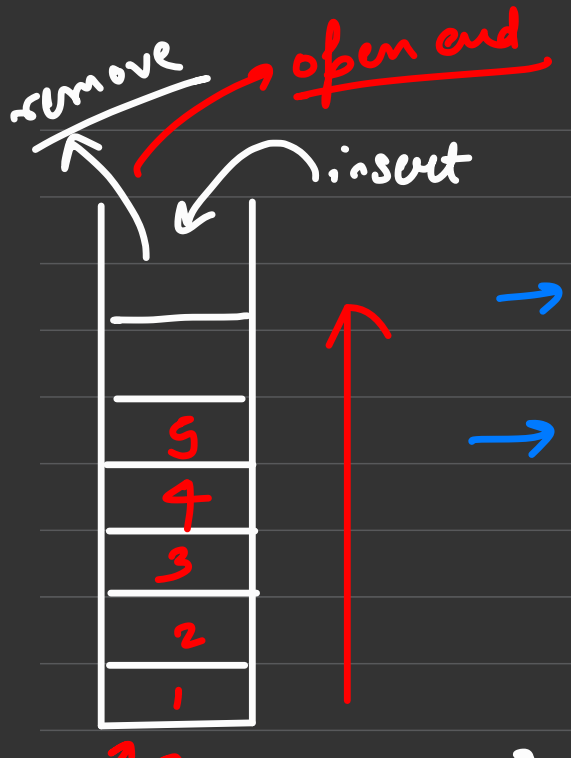
STACKS



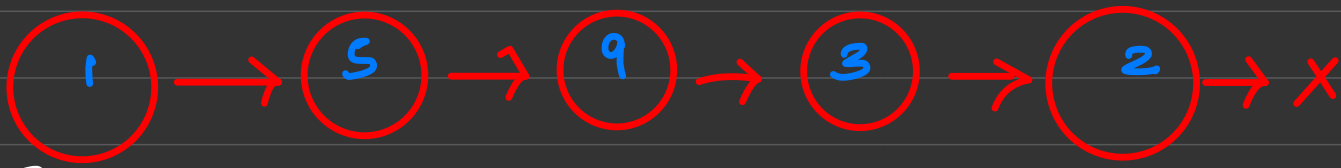
→ Stack is a linear data struc-

→ Stacks Support LIFO

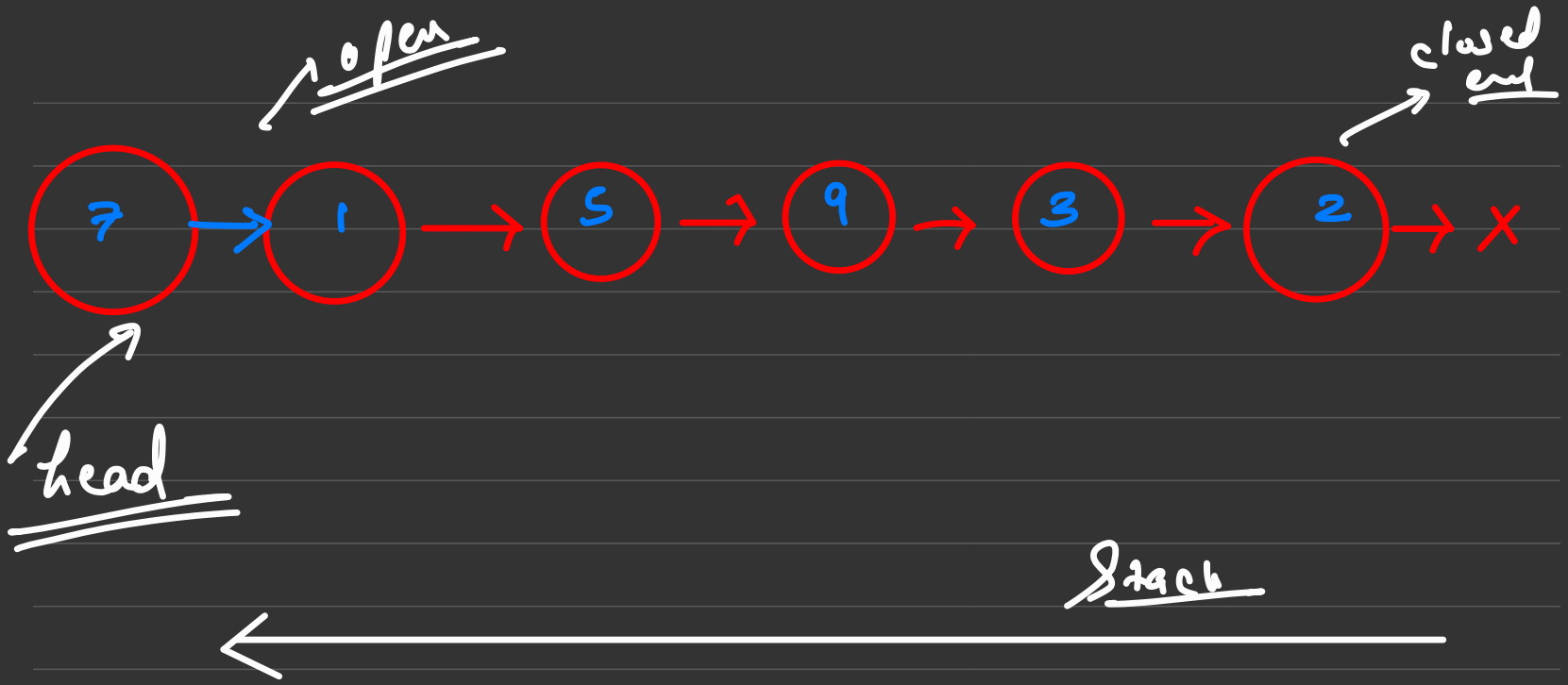
Last In first Out

→ Stack does not allow access of any element apart from Topmost element.

Stacks Using LL



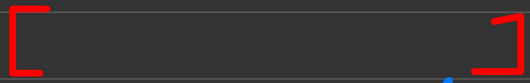
head → Insert at head
remove at head
] → O(1) time



push → insert At Head

pop → remove At Head

Stacks Using Arrays



↳ adding / removal at the last of an array is very efficient.

push(10)

push(12)

push(3)

pop()

[10, 12,]

last index

arr

arr.push

arr.pop

Valid Parenthesis

(⟷) { ⟷ } [⟷]

(()) → Balanced

()() → Balanced

()) → Not Balanced

Properties of Valid strings :

→ if $S_1, S_2 \rightarrow$ is a valid string

① $S_1 + S_2 \rightarrow$ also valid

if we concatenate 2 valid strings then the result is also valid :

Ex $S_1 \rightarrow ()()$

$S_2 \rightarrow [()]\{\}$

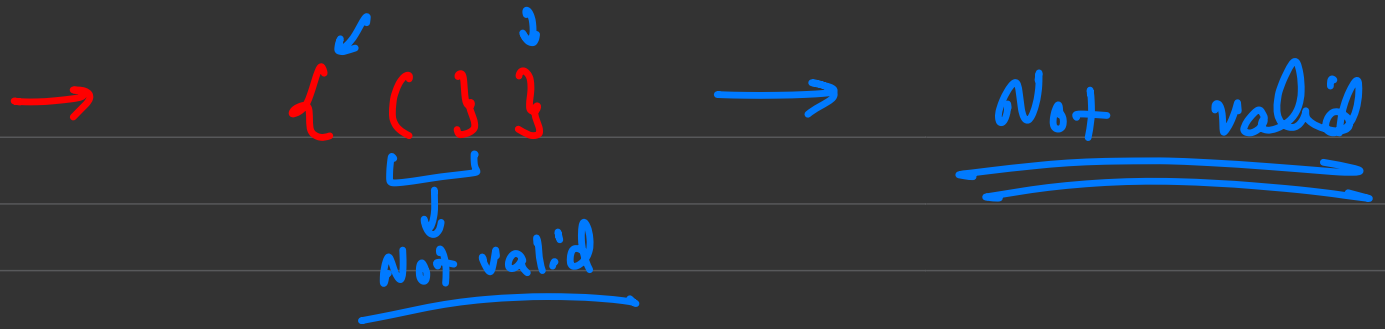
$S_1 + S_2 \rightarrow ()()[()]\{\} \rightarrow$ valid

if S is a valid string

(2) (S) or {S} or [S] → valid

Ex → S → (())

(S) → ((())) }
{S} → {(())} } valid
[S] → [(())] }



③ if any part / substring of the string is invalid then whole string is invalid.

The smallest possible valid strings are

→ ()

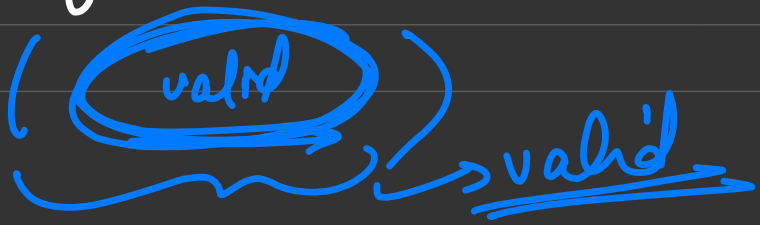
→ [ ]

→ { }

And any big string is generated by applying the few properties on these three.

→ Let's say we encounter an opening parenthesis / brace / bracket,

then we can check that for this opening when do we get a closing. If in between the opening & closing the string is valid then whole String is valid.



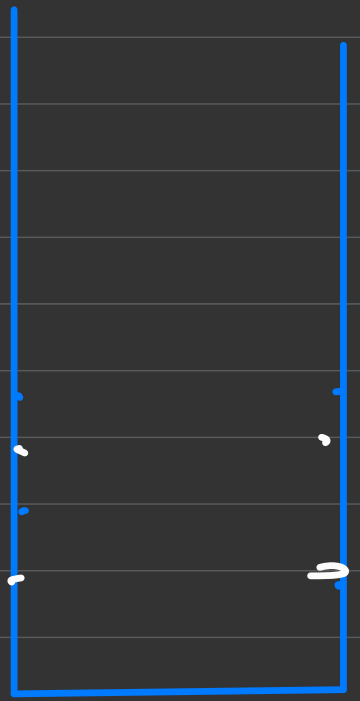
In this case Stacks can help us.

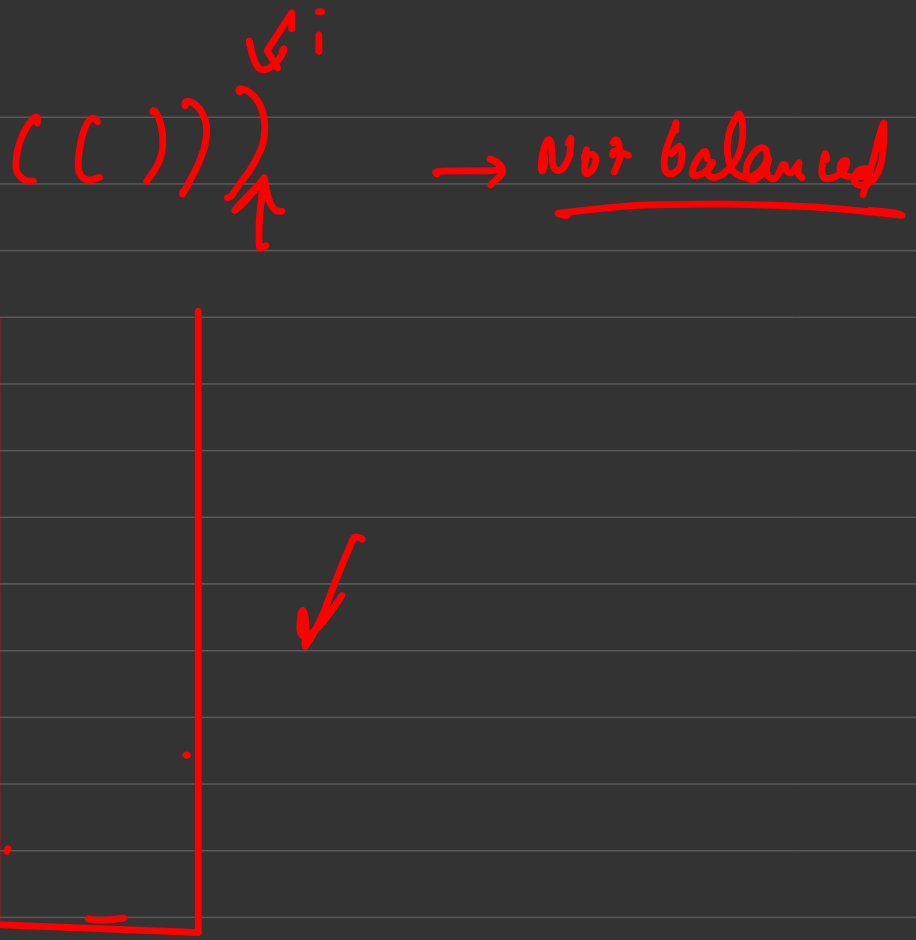
→ Let's try to remember / store an opening bracket / brace / parenthesis. So that when a closing one comes we can check if we have a counterpart opening for it or not.

i

(([ ])) { { } }

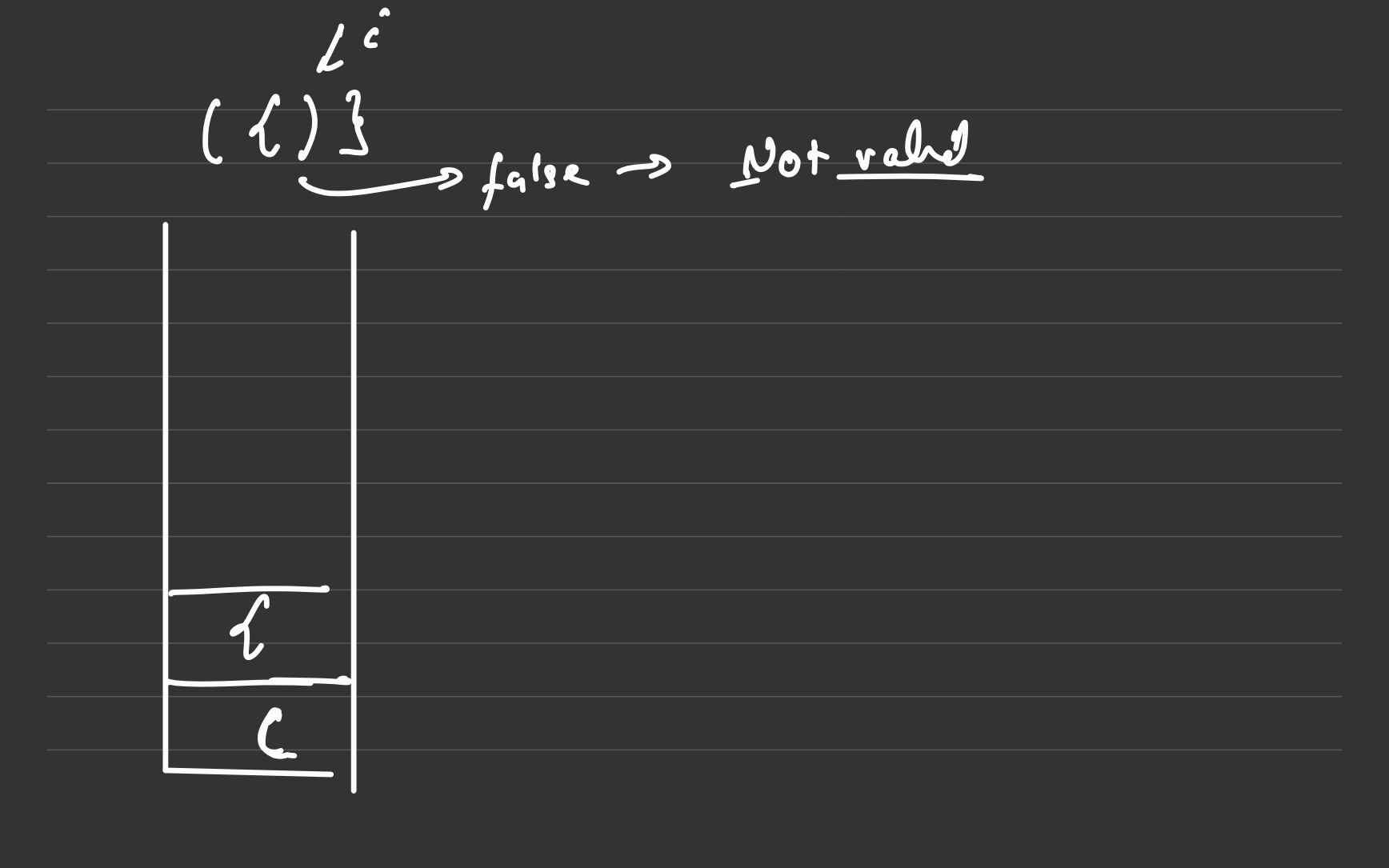
The moment we detect the smallest valid string we can neutralize it.




(( ))
i
→ Not balanced

)(<i

Not valid


i
({) }
false → Not valid
{
(