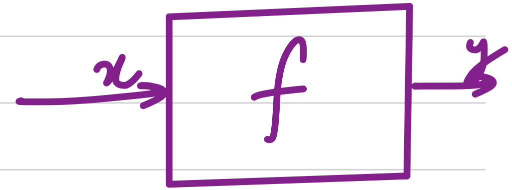
Recursion

→ Recursion is not just a computer science topic

It corelates with maths as **well**.

Mathematically, recursion means a function calling ~~itself~~.



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

$$f(a,b) = \underbrace{a \times f(a,b-1)}$$ → func definition

this function calculates $\underline{\underline{a^b}}$

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

Ex $f(2,3) \to 2^3 \to \underline{\underline{8}}$

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

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

In computer science, recursion means function calling ~~itself~~.

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

```
function gun(x) {
    console.log(x);
}

function fun(x) {
    ...
    ...
    gun(x-1)      → This is not recursion
    fun(x-1);     → This is recursion
    ...
    ...
}
```

Example-1 **Factorial**

$$\underbrace{\text{factorial of any value } n}_{\to\ n!} \Rightarrow n \times (n-1) \times (n-2) \cdots \times 2 \times 1$$

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

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

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

$$f(n) = \underbrace{n \times f(n-1)}$$

this function returns $n!$

The mathematical representation of a recursive function is called **Recurrence Relation**

$5! = 120$ ✓✓

$\hookrightarrow 5 \times 4! = 24$

$\hookrightarrow 4 \times 3! = 6$

$\hookrightarrow 3 \times 2! = 2$

$\hookrightarrow 2 \times 1!$

we already know them

What is $1!$? $\Rightarrow 1$

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

PMI (principal of Mathematical Induction)

Q→ What is the sum of first N natural numbers ??

Ans → $\frac{n \times (n+1)}{2}$

3 step process

1) Base Case → the smallest value for which we can directly verify the <u>ans</u>.

2) Assumption

3) Verification / self task

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

$\hookrightarrow$ Base Case

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

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

Let's prove ourselves that formula is correct for $\underline{\underline{n = k+1}}$

$$\underbrace{\underbrace{1 + 2 + 3 - - - - - - - - k}_{\downarrow} + k+1}$$

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

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

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 solⁿ for factorial of n

We will write a funcⁿ

$f(n)$

return $n!$

Base Case → $(n==1)$ $f(1) \rightarrow 1$

if $(n==1)$ return 1

Assumption → assume the function works fine for $\underline{\underline{n-1}}$

$f(n-1)$ correctly calcs $\underline{\underline{(n-1)!}}$

$\underline{\underline{\text{Selfwork}}}$ → $\underline{\underline{f(n) = n \times f(n-1)}}$

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

$f(a, b)$

$\downarrow$

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

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

Assumption → assume func^n works fine for $b-1$

i.e. $f(a, b-1)$ is correct.

Selfwork → $a^b = a \times a^{b-1}$

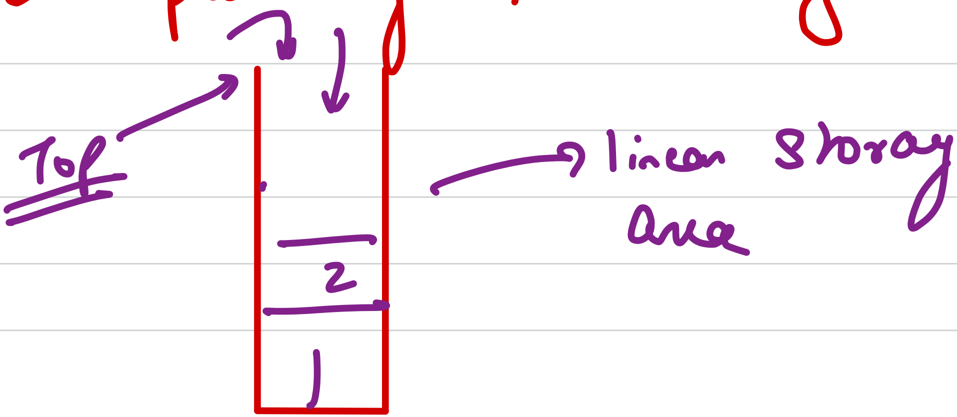
$$f(a,b) \longleftarrow a \times \underbrace{f(a,b-1)}_{\text{correct}}$$

$$27^{51} \qquad 27 \times \underbrace{27^{50}}_{y}$$

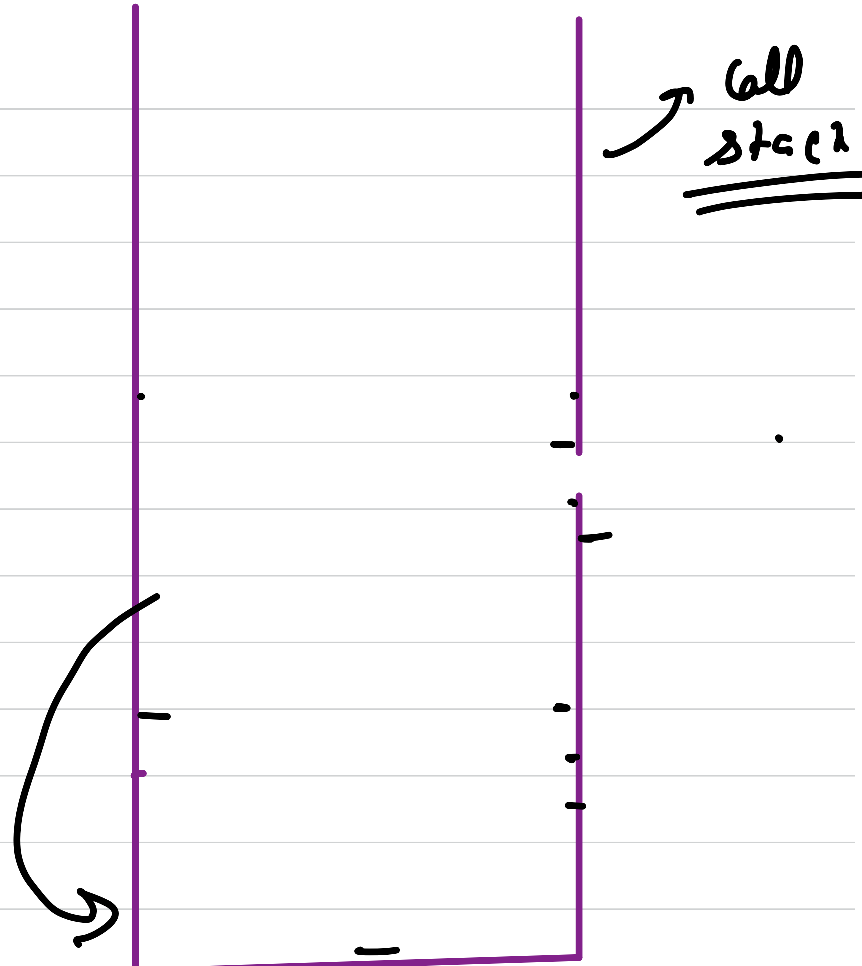
Q: What happens when we call a function ??

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

One part of the memory is **call stack**



```
function fun(x)
  let a = x+10;
  return a;
}

function gen(y)
  let b = y+20
  let z = fun(b);
  return z;
}
console.log(gen(10))
```



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.

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

fact(4)

24

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

starting value

$\rightarrow$ 0, 1, 1, 2, 3, 5, 8, 13, 21 . . . .

0^{th} fib, 1^{st} fib, 2^{nd} fib, 3^{rd} fib, 4^{th} fib, 5^{th} fib, 6^{th} fib . . . .

$n = 7$

Ans = 13

Solve it recursively

n^{th} fib

$f(n)$

return n^{th} fib

Base Case

$f(0) \rightarrow 0$

$f(1) \rightarrow 1$

→ **starter**

Assumption $\rightarrow$ we assume $f(n-1)$ works fine & $f(n-2)$ works fine

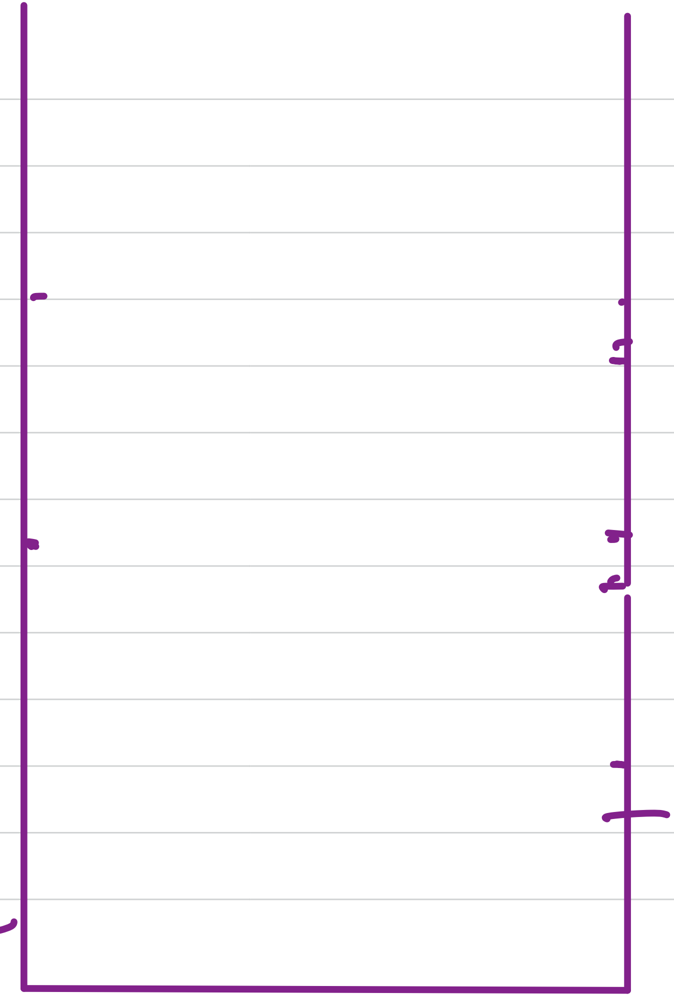
Self work $\rightarrow$ $f(n-1) + f(n-2)$

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

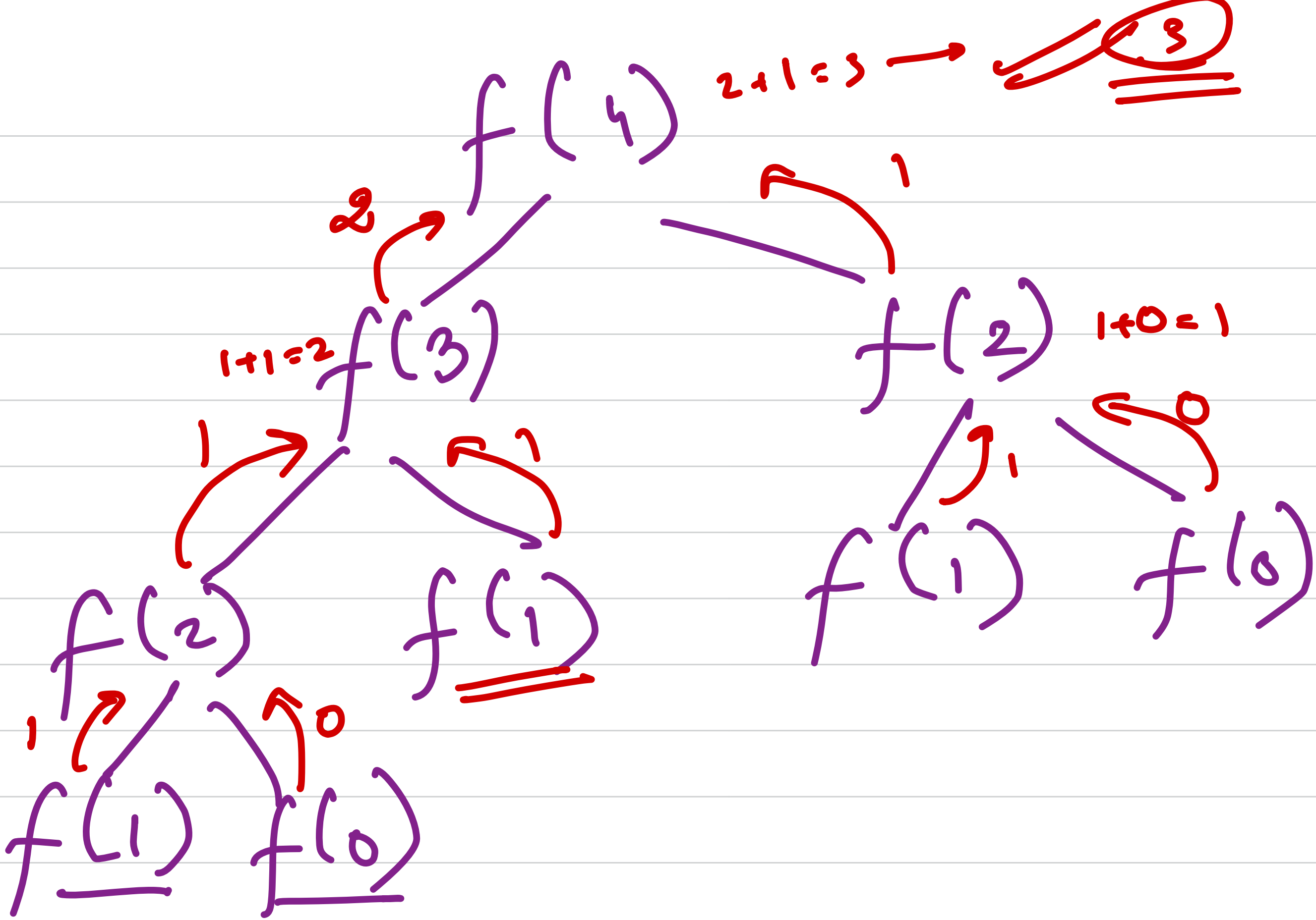
```
function fib(n) {
    // base case
    if(n == 0) return 0;
    if(n == 1) return 1;
    // assumption
    let a = fib(n-1);
    let b = fib(n-2);
    // self work
    return a+b;
}

console.log(fib(3))
```

Recursion takes

Extra space
in memory



Tree



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

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

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

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 → n == 1 → console.log(1)

ans → 1
2
3
4
5

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

self → print 5

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

console.log(n)

$\downarrow$

which print first n natural no.s

assume $f(n-1)$ works correctly

5

5
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

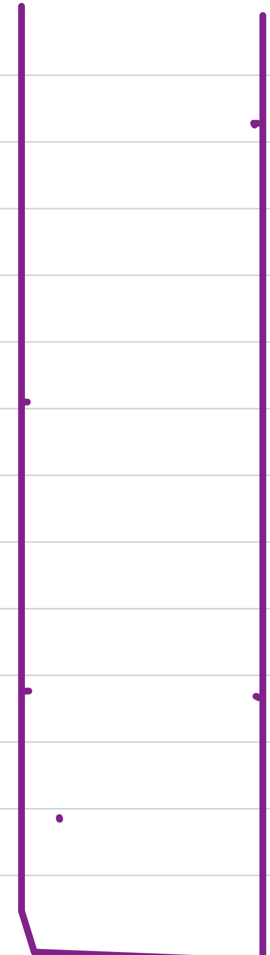
console.log(n)
$f(n-1)$ → assume it works fine

```
function print_inc(n) {
    // base case
    if(n == 1) {
        console.log(1);
        return;
    }
    print_inc(n-1);
    console.log(n);
}

print_inc(3)
```

1
2
3

```
function print_dec(n) {
    // base case
    if(n == 1) {
        console.log(1);
        return;
    }
    console.log(n); // self task
    print_dec(n-1); // recursive assumption
}

print_dec(3)
```

3
2
1



Qn 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 Recursively)

Ex $n = 3$

(A) (B) (C) → ✓

(AB) (C) → ✓

(AC) (B) → ✓

(A) (BC) → ✓

ans → 4

$n = 4$

A B C D

Base Case

$n == 1 \rightarrow$ ans $\rightarrow 1$

$n == 2 \rightarrow$ ans $\rightarrow 2$

A B $\rightarrow$ (A) (B)
(A B)

$n=4$

A B C D

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

$f(n)$ → return the no. of ways in which n friend can go

$f(n-1)$: n^{th} friend send to go alone

$(n-1) \times f(n-2)$: n^{th} friend send to make a pair

$n=4$

A B C D $\rightarrow$ n^{th} friend

D $\downarrow$ alone $\rightarrow$ pair

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??

$\rightarrow$ n^{th} friend

$n=4$ A B C D

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

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

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

3×2

$= 6$

A , B , C , D

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

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

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

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

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

6

Q. Given a number n, calculate the no. of binary strings of length n with no consecutive one.

n = 3

ans → 5

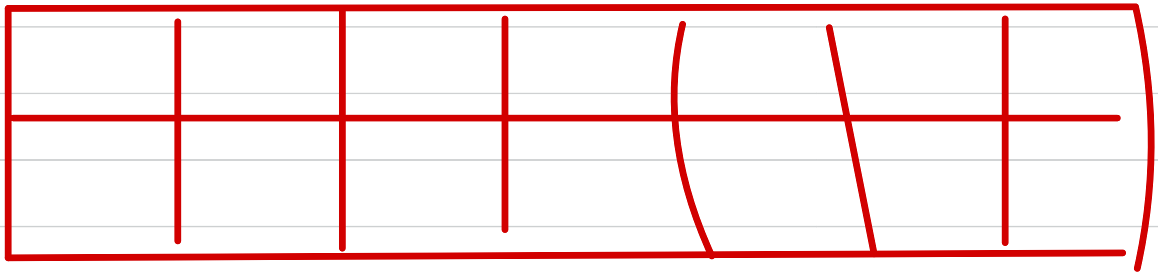
000
001
010
100
101

fibonacci

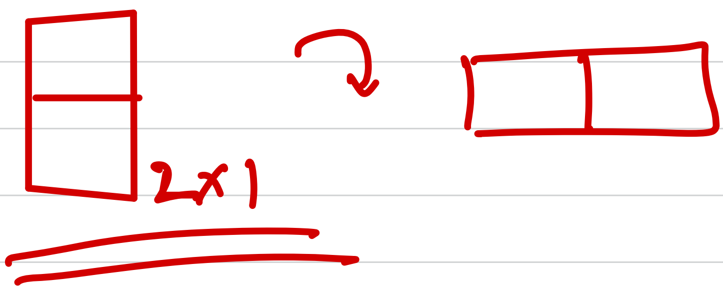
n	ans	Strings
1	2	0, 1
2	3	00, 01, 10
3	5	000, 001, 010, 100, 101
4	8	0000, 0001, 0010, 0100, 1000, 1001, 1010, 0101
5	13	

Base Case → if (n==1) return 2
if (n==2) return 3

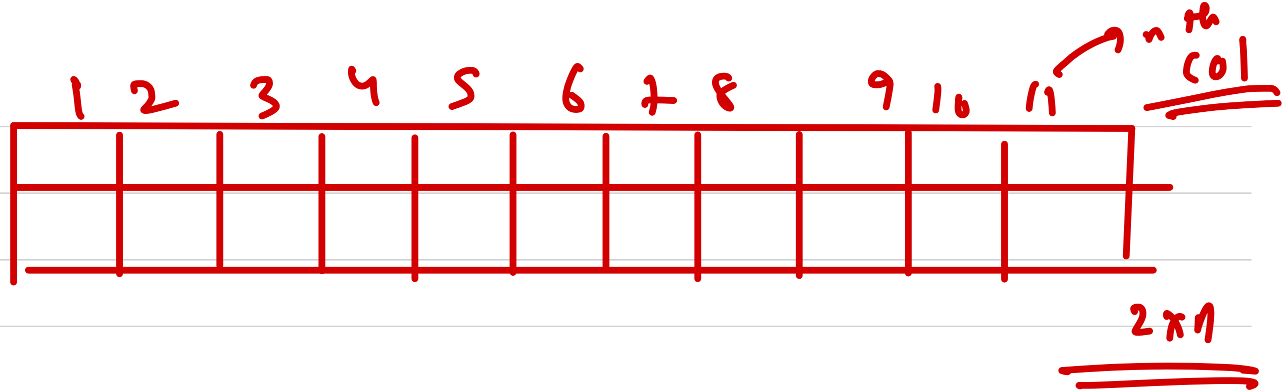
$$f(n) = f(n-1) + f(n-2)$$



2 x 1



2 x 1



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

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

$f(n)$ — return the no. of ways to put tiles on $2 \times n$ board

$f(n-1)$ — n^{th} board we put a vertical tile

$f(n-2)$ — n^{th} board with $(n-1)^{th}$ board filled with horizontal tile

Base Case

$n == 1 \rightarrow 1$

$n == 2 \rightarrow 2$


2
3
1

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 ??

2^n

Permutation Combination

✓ x

[a , b , c]

2 × 2 × 2 .....

2^n

a	b	c	
✗	✓	✓	→ b c
✓	✗	✓	→ a c
✗	✓	✗	→ b
✓	✓	✓	→ a b c
✗	✗	✗	→ empty
✓	✓	✗	→ a b
✗	✗	✓	→ ⋮ c

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

$[1, 2, 3]_n$

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

empty
1
2
1, 2

chosen →

3
1, 3
2, 3
1, 2, 3

not to be chosen →

empty
1
2
1, 2

1st element

[1, 2, 3]

1, 2
1, 3
1, 2, 3
1

2
3
2, 3
empty

✓ include 1

2
3
2, 3
empty

x do not include 1

2 ✓

3
empty

2 x

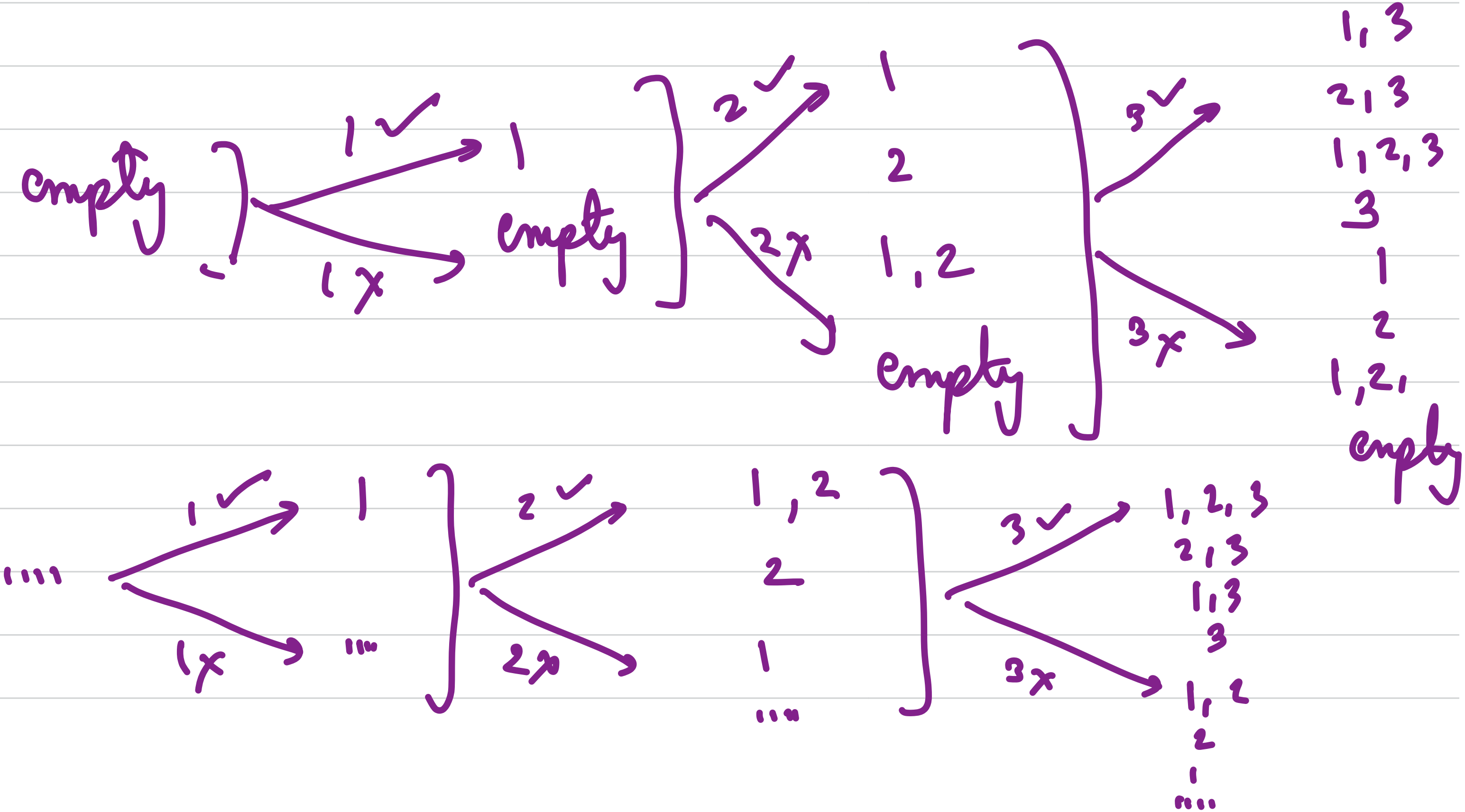
3 ✓

empty

3 x

0 1 2 → indexes

[1, 2, 3]



$f(arr, res, i) = f(arr, res + arr[0], i+1)$ (append)

$f(arr, res, i+1)$

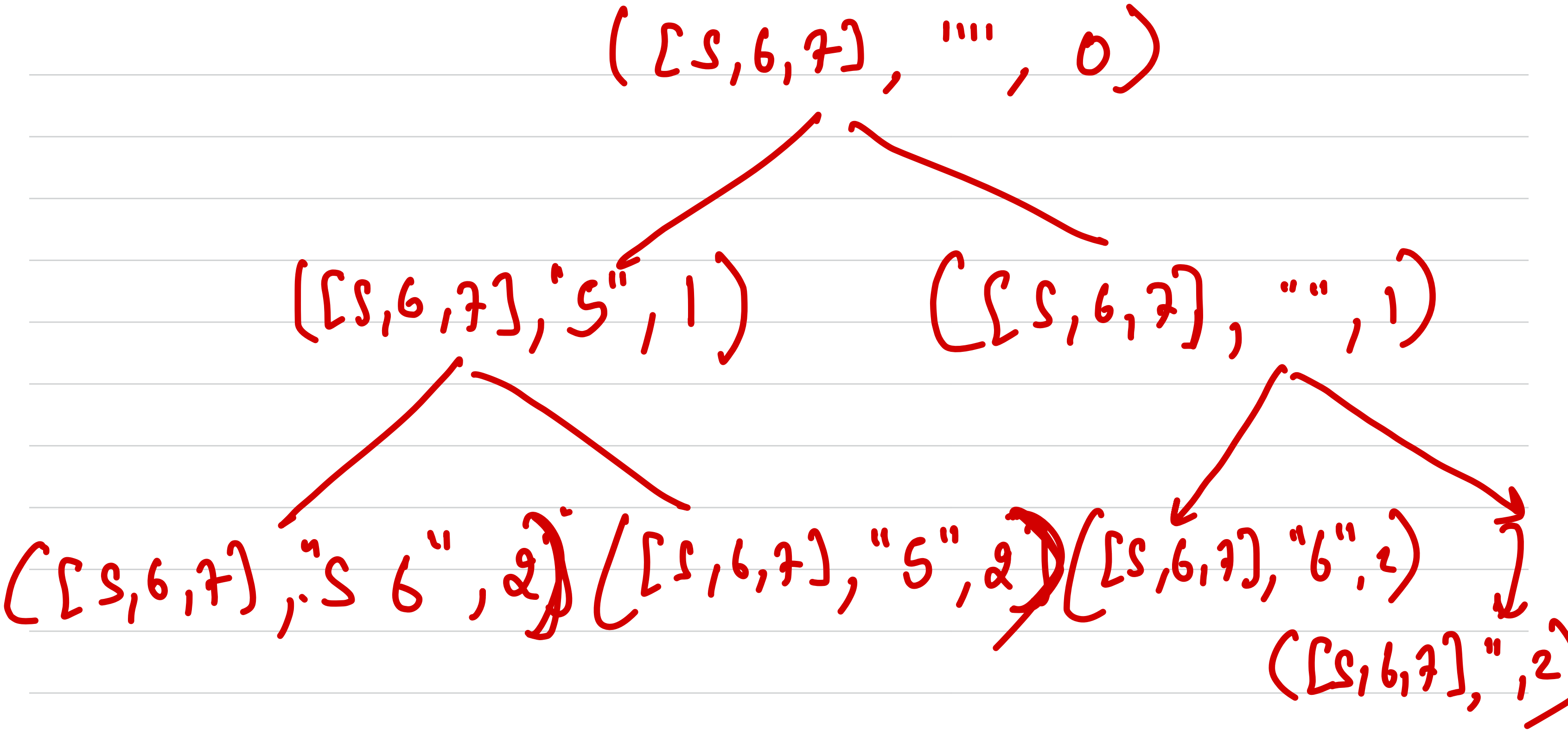
this function prints all the subsets of arr.

"5" + 7

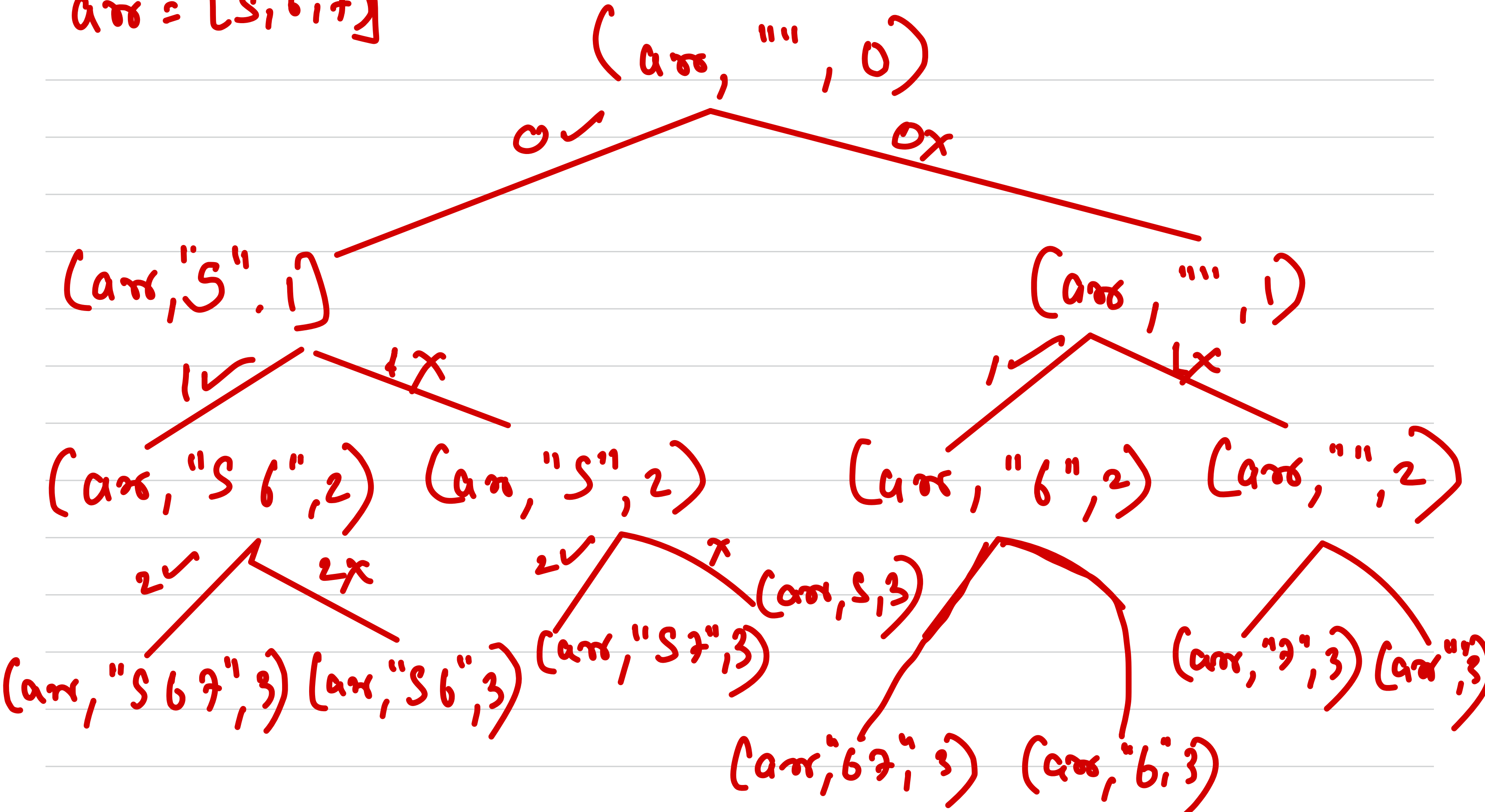
"57"

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

0 1 2


([5,6,7], "", 0)
([5,6,7], "5", 1)
([5,6,7], "", 1)
([5,6,7], "5 6", 2)
([5,6,7], "5", 2)
([5,6,7], "6", 2)
([5,6,7], "", 2)

arr = [5, 6, 7]


(arr, "", 0)
0 ✓
0 ✗
(arr, "5", 1)
(arr, "", 1)
1 ✓
1 ✗
(arr, "5 6", 2)
(arr, "5", 2)
(arr, "6", 2)
(arr, "", 2)
2 ✓
2 ✗
(arr, "5 6 7", 3)
(arr, "5 6", 3)
(arr, "5 7", 3)
(arr, 5, 3)
(arr, "6 7", 3)
(arr, "6", 3)
(arr, "7", 3)
(arr, "", 3)

Q: Given an array, check if it is sorted or not ?? (Recursively)

Sorted means arranged in ascending order or not

[1, 2, 3] → True

[5, 9, 2] → False

Try recursively

Qs Given a number n, you can do any 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 → 2

Say,

$f(n)$

$\downarrow$

returns the min

no. of ops reqd to

reduce $n \rightarrow 1$ with

our given ops

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

$$10 \xrightarrow{-1} 9 \xrightarrow{/3} 3 \xrightarrow{/3} 1 \quad \rightarrow \underline{\underline{3}}$$

~~Wrong Ans~~

Instead of reducing no. with any one op.

Try out all the possibilities.

Say,

$f(n)$ = $1 + \min \begin{cases} f(n/3) \\ f(n/2) \\ f(n-1) \end{cases}$

↓

returns the min no. of ops reqd to reduce $n \to 1$ with our given ops

Base Case

$$\left.\begin{array}{l} n == 1 \longrightarrow 0 \\ n == 2 \longrightarrow 1 \\ n == 3 \longrightarrow 1 \end{array}\right\}$$

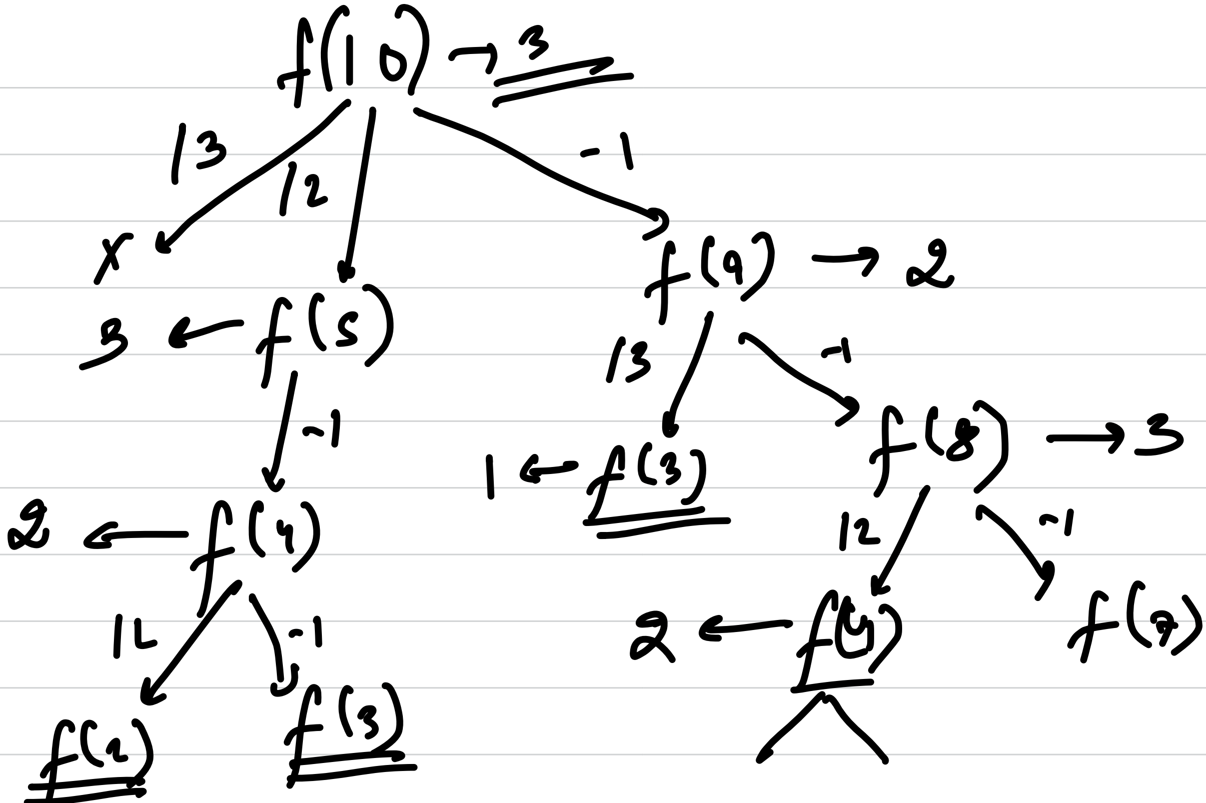
$$
\begin{tikzcd}
 & 18 \arrow[dl, "13"'] \arrow[d, "12"] \arrow[dr, "-1"] & \\
6 & 9 & 17
\end{tikzcd}
$$

$6 \to 1 \to x$

$9 \to 1 \to y$

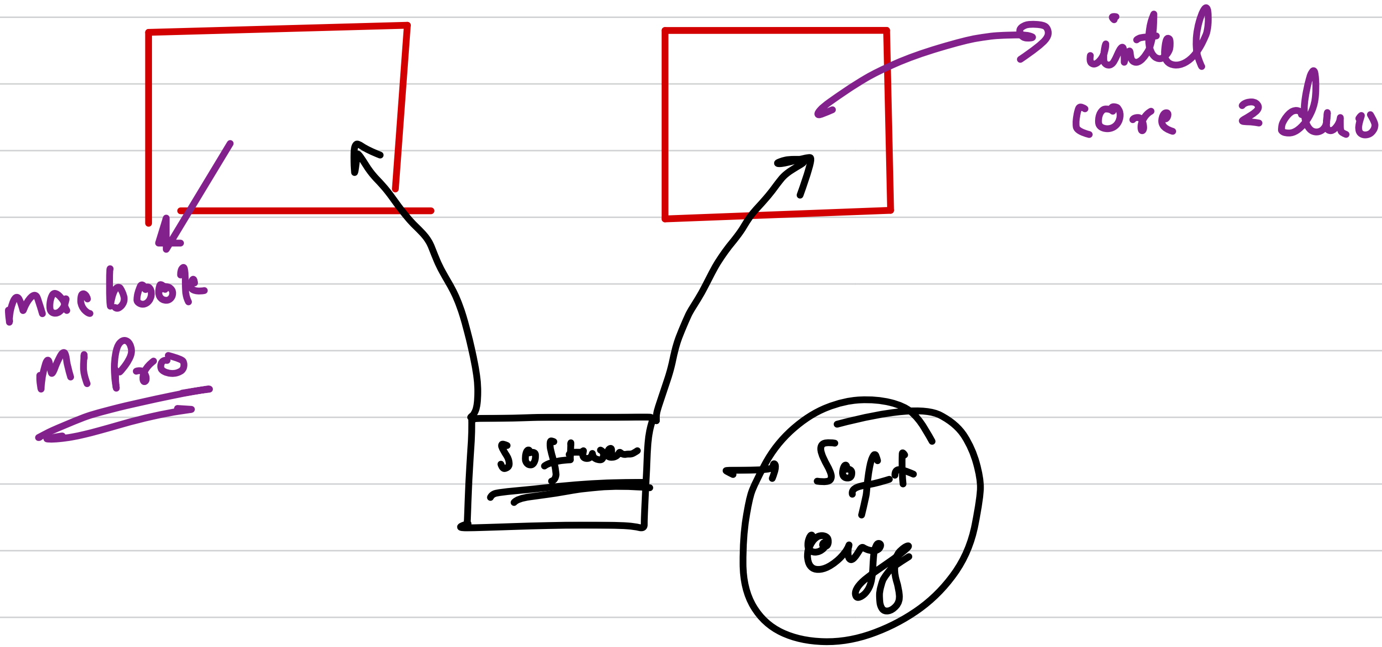
$17 \to 1 \to z$

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



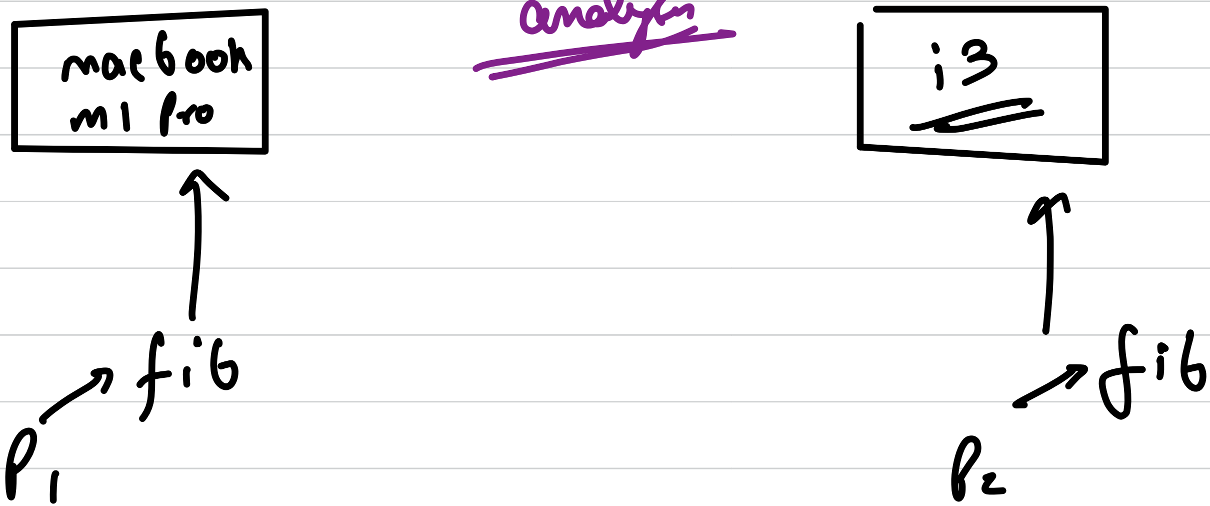
f(10) → 3
/3
/2
-1
x
f(5)
3
f(9) → 2
-1
/3
-1
f(4)
2
f(3)
1
f(8) → 3
/2
-1
f(2)
f(3)
f(4)
2
f(7)

$$\min(\underbrace{+\infty}, 3, 2)$$



intel
core 2duo
macbook
M1 Pro
software
Soft engg

Experimental analysis



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

Asymptotic Analysis

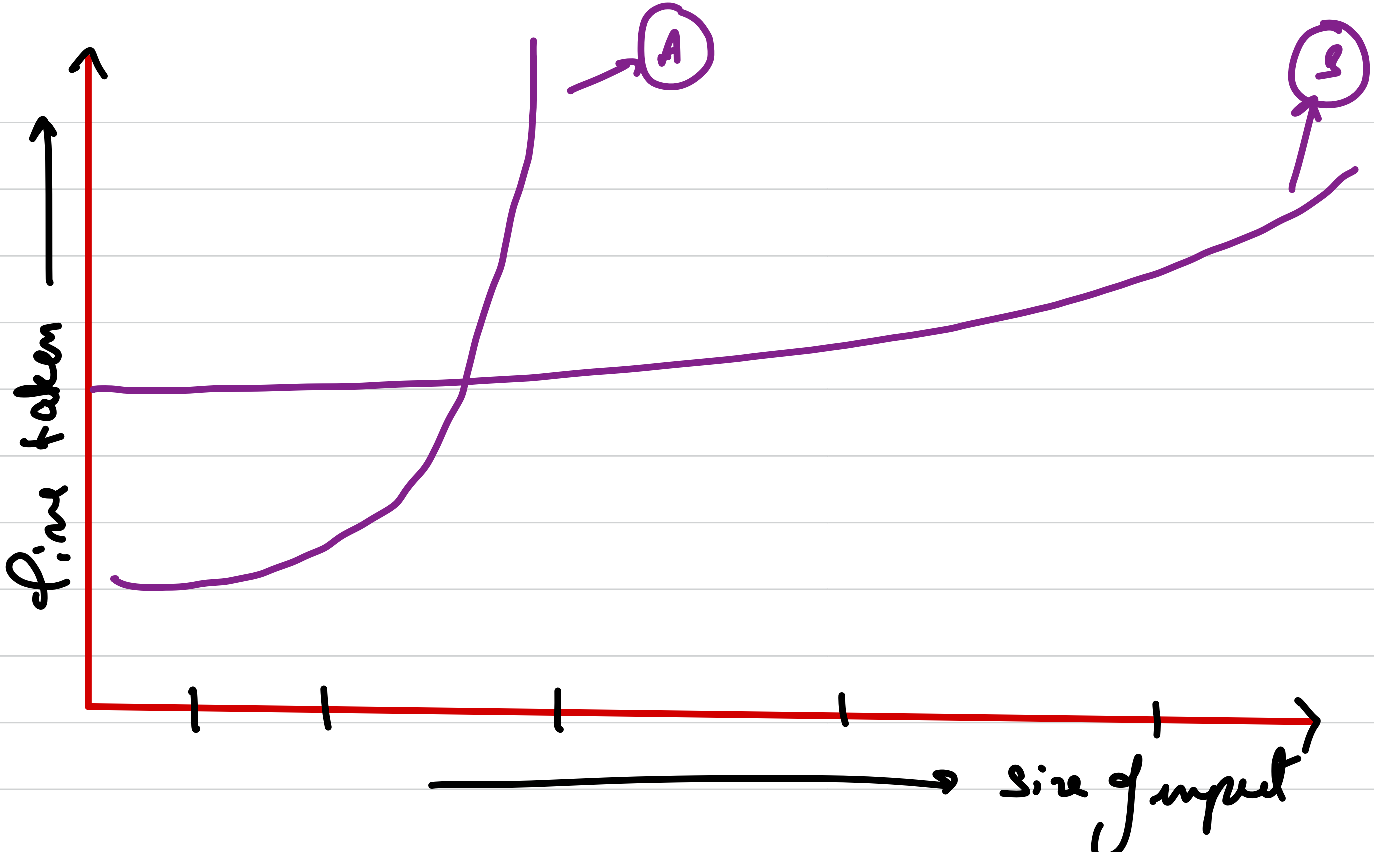
↳ input size dependent

Asymptote

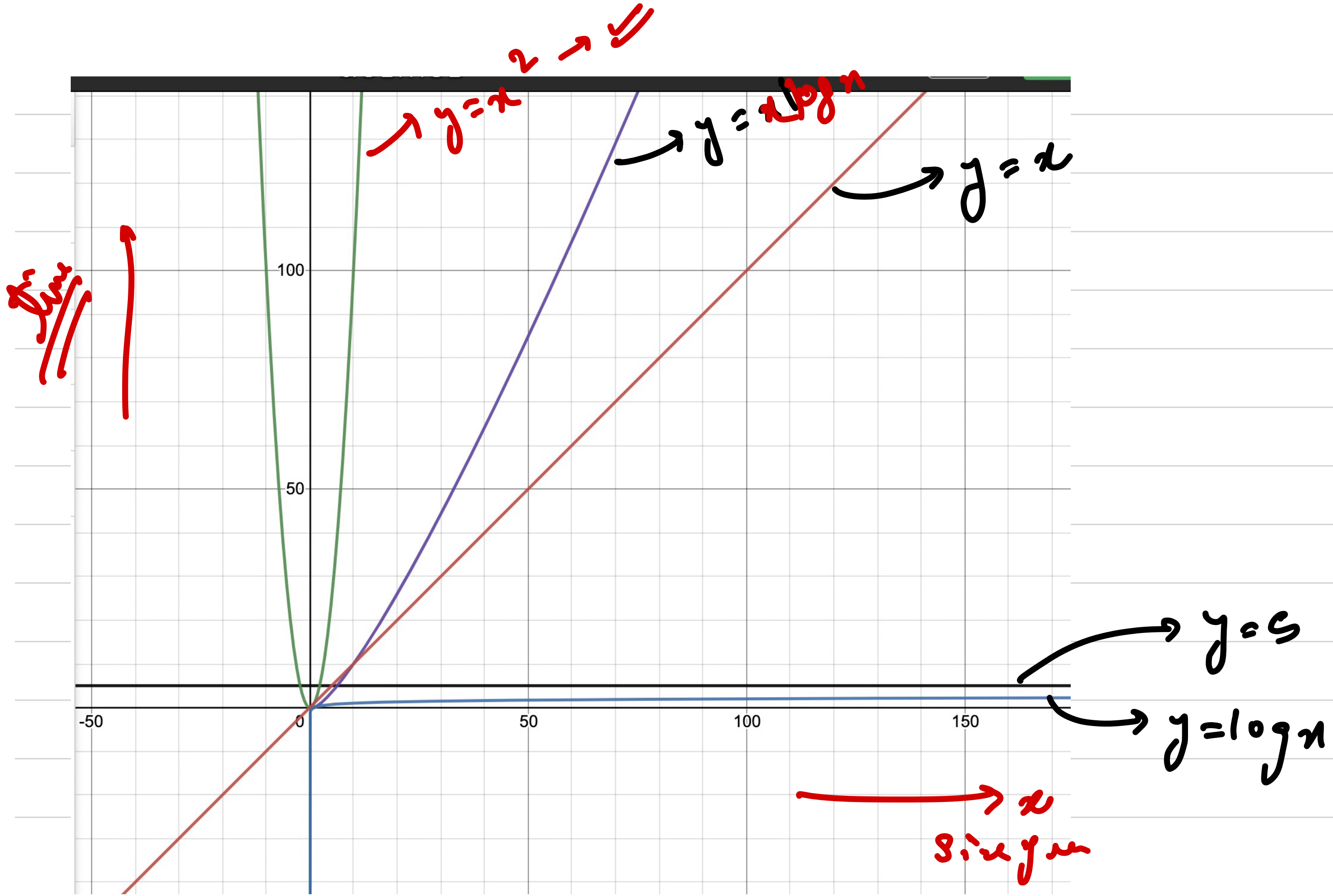
↳ On very big values of your input how the algo behaves is called asymptotic analysis

$2^3 \longrightarrow 8$ ✓✓

$\underline{\underline{2^{10^6}}} \longrightarrow$



The rate of growth of B is less



100
50
-50
0
50
100
150
$y = x^2$
$y = x\log x$
$y = x$
$y = 5$
$y = \log x$
x

$\rightarrow \quad y = x^2 \longrightarrow \dfrac{dy}{dx} \rightarrow 2x$

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

$y = x \quad \dfrac{dy}{dx} \rightarrow 1$

$y = \log x \quad \dfrac{dy}{dx} \rightarrow 1/x$

$y = 5 \quad \dfrac{dy}{dx} \rightarrow 0$

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

$\rightarrow \approx 20$

$\cancel{y = S}$

$x = 10^6$

Rate of change in growth

$x!$

2^x

$\vdots$

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);
}
```

A

$\underline{\underline{x}}$

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

B

$\underline{\underline{x^2}}$

1 (1 − x) $\underline{x}$

2 ⟶ $\underline{x}$

3 ⟶ $\underline{x}$

4 ⟶ $\underline{x}$

⋮ ⋮

x ⟶ $\underline{x}$

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

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 highest degree term only

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

$x \rightarrow 10^9$

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

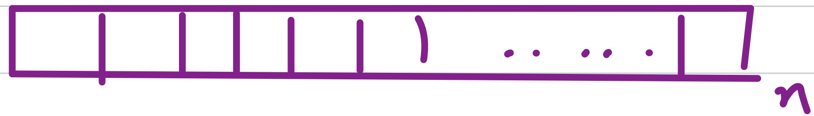
$\Rightarrow$

3 major Cases of algo analysis

1) Worst Case → Priority highest

2) Avg Case

3) Best Case



$n \to 10^6$

element = x

Best Case → $\Omega(1)$

Worst Case → $O(n)$

Avg Case → $\Theta(c \times n) \Rightarrow \Theta(n)$

3 Cases → 3 notations

1) Worst Case → Big O notation — Oh — tightest upper bound

2) Avg Case → Big Θ notation — theta — avg time

3) Best Case → Big Ω notation — omega — tightest lower bound

$i = 1$

$i = 2$

$i = 4$

8

16

32

$\vdots$

2^k

$\underbrace{\qquad}_{k \text{ operation}}$

k operation

$k \approx \log_2 n$

$2^k > n$

$\log_2 2^k > \log_2 n$

$k \log_2 2 > \log_2 n$

$\boxed{k > \log_2 n}$

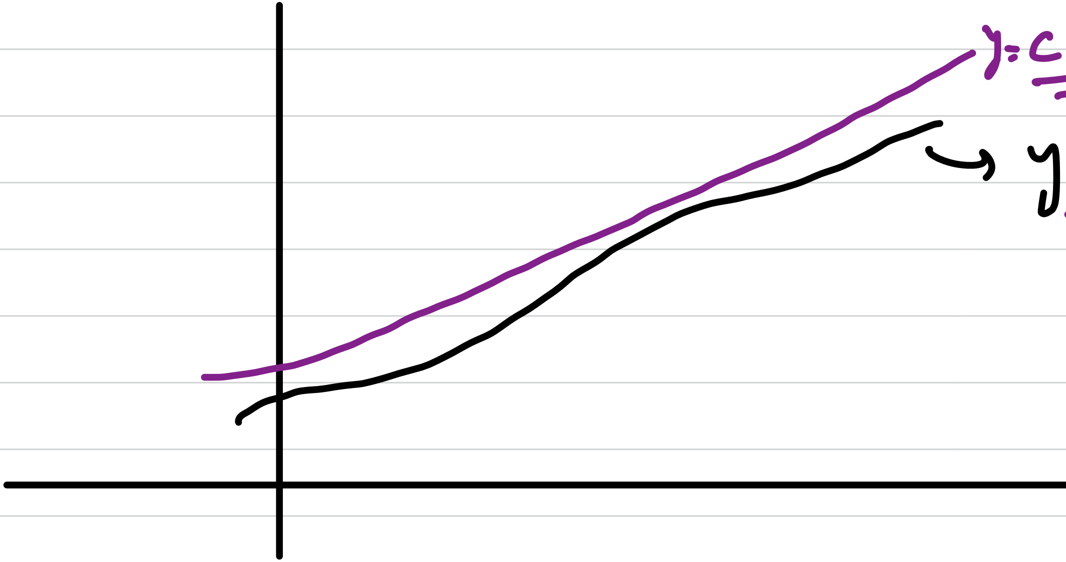
$$\log_3 x = \frac{\log_2 x}{\log_2 3} \rightarrow \text{last}$$

$$i * i \le n$$

$$i^2 \le n$$

$$i \le \sqrt{n}$$

for (i = 1 ; i ≤ √n ; i++)



$y = c \times x^2$

$c \geq 3$

$y = 2x^2 + 10x$

$O(cx^2)$

$\approx O(x^2)$

Space Complexity

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

$\underline{\underline{n}} = 10^5$

```
function sum(arr) {
    let ans = 0;
    for (let i = 0; i < arr.length; i++) {
        ans += arr[i];
    }
    return ans;
}
```

recursion → Call stack → will be considered in space Complexity