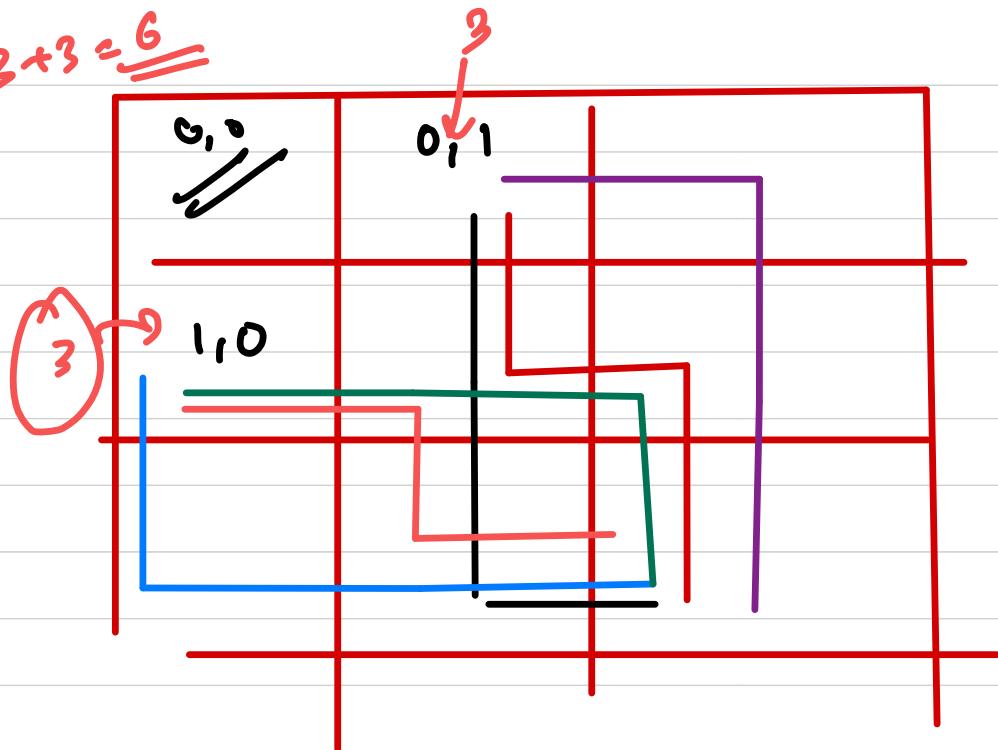
value the  
no. of ways  
to reach bottom  
right from  
 $i, j$

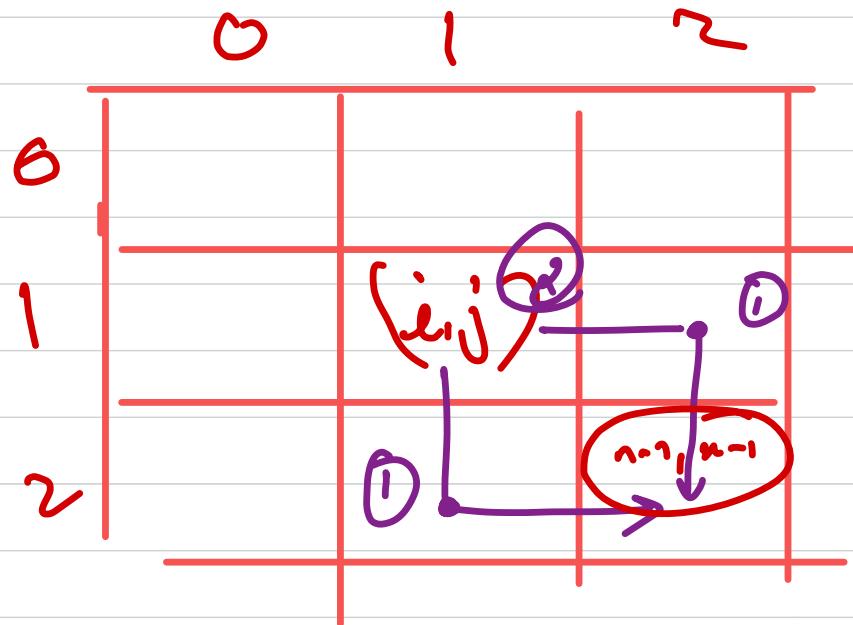
no. of ways  
to reach bottom  
right from  
rightward  
cell to  $i, j$

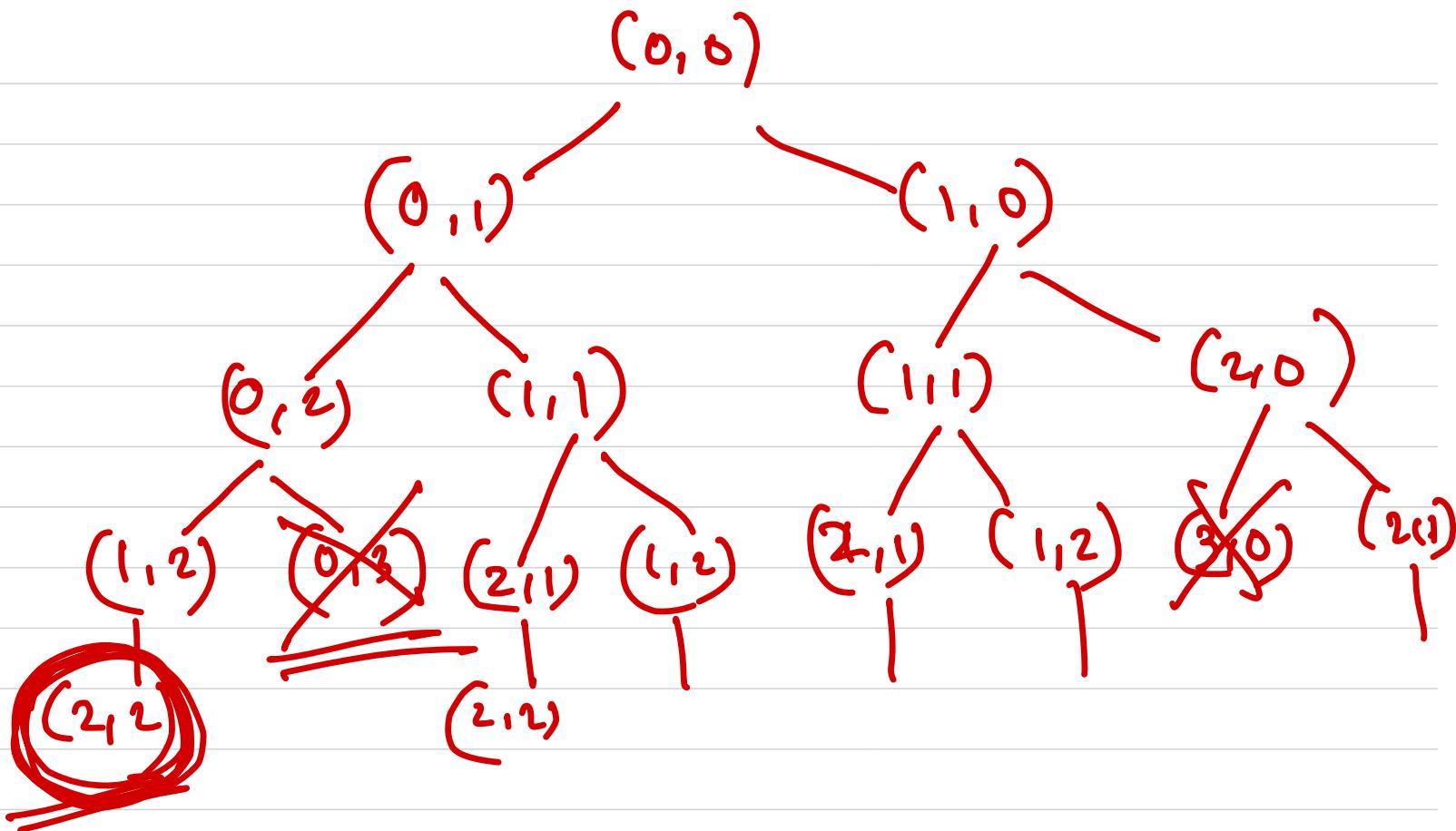
no. of ways  
to reach bottom  
right from  
downward  
cell to  $i, j$

ans →  $f(0, 0)$

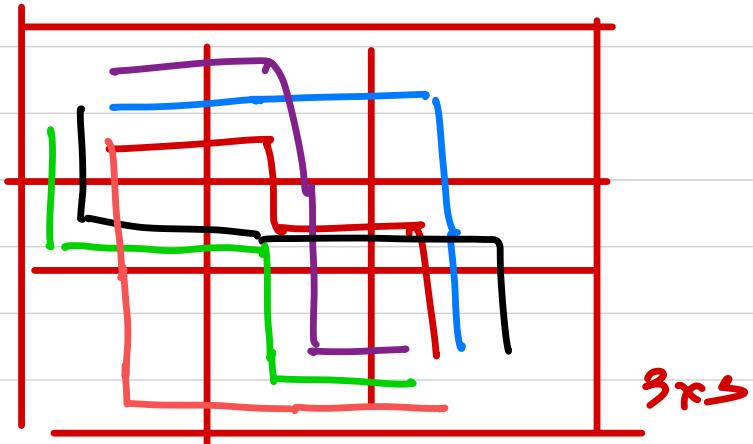
$$3 \times 3 = 9$$







Instead of just count, can we also print the path?



R R D D

R O R D

R O O R

D R R D

D R D R

D D R R

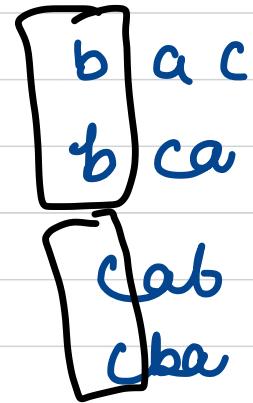
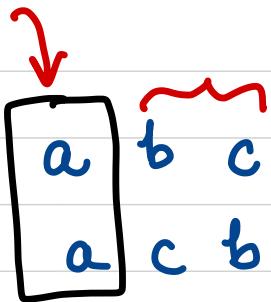
~~Q.~~ Given an array of characters, print all permutations of the array (recursively)

Ex `['a', 'b', 'c']`

abc  
acb  
bac  
bca  
cab  
cba

all permutations

↳ [a, b, c]



[a, b, c, d]

a b c d  
a b d c  
a d b c  
a d c b  
a c b d  
a c d b

b a c d  
b a d c  
b d a c  
b d c a  
b c a d  
b c d a

c →

d →

$f(\text{arr}, \text{idx}, \text{output})$

This function returns  
permutation of the  
elements starting from  
index idx.

answer →  $f(\text{arr}, \underline{\underline{0}}, \underline{\underline{\text{"}}})$

This will  
not  
work

[a, b, c]

(arr, 0, '')

(arr, 1, 'a')

(arr, 1, 'b')

a✓

b✓

c✓

Every char is gonna get a chance to  
become the first char.



$f(\text{arr}, \text{id}_x)$



return all permute

startly from index  $\text{id}_x$

$\text{arr} \rightarrow [a, b, c]$

$([[c, b, a], 0])$

$([a, c, b], 1)$

$([c, b, a], 1)$

$([a, b, \underline{c}], 2)$

$([a, c, b], 2)$

$([\underline{c}, a, b], 2)$      $([\underline{c}, b, a], 2)$

$a \checkmark$

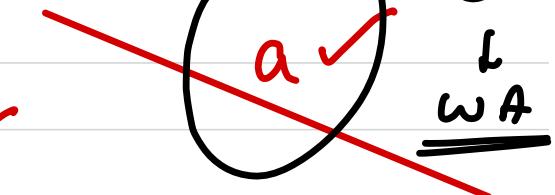
$c \checkmark$

$b \checkmark$

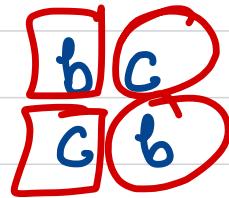
$c \checkmark$

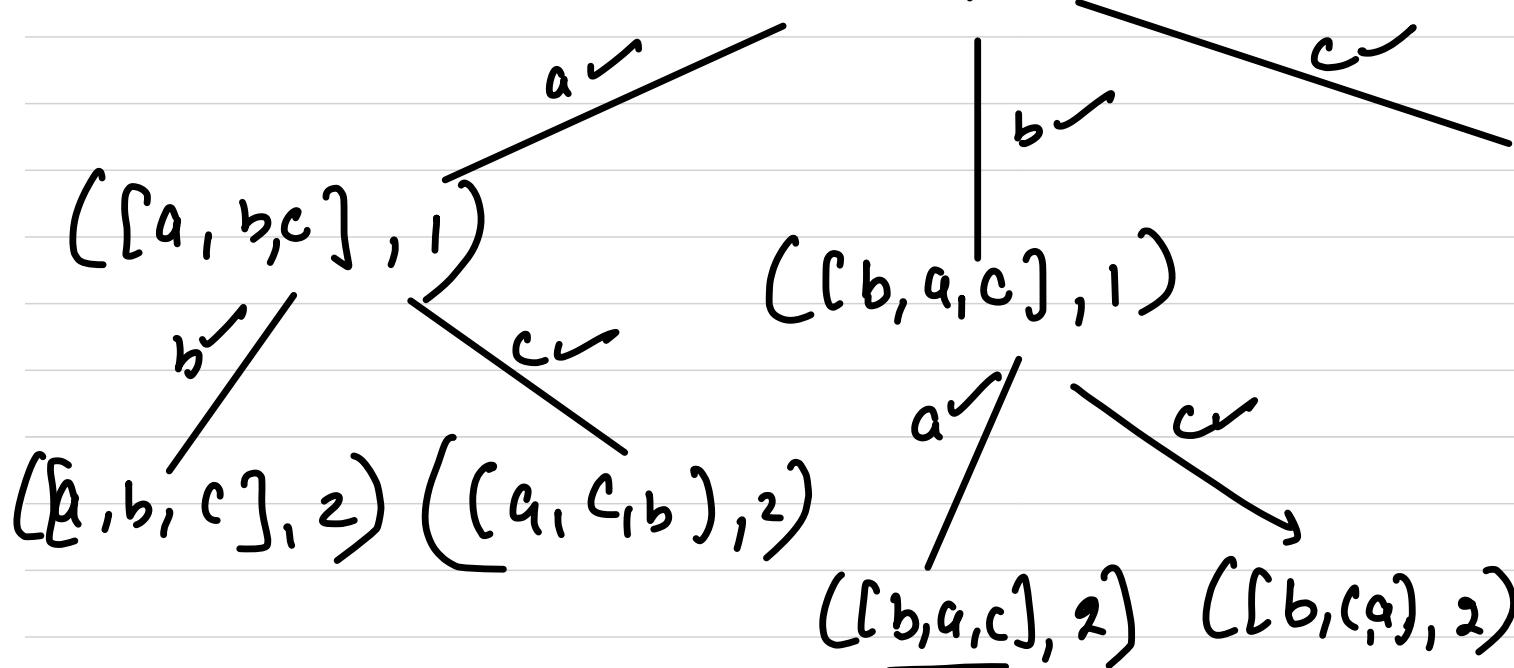
$a \checkmark$

$b \checkmark$



$b\bar{c} \rightarrow$



$([a,b,c], \circ)$ 

abc, acb, bac, bca

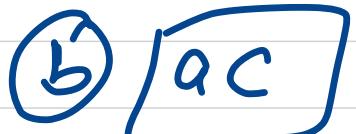
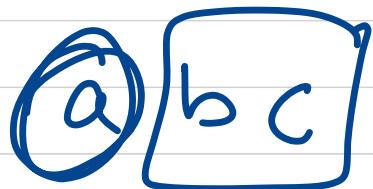
Swap( )

f( )

Swap( )

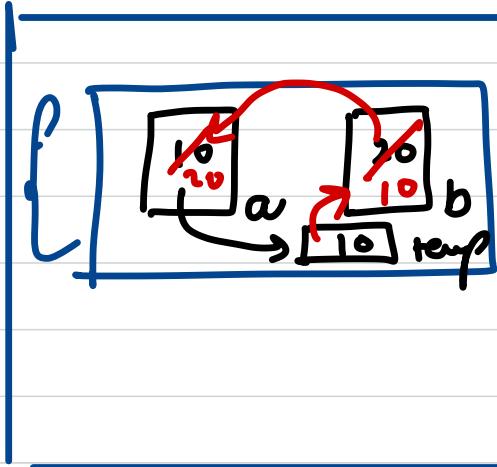


#Self work → To give every char a chance  
to become the first char.

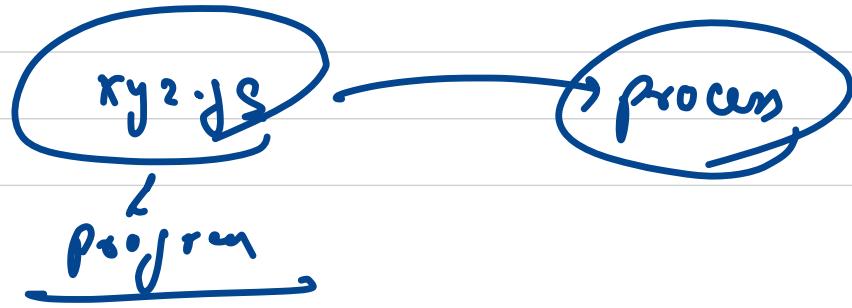


..  
..

Stack  
frame

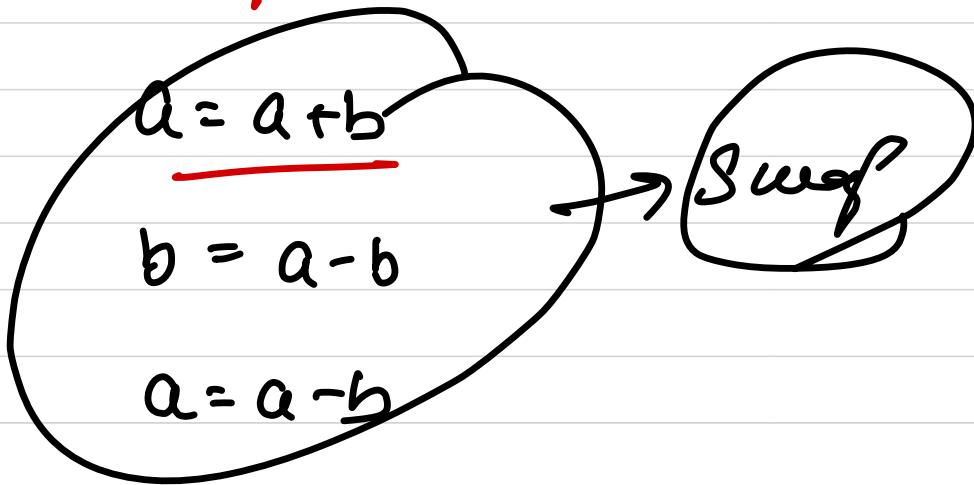


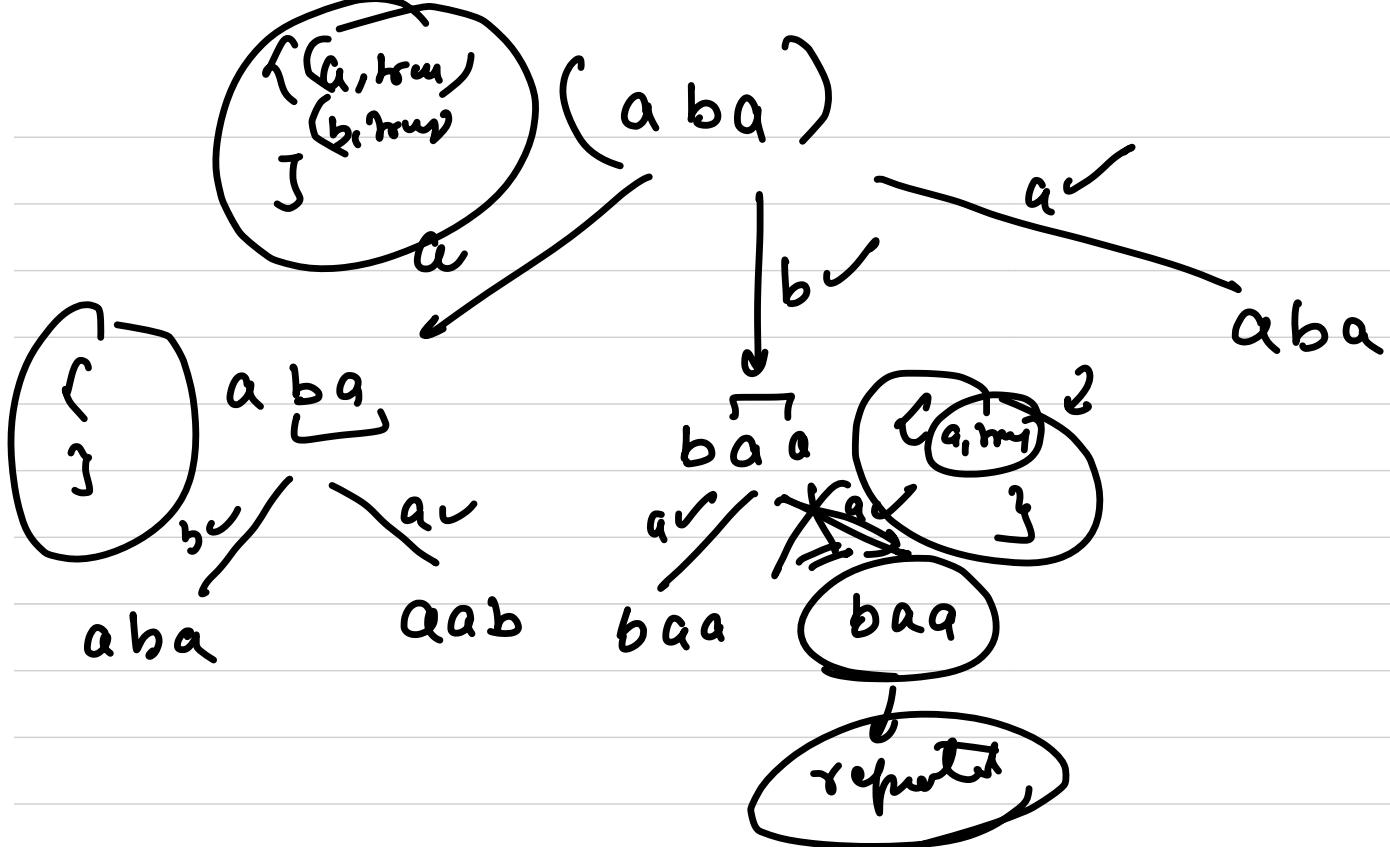
RAM



$$a = \cancel{10} \cancel{30} 20$$

$$b = \cancel{20} \cancel{10}$$





over" → over

= {

key - value

}

(top)

([a,b,c], "...")

([b,c], "a")

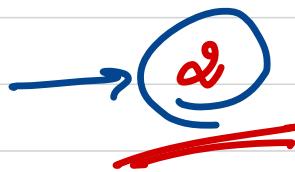
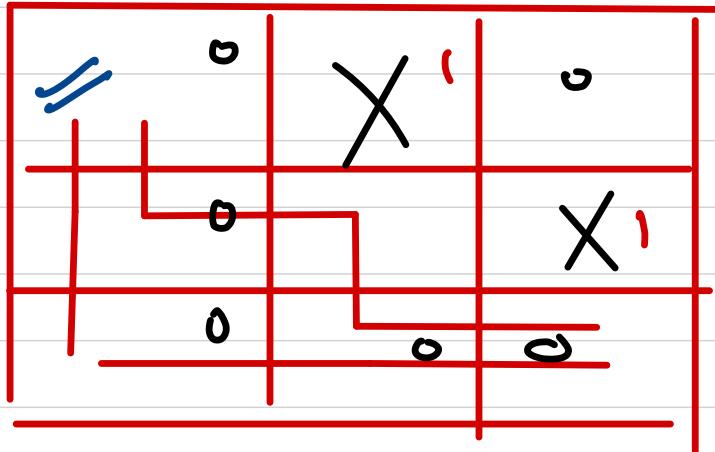
([a,c], "b")

([a,b], "c")

([c], 'ab')

([b], 'ac')

Dn Given a matrix of  $m \times n$ . You're starting from top left. From every cell you can either go right or down. Few cells are open and few are blocked. From blocked cell we cannot go anywhere. All the movement can happen on the open cell. Calc the no. of ways to reach bottom-right.



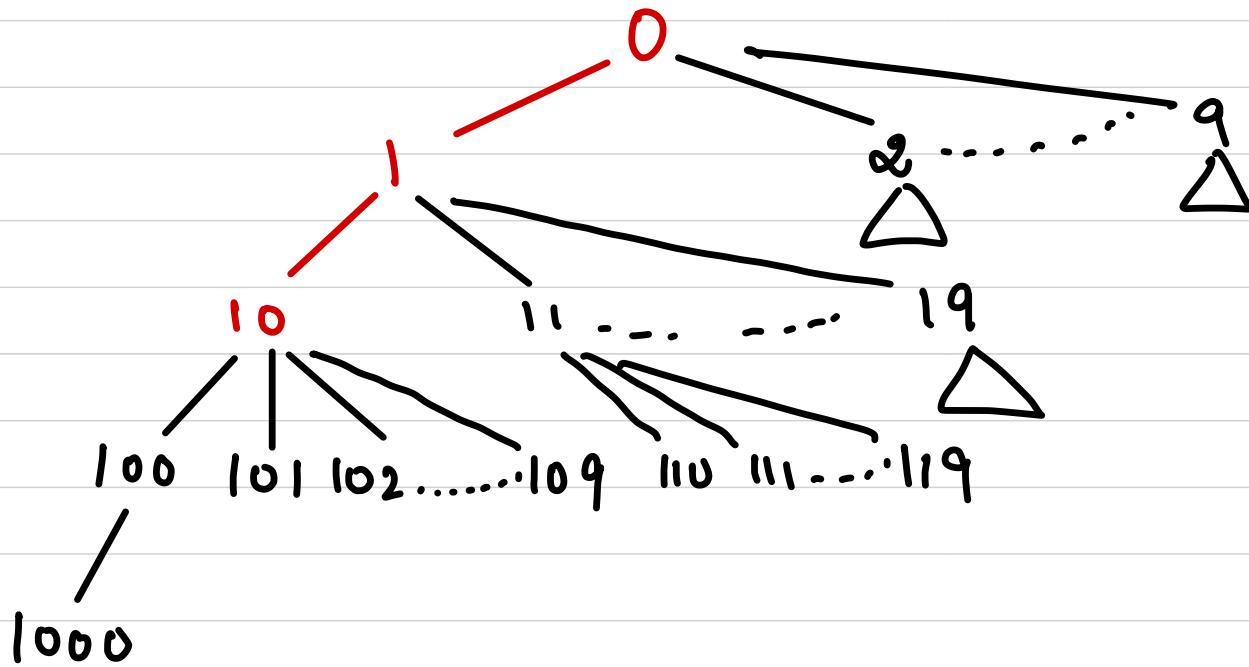
~~Fliegend~~

if ( $\overset{2}{\text{grid}[i][j] == 1}$ )  
return 0;

# # Demographical order

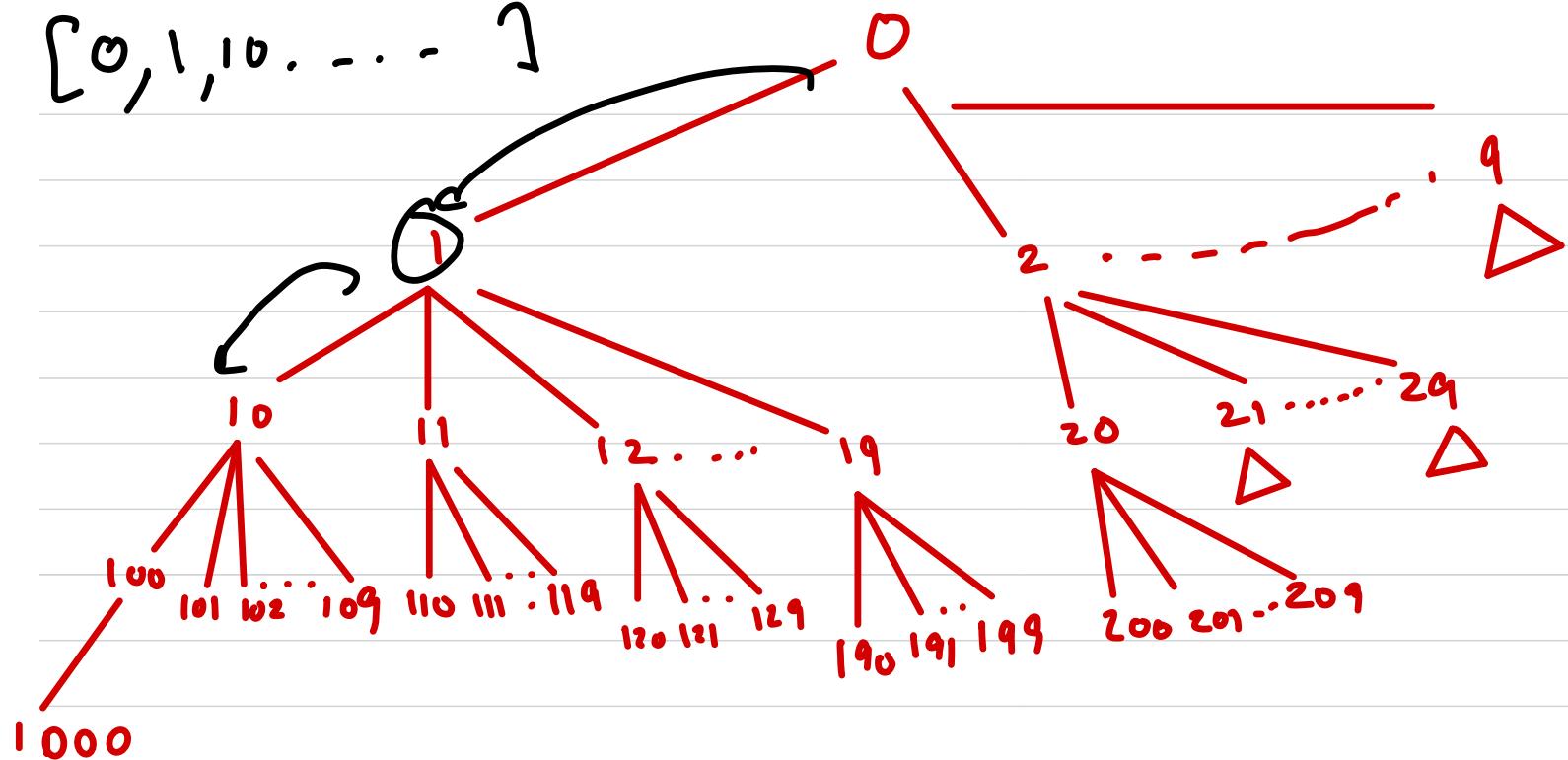
0-1000

0-a  
1-b  
2-c  
3-d  
⋮ ⋮



baaa  
bab  
bac

$[0, 1, 10, \dots, 2]$



$$f(i, n) = f(10xi + 0, n)$$

↓  
points numbers

from  $i, n$

in lexic order

$$f(10xi + 1, n)$$

$$f(10xi + 2, n)$$

⋮

$$f(10xi + 9, n)$$

Ans →  $f(0, n)$

$\text{DCCCLX}$

$X$

$(C)$

$(C)$

$C)C$

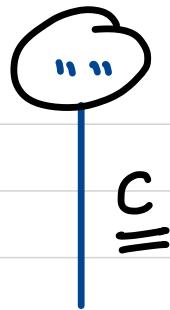
$X$

$DC$

$X$



print all well formed balanced parentheses  
of length  $2n$ .



" ( ) ( ) ( ( ) ) "

| c

↳ for an empty string or an already balanced string, we can't add closing parentheses at the end.

if `open-parenthesis-count == n`, Then we can only add closing.

$f(\overbrace{\text{open}, \text{closing}, \text{rest}, n}^{\text{count}}) =$

$f(\text{open}+1, \text{close},$   
 $\text{rest} + "(", n)$

if  $\text{open} ==$   
 $\text{close}$   
and  
 $\text{open} < n$

prints balanced

strings of length  $\underline{\underline{n}}$

$f(\text{open}, \text{close}+1, \text{rest} + ")", n)$  else if  $\text{open} ==$

$f(\text{open}+1, \text{close}, \text{rest} + "(", n)$

$f(\text{open}, \text{close}+1, \text{rest} + ")", n)$

Base Case

if ( $\text{close} == n$ )  
console.log(rest);

$n=3$

$2^n \Rightarrow$

"open  
n<sub>103</sub>"

$(0, 0, "", 3)$



$(1, 0, "C", 3)$

$(1, 1, "C)", 3)$

$(2, 0, "CC", 3)$

$(2, 1, "C)C", 3)$

$(3, 0, "CCC", 3)$   $(2, 1, "C(C)", 3)$

$C$

$\sim \approx 3$

$(0, 0, "", 3)$

$(1, 0, 'c', 3)$

$(2, 0, 'cc', 3)$

$(1, 1, 'c)', 3)$

$(3, 0, 'ccc', 3)$

$(2, 1, 'cc)', 3)$

$(2, 1, 'c)c', 3)$

$(3, 1, 'ccc)', 3$

$(3, 1, 'c)cc', 3)$

$(2, 2, 'cc)', 3)$

$(3, 2, 'cccc', 3)$

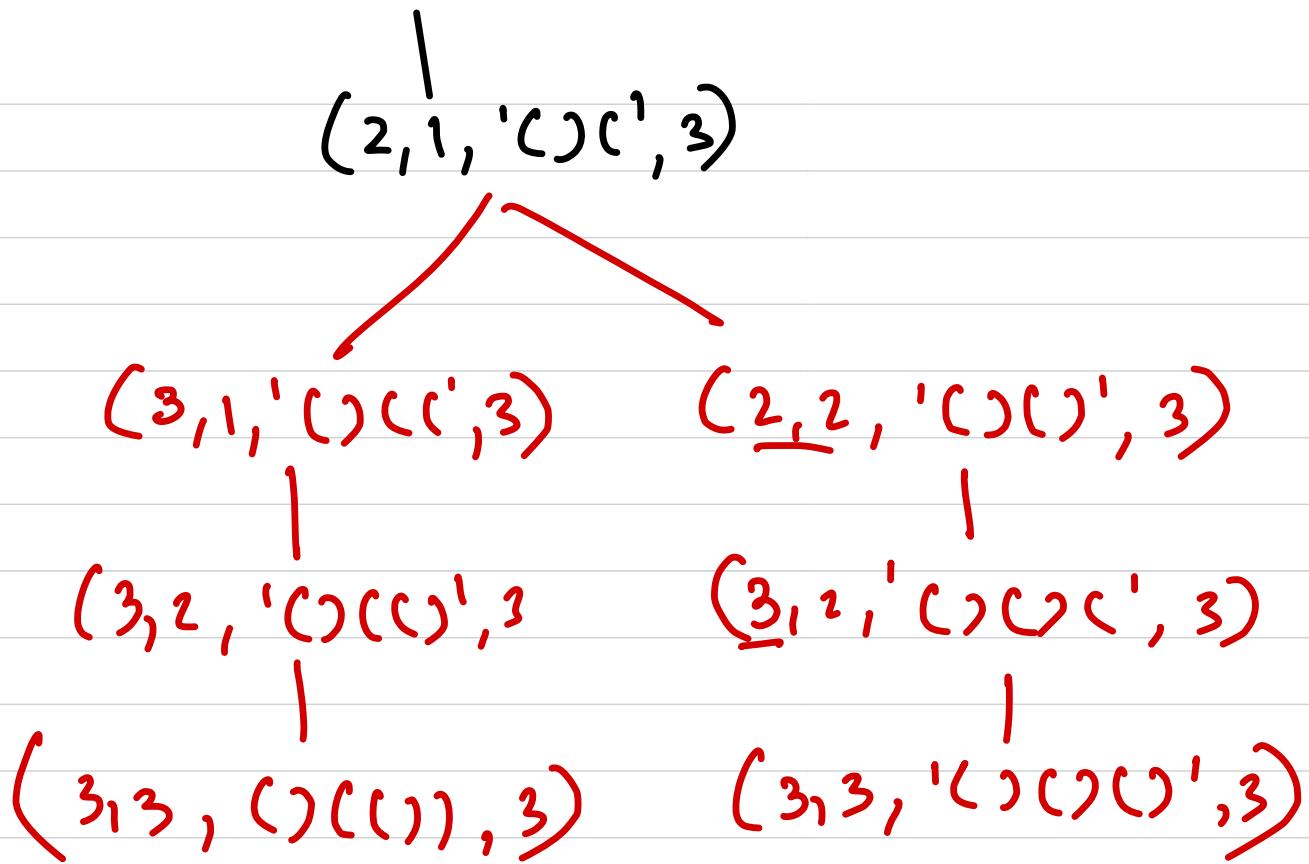
$(3, 2, 'c)cc', 3)$

$(3, 2, 'c)cc', 3)$

$(3, 3, 'cccc', 3)$

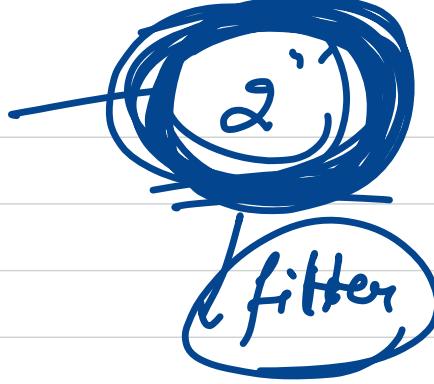
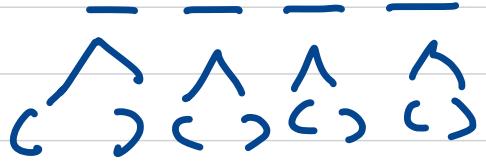
$(3, 3, 'c)ccc', 3)$

$(3, 3, 'c)ccc', 3)$



So logic is based on the fact that how  
many opening & closing parentheses you have.

$$\frac{n=2}{2^n}$$



$m \leq k \leq n$

$k$

1	2	3
4	5	6
7	8	9

get me all  
pattern  
which start  
starts with  
1

all pattern starts with 1  $\rightarrow x$

all pattern starts with 2  $\rightarrow y$

all pattern starts with 5  $\rightarrow z$

Ans  $\rightarrow$   $4xz + 4xy + 2$

$$m = 2$$

[ 2, 10 ]

$$n = 10$$

$$m \leq k \leq n$$

$k = 2, 3, 4, 5, 6, 7, 8, 9, 10$

add

for ( $i = m; i \leq n; i++$ )

$\underbrace{k}$

$$\overbrace{f(k)}$$

all the unlock

pattern of length k

$$\sum_{k=m}^n f(k)$$

for ( $k = m$ ;  $k \leq n$ ;  $k++$ )

$$\underline{\underline{ans += f(k)}}$$



~~mappy~~

	1	2	3	4	5	6	7	8	9
1	0	0	2	0	0	0	4	0	5
2	0	0	0	0	0	0	0	5	0
3	2	0	0	0	0	0	5	0	6
4	0	0	0	0	0	5	0	0	0
5	0	1	0	0	0	0	0	0	0
6	0	0	0	5	0	0	0	0	0
7	0	0	5	0	0	0	0	2	8
8	0	5	0	0	0	0	0	0	0
9	5	0	6	0	0	0	8	0	0

map

visited  $\rightarrow 9$

$[ \frac{1}{1} \frac{1}{2} \frac{1}{3} \frac{1}{4} \dots \frac{1}{q} ]$

$$f(\underline{\text{visited}}, \underline{\text{cur}}, k) = \sum f(\text{visited}, j, k-1)$$

$\forall j \in [1, 9]$  and  
mapping  $[;]l_j] = 0$   
or  
 $\text{vis}[\text{mapping}([;]l_j)] =$   
Tour

and

$\text{visited}[j] \Leftarrow$   
False

$f(\text{visited}, 1, k-1) \rightarrow x$

$$q_x + q_y + z$$

$f(\text{visited}, 2, k-1) \rightarrow y$

$f(\text{visited}, 3, k-1) \rightarrow z$

$x = \cancel{x}_1 \cancel{x}_2 \cancel{x}_0$

	1	2	3
1	5	6	
2	8	9	

①

