

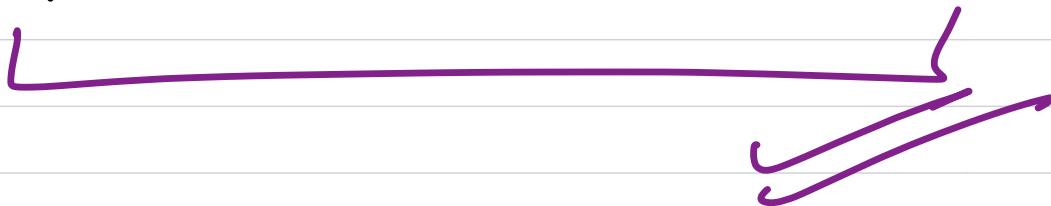
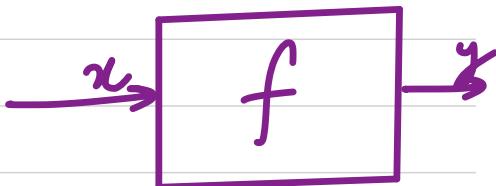
Recursion

→ 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

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

funcⁱ 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 $n \Rightarrow n \times (n-1) \times (n-2) \dots \times 2 \times 1$

$\overbrace{\qquad\qquad\qquad}^{\rightarrow n!}$

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

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

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

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

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

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


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

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

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 funcⁱ

$f(n)$



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 funcⁿ works fine for $b-1$
i.e. $f(a, b-1)$ is correct.

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

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

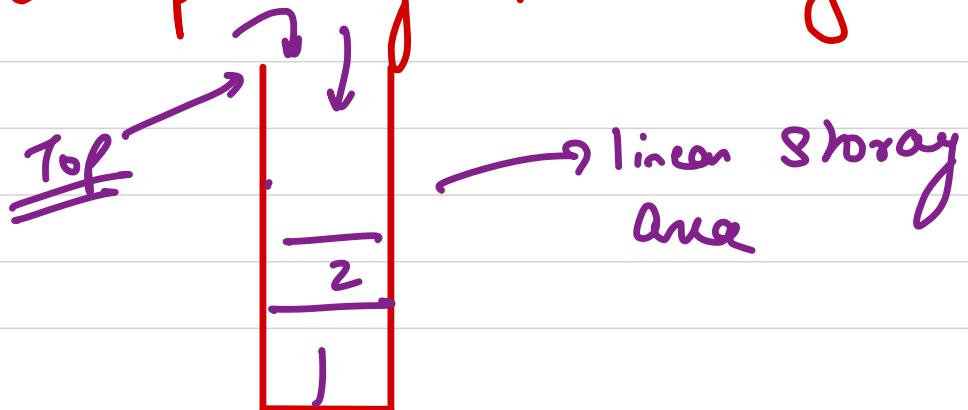
↓
Conut

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

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

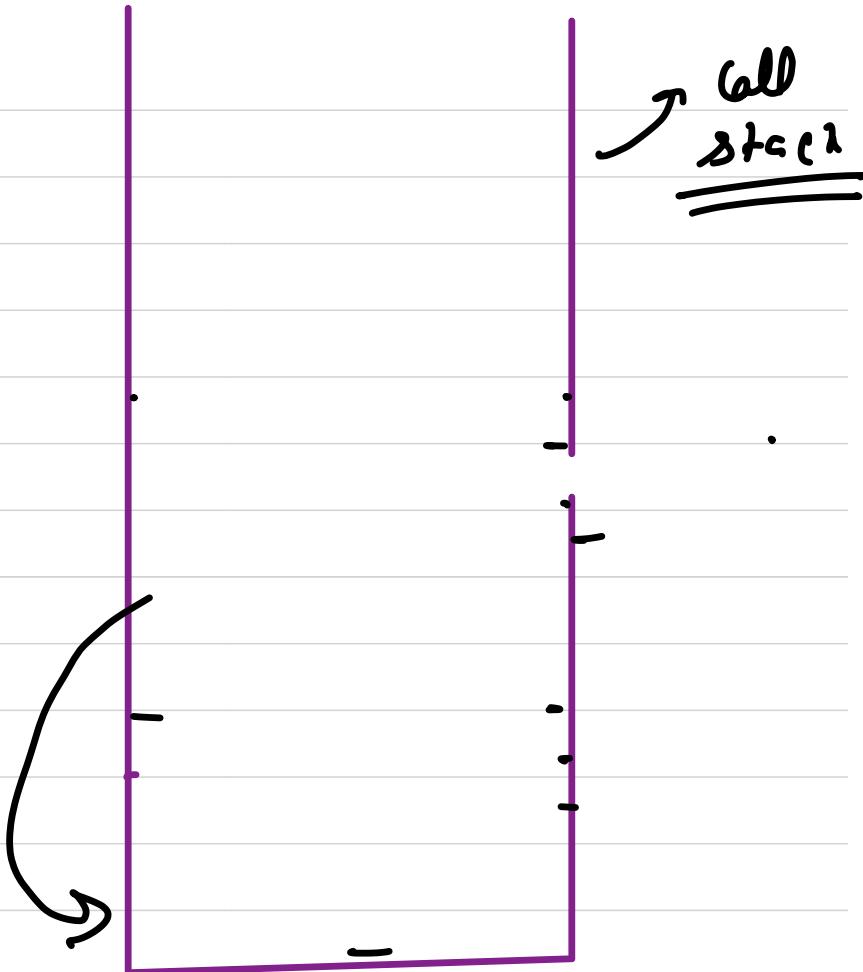
7 let z = fun(b);

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.

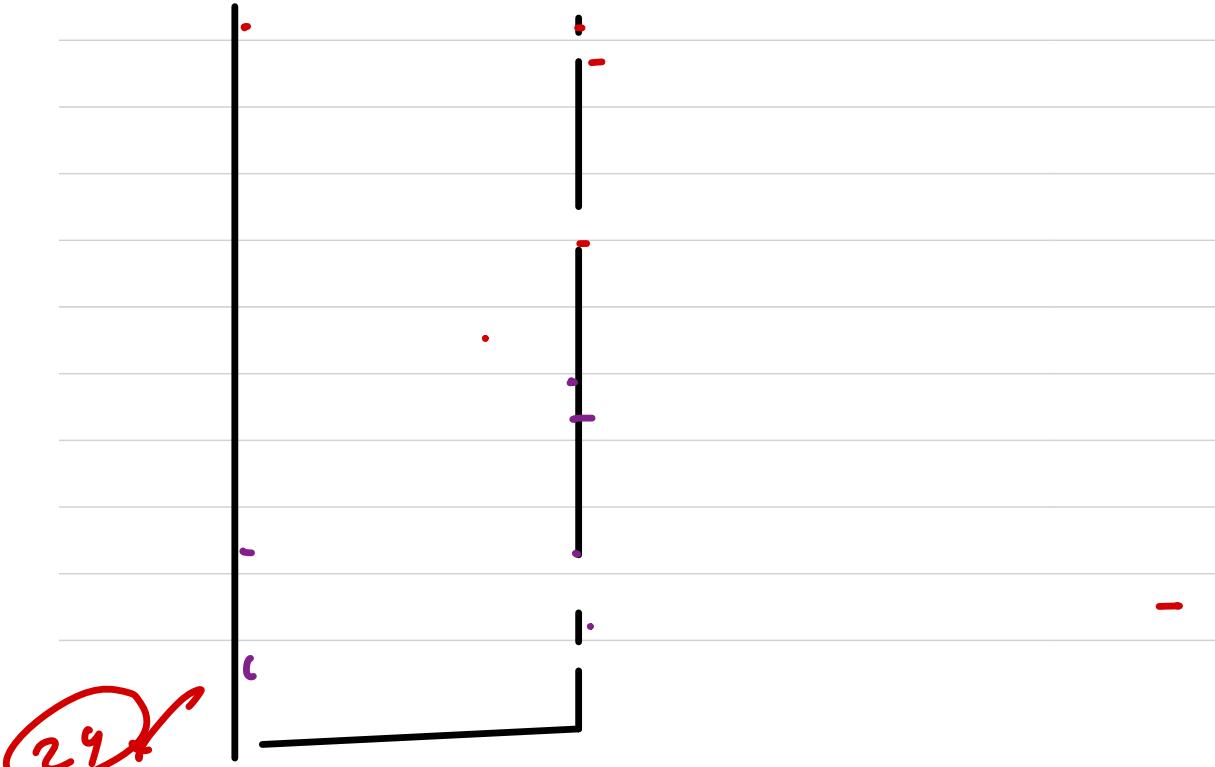
This new entry is called stack trace.

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)



~~ϕ^n~~ Given a value n , calc n^{th} fibonacci.

starting value

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

$n = ?$

ans = 13

Solve it recursively

n^{th} fib

$f(n)$



return n^{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

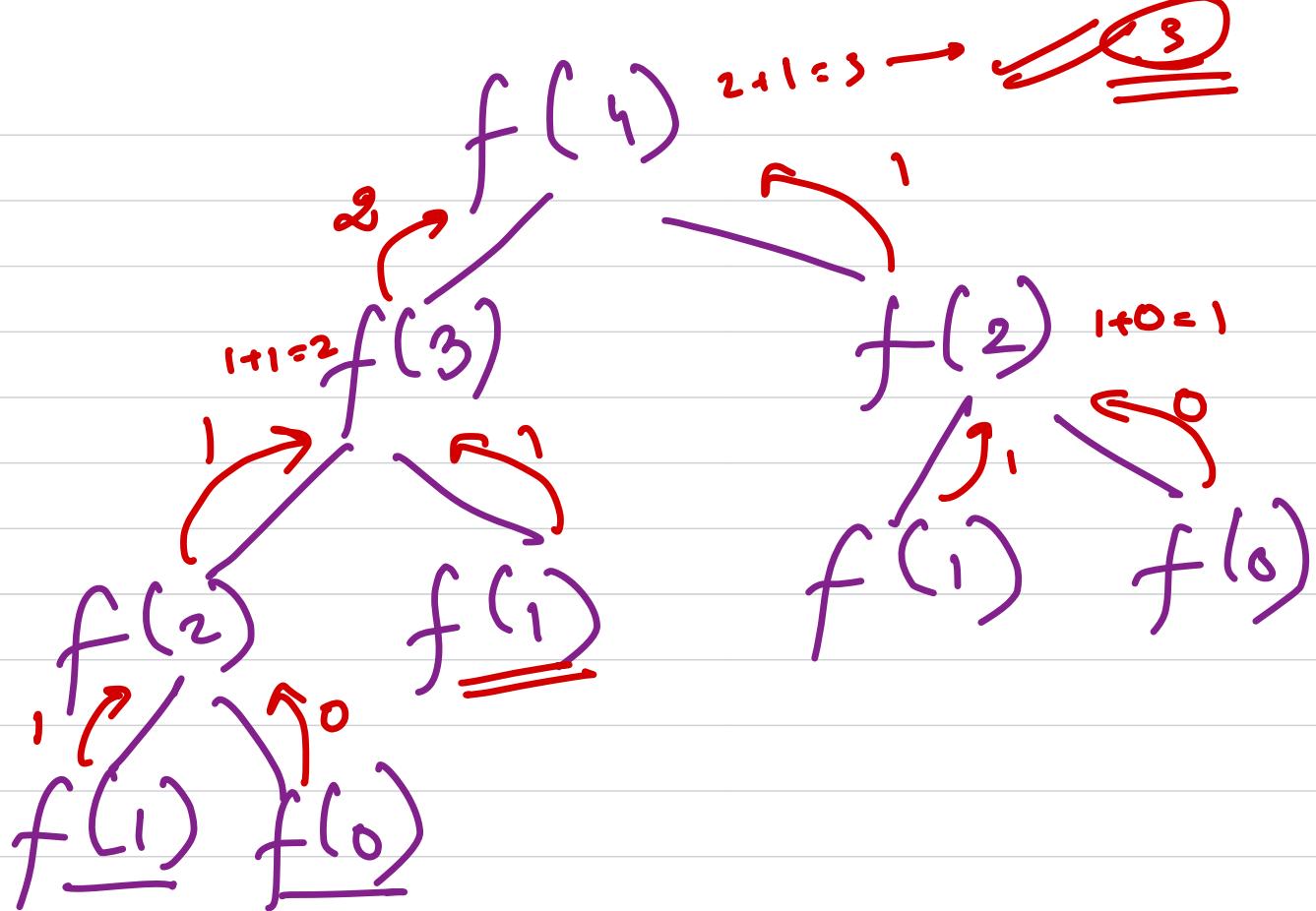
Self work → $f^{(n-1)} + f^{(n-2)}$

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

```
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

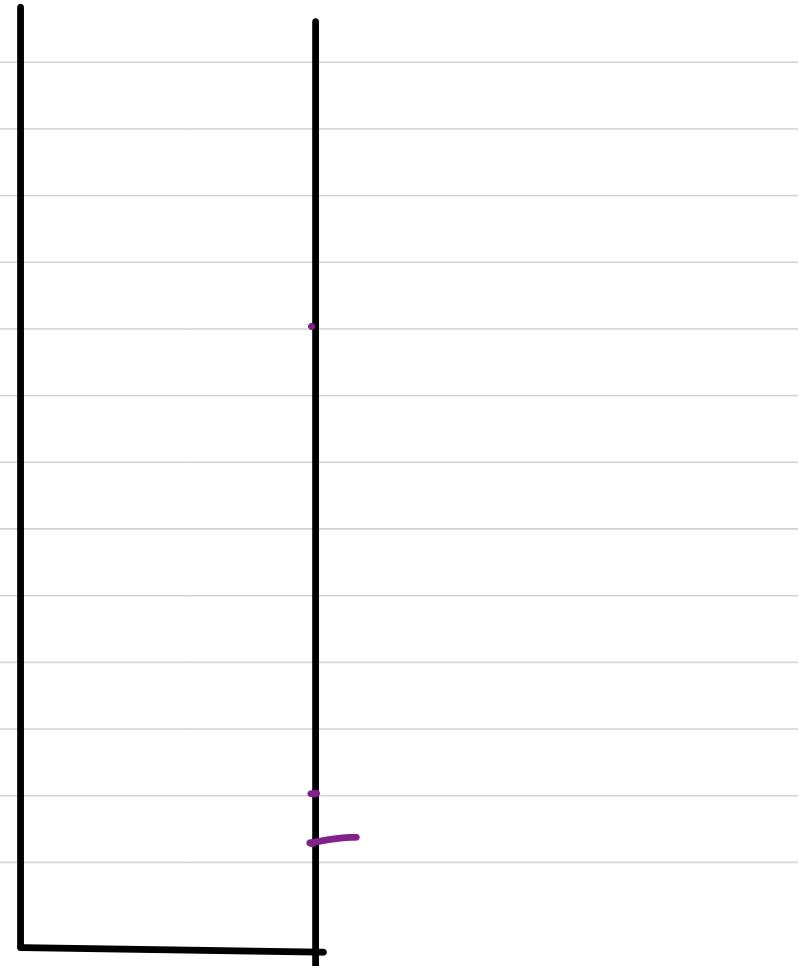
$f(n)$

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 $n=3$

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 → $(A)(B)$]
 (AB)]

$n=9$

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

nth foiled

alone fair

what if D goes alone ??

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

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

$$n=4$$

A

B

C

D

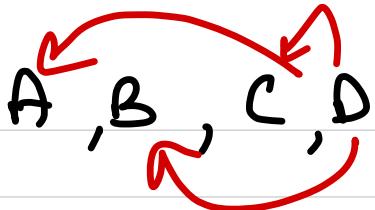
$\rightarrow \underline{n^{\text{th}} \text{ friend}}$

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

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

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

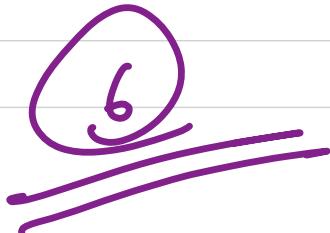
After making a pair how many elements left ?? $\rightarrow (n-2)$



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

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

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



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

Ω^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	

$$\underline{\underline{\quad\quad}}$$

n

1

2

3

4

5

ans

2

3

5

8

13

fibonacci

strings
0, 1

00, 01, 10

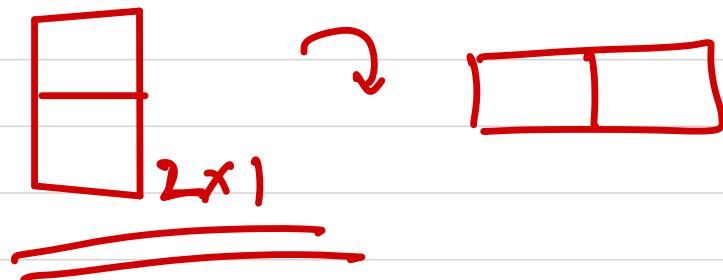
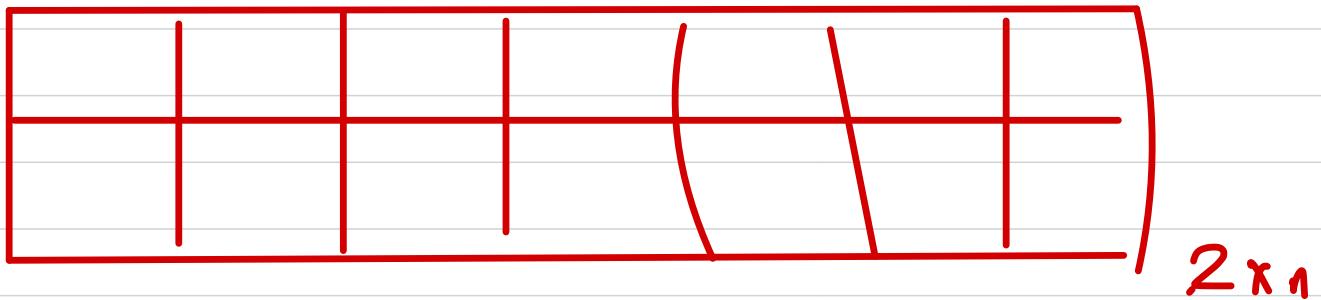
000, 001, 010,
100, 101

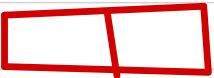
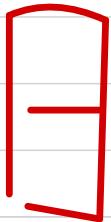
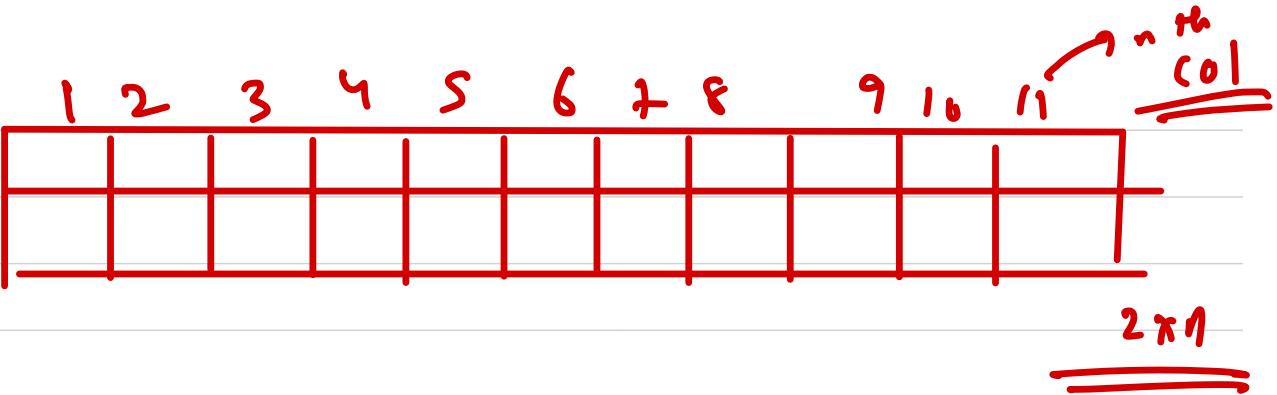
0000, 0001, 0010
0100, 1000, 1001
1010, 0101

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^{th} column you can put a tile either vertically or horizontally

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

value of the
no. of ways to
put tiles on

$2 \times n$ board

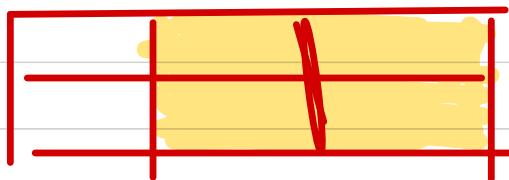
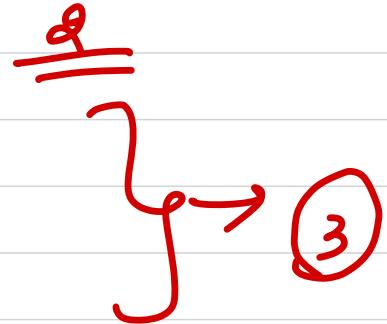
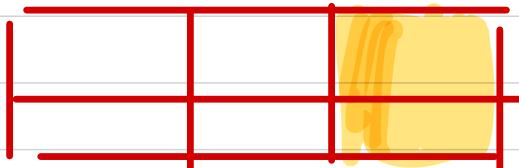
n^{th} board
in part a
vertical
tile

+

n^{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

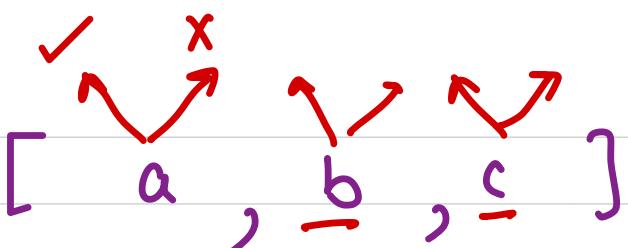
1 2 3

empty

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

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

Permutation Combination

 [a , b , c]

$2 \times 2 \times 2 \dots$

x ✓ ✓

$\rightarrow b\ c$



✓ x ✓

$\rightarrow a\ c$

x ✓ x

$\rightarrow b$

✓ ✓ ✓

$\rightarrow a\ b\ c$

x x x

$\rightarrow \text{empty}$

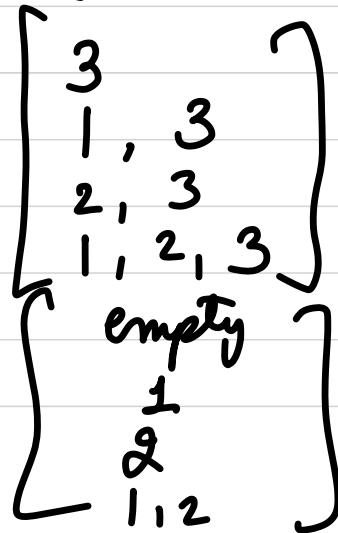
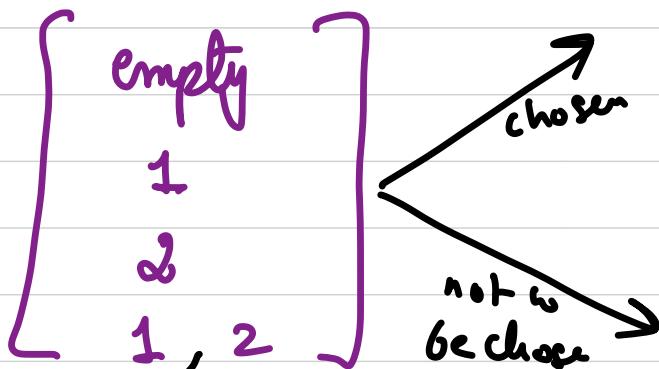
✓ ✓ x
x x ✓

$\rightarrow a\ b$
 $\rightarrow \therefore c$

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

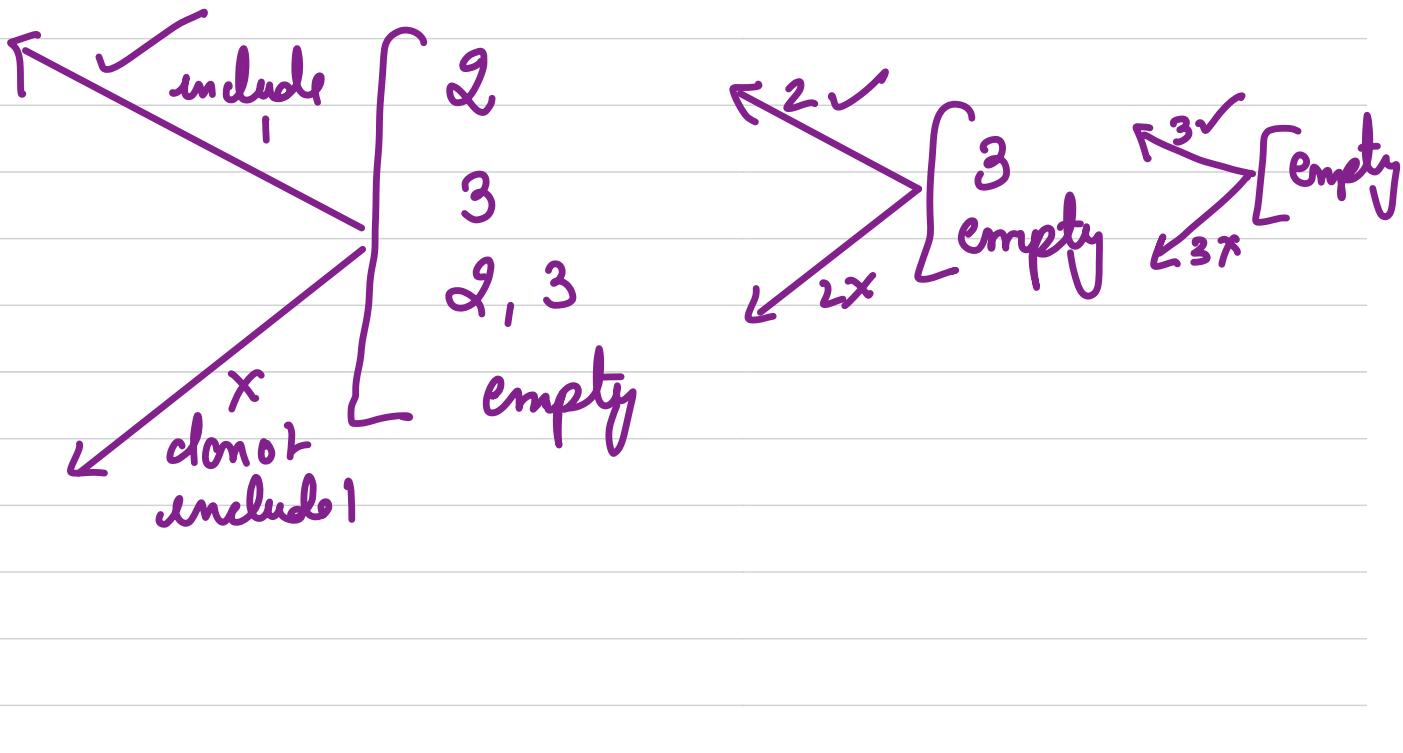
$$[1, 2, 3]_n$$

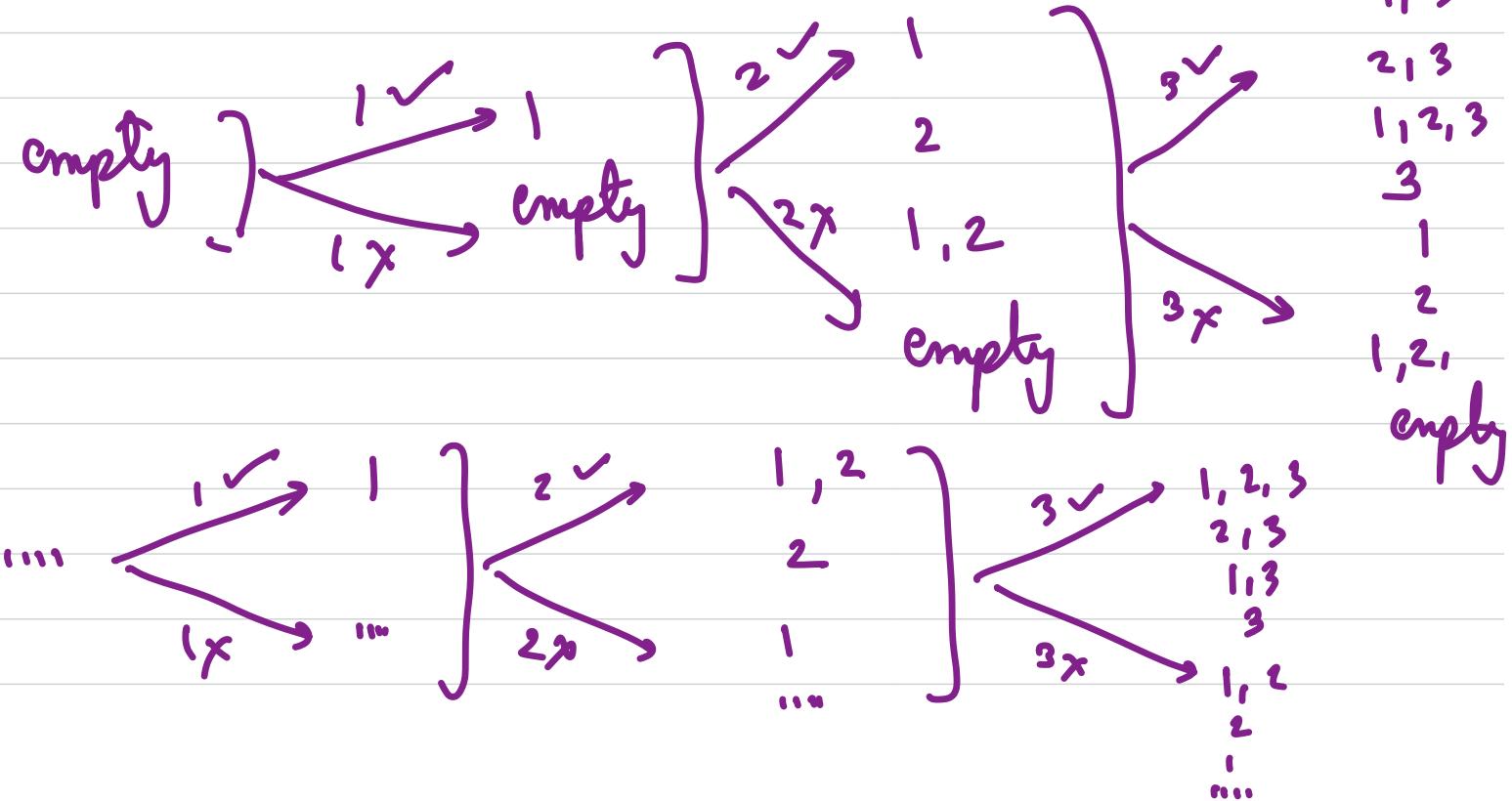
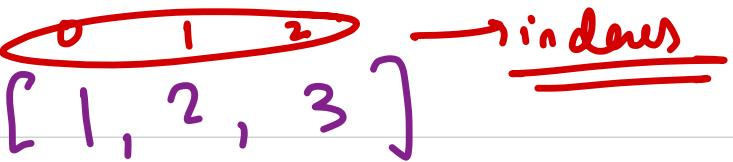
→ we can take the
choices of n^{th}
element.



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)

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

$(\alpha, "", 0)$

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

$1v$

$1x$

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

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

$2v$

$2x$

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

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

$(\alpha, S, 3)$

$2v$

x

$(\alpha, 6 7, 3)$

$(\alpha, 6, 3)$

$(\alpha, "", 1)$

$1v$

$1x$

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

$(\alpha, "", 2)$

$(\alpha, "9", 3)$

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

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 ≥ 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,

$$f(n) =$$

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

returns the min

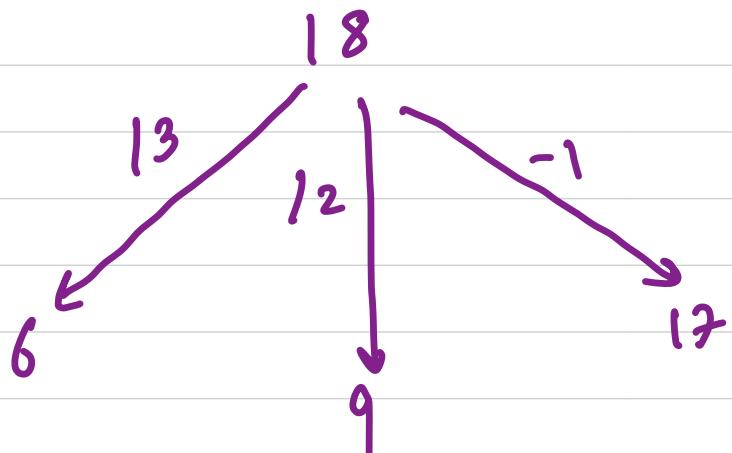
No. of ops reqd to

reduce $n \rightarrow 1$ with

our given ops

Base Case

$$\begin{aligned} n &= 1 & \xrightarrow{0} & 0 \\ n &= 2 & \xrightarrow{1} & 1 \\ n &= 3 & \xrightarrow{1} & 1 \end{aligned} \quad \left. \right\}$$

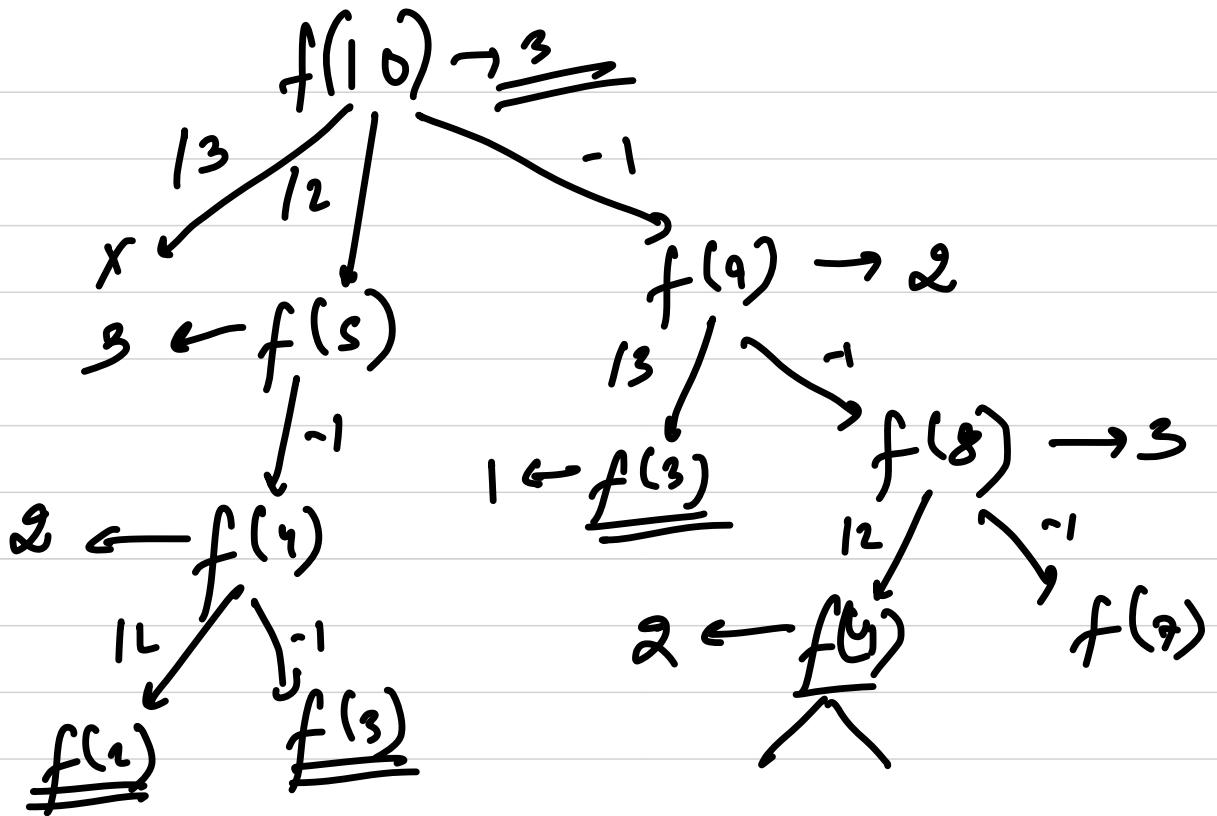


$6 \rightarrow 1 \rightarrow \textcircled{x}$

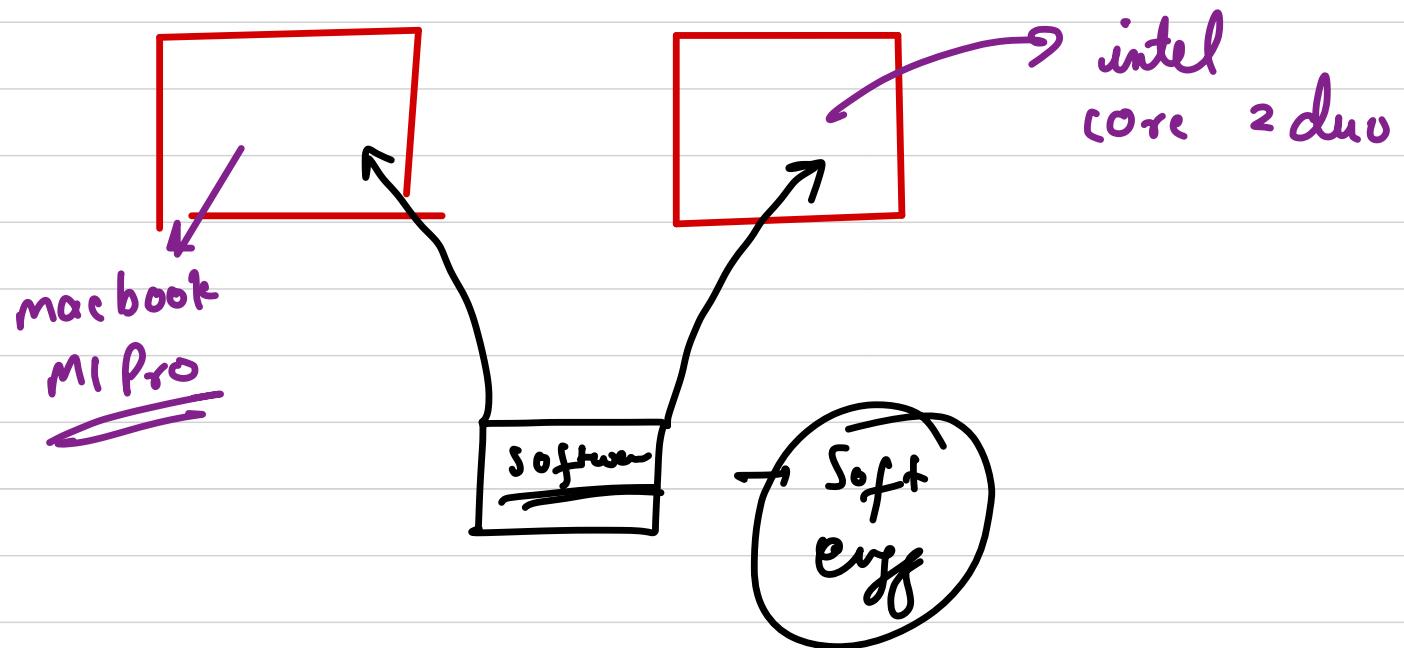
$9 \rightarrow 1 \rightarrow \textcircled{y}$

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

$$1 + \min(x_1, y_1, z)$$

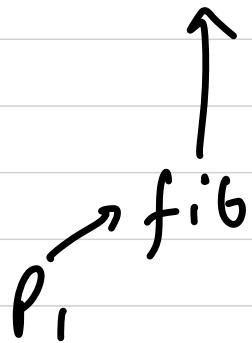


$$\min \left(\underbrace{+\infty}_{\downarrow}, 3, 2 \right)$$

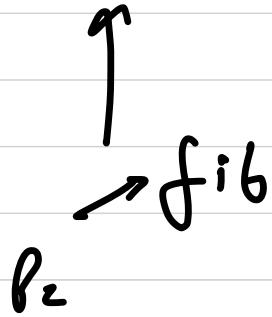


Experimental analysis

macbook
m1 pro



i3



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

Asymptotic Analysis

→ input size dependent

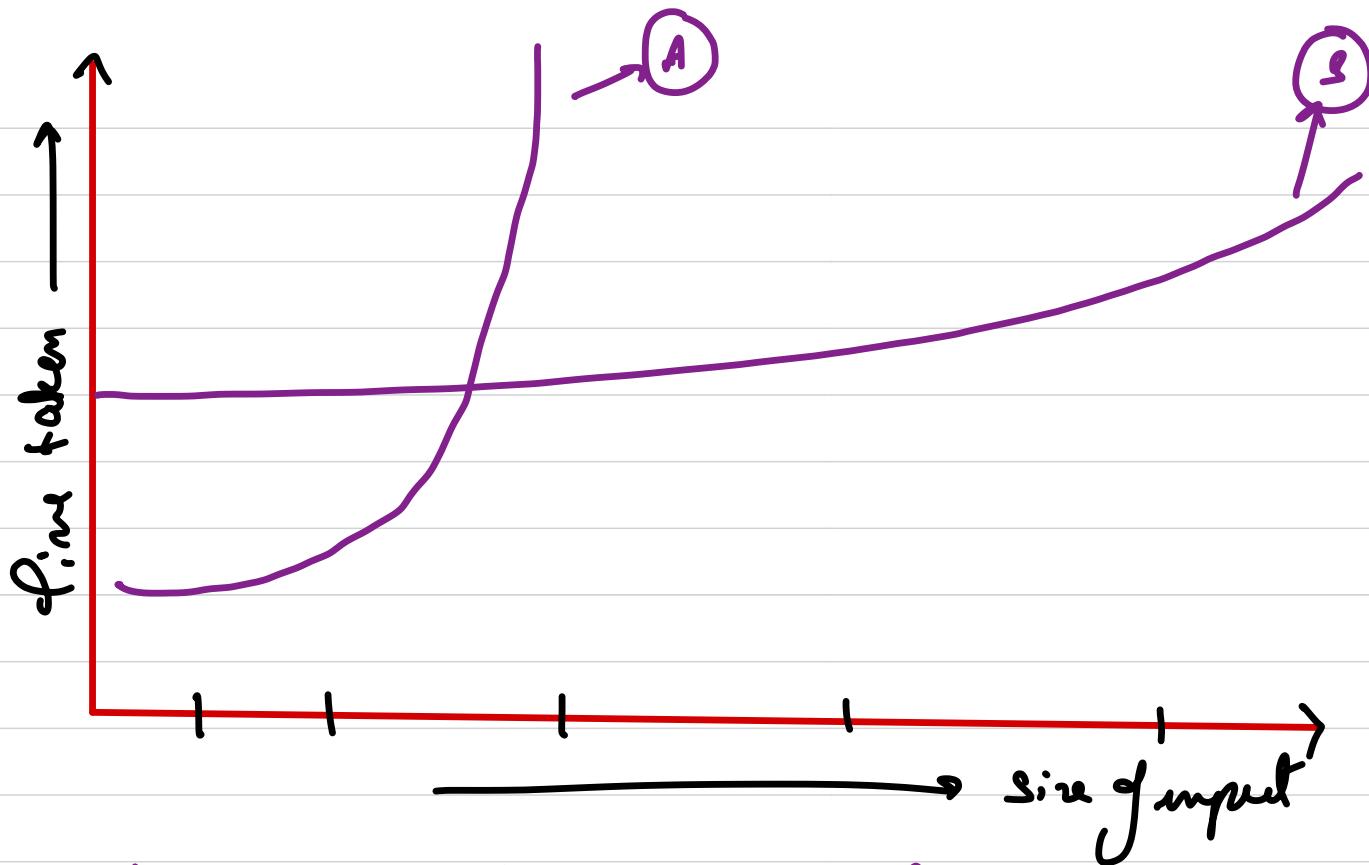
Asymptote

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

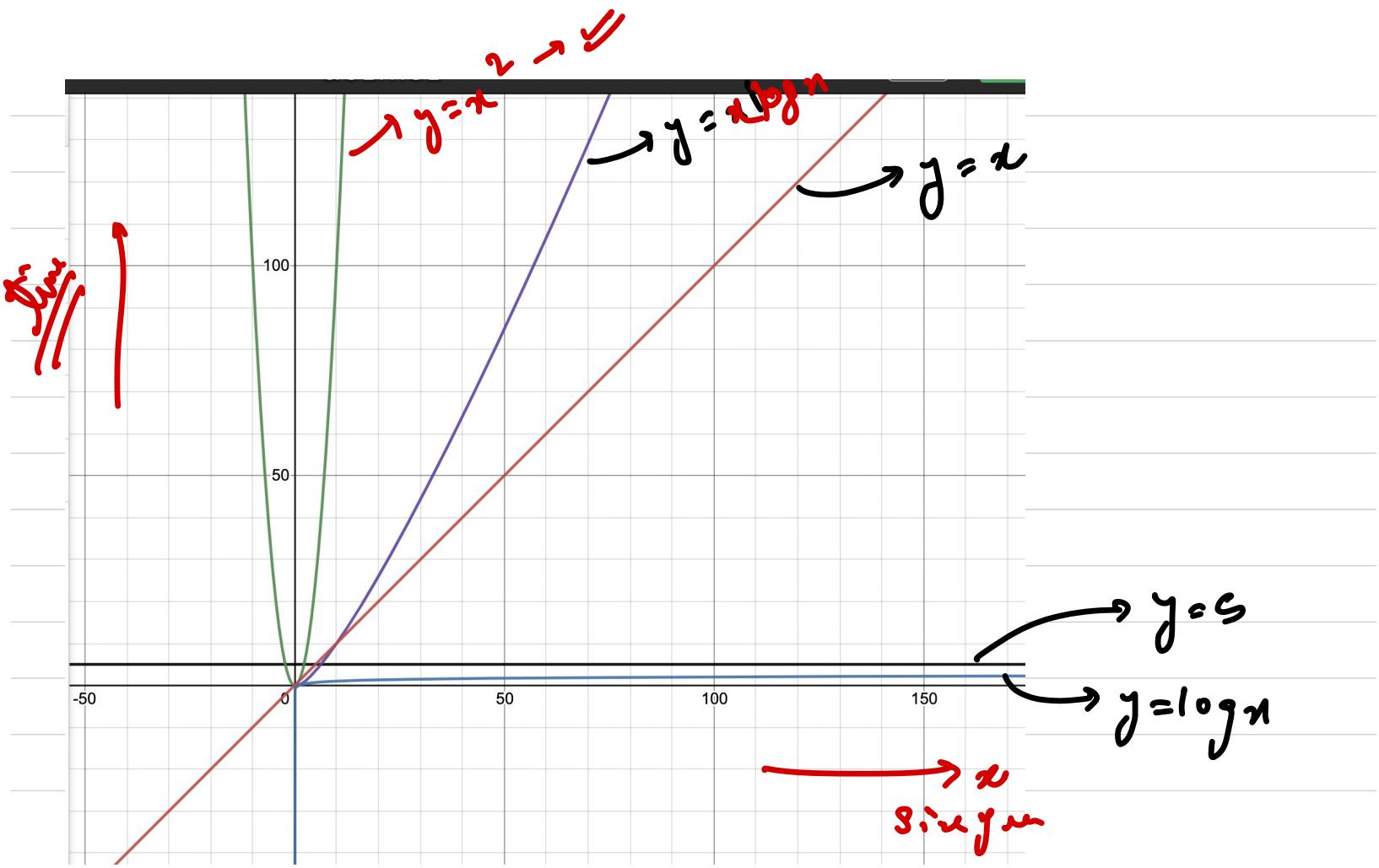
$$2^3 \rightarrow 8$$



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



The rate of growth of B is less



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

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

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

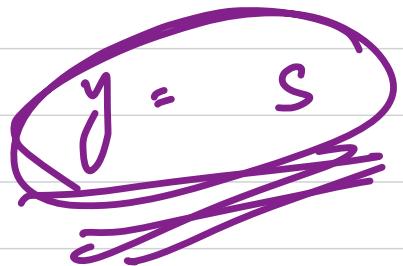
$$y = \log x \rightarrow \frac{dy}{dx} \rightarrow \frac{1}{x}$$

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

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

≈ 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 );  
}
```

~~x~~

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

~~x~~ → x
~~x~~ → x
~~x~~ → x
⋮
~~x~~ → x
⋮

~~x²~~

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

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

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

highest degree term only

$$\underline{\underline{x \rightarrow 10^9}}$$

\Rightarrow

$$3\pi(10^9)^3 + (10^9)^2 + 10 \times 10^9 + 3$$

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

$$K \approx \log_2 n$$

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

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

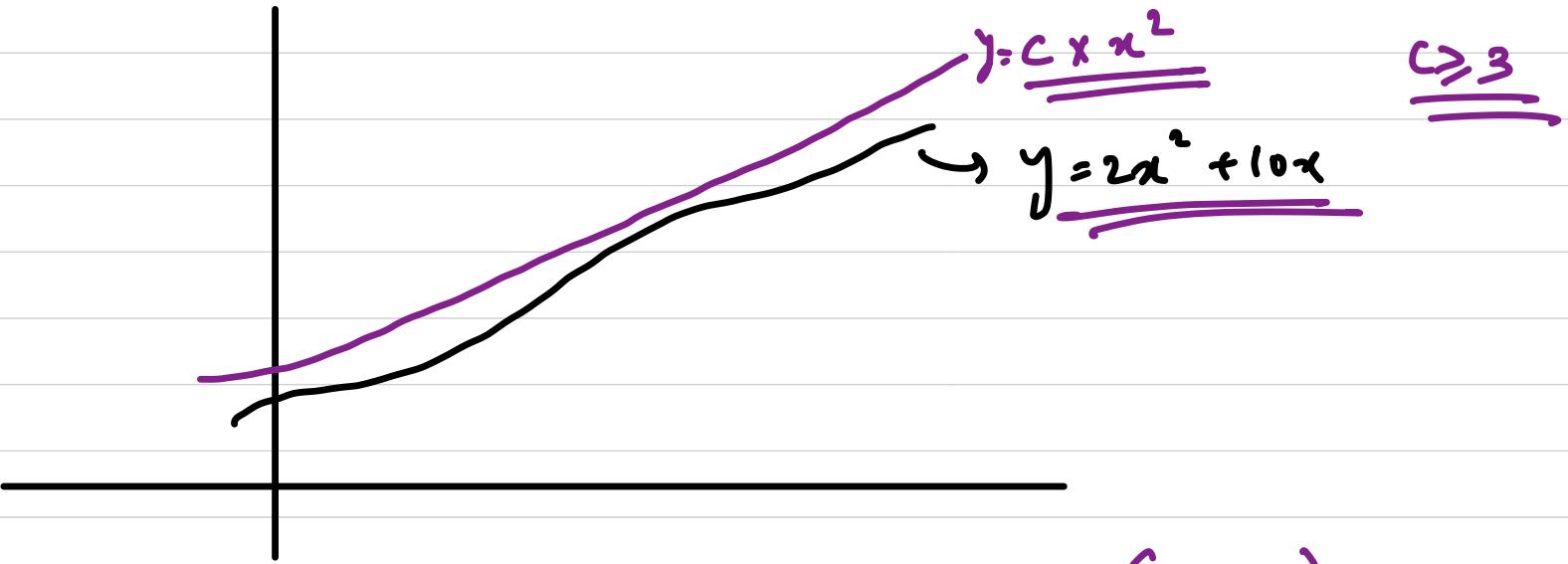
→ last

$$i^* i \leq n$$

$$i^2 \leq n$$

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

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];

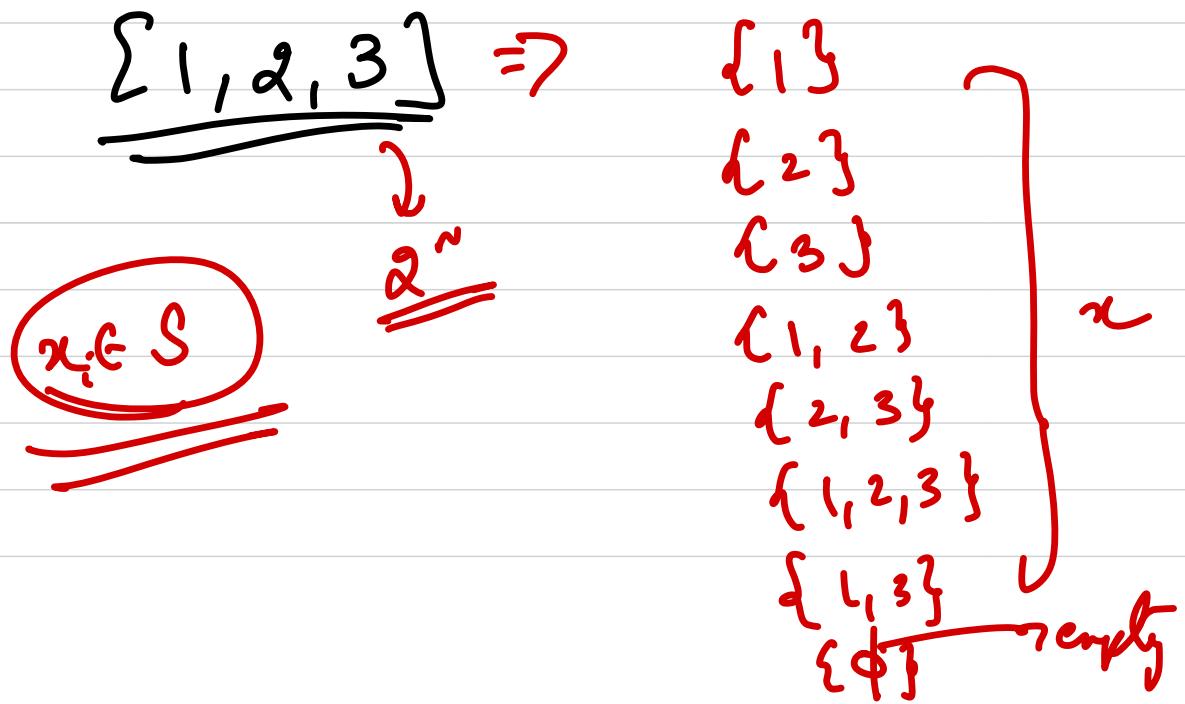
} return³ ans;

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

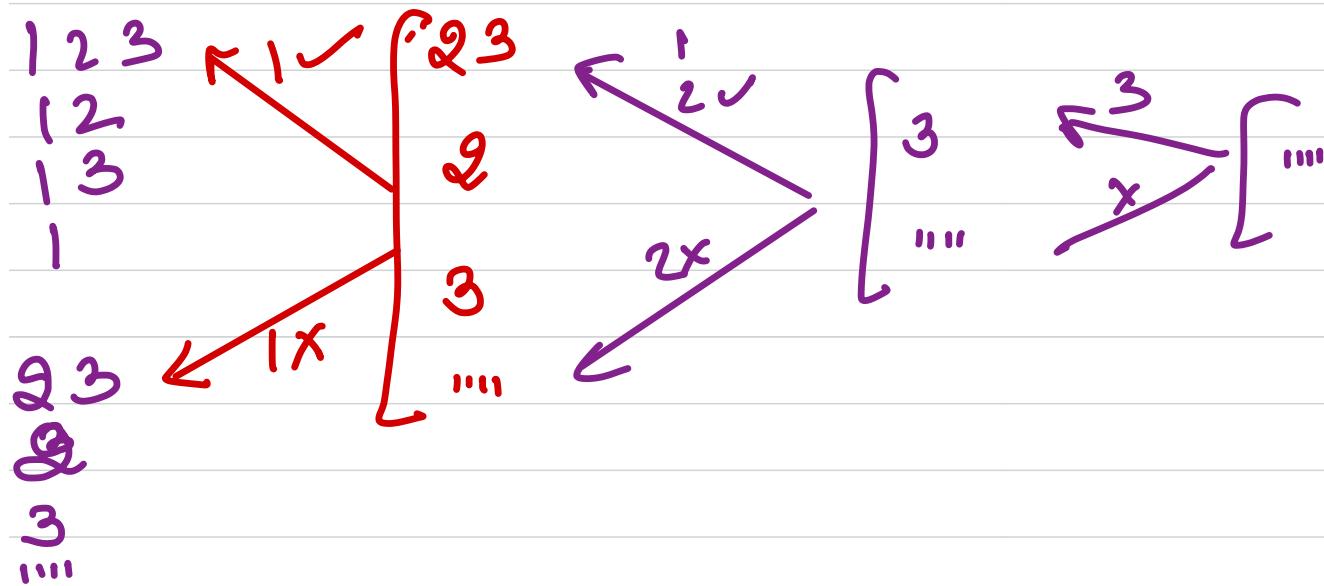
Points the subset
of arr starting from

Index i

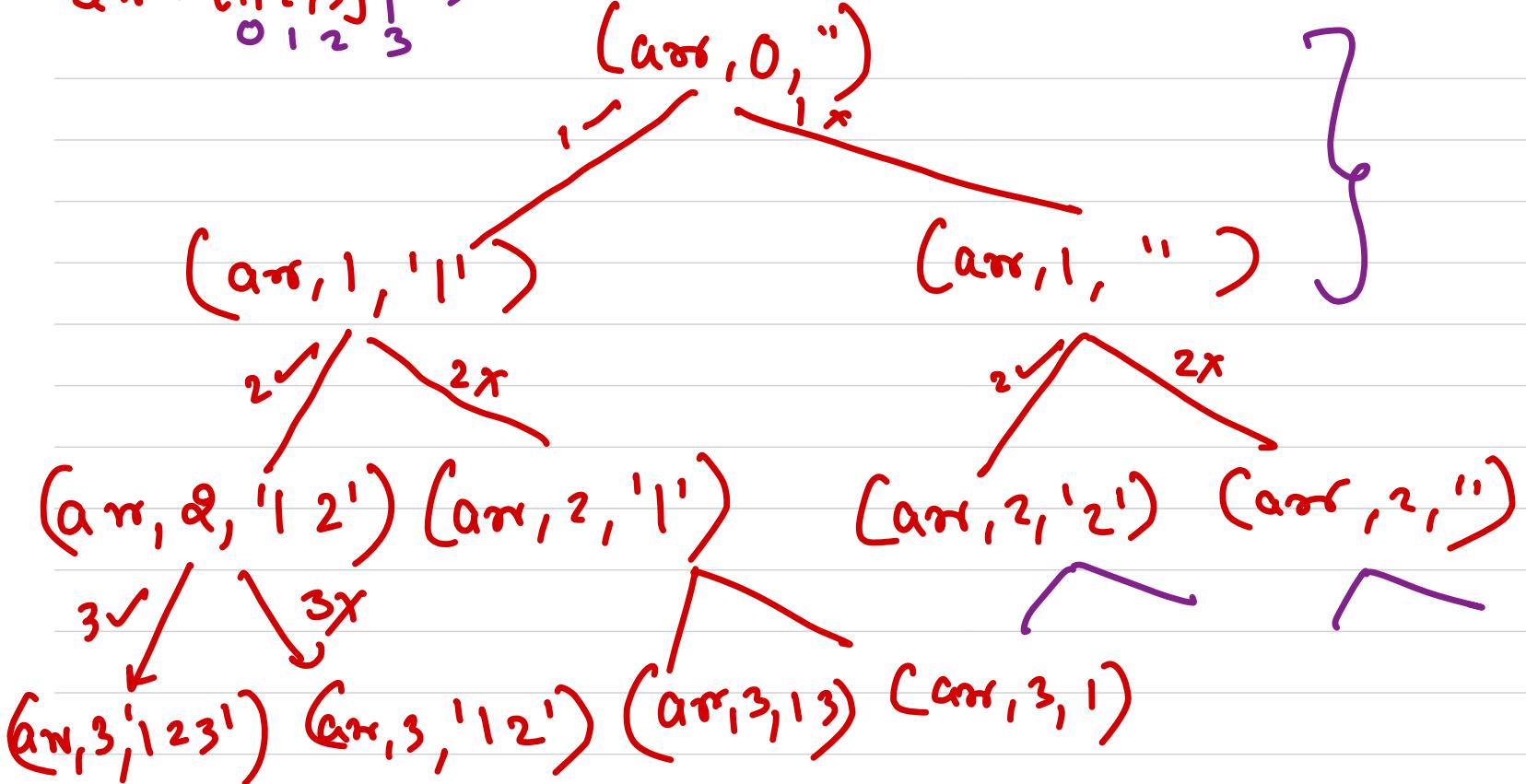
arr

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

$\{1, 2, 3\}$



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



```
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);
```

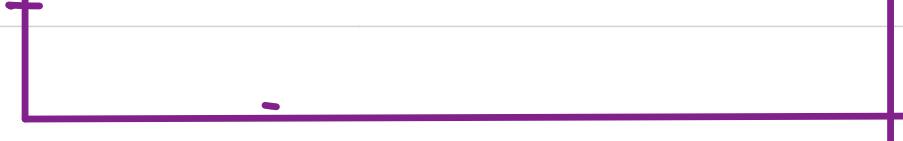
| 4 9 |

| 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

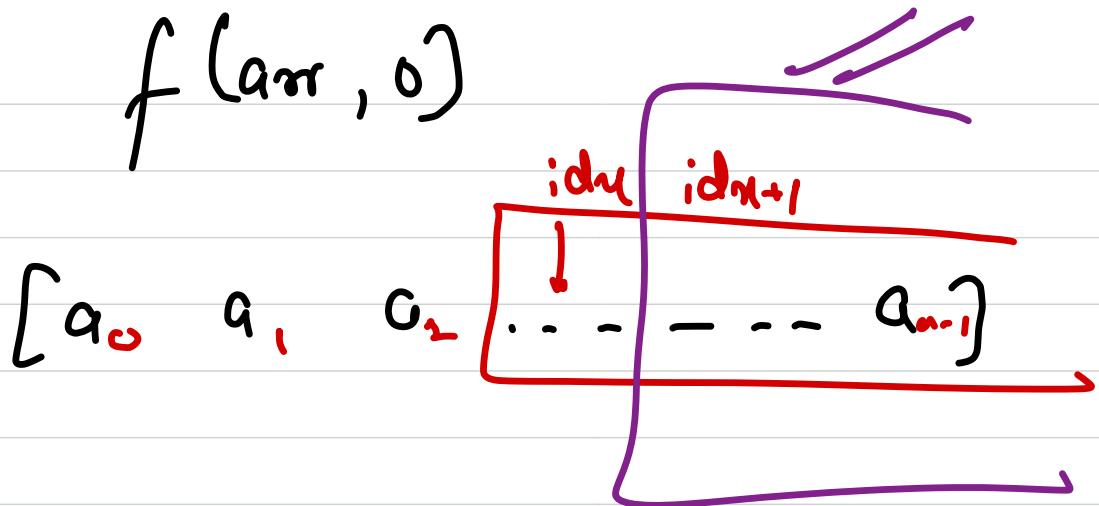
isSorted
 $\text{arr} \rightarrow [1, 4, 2, 3]$
 $\text{idx} = 0$
assume = false

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