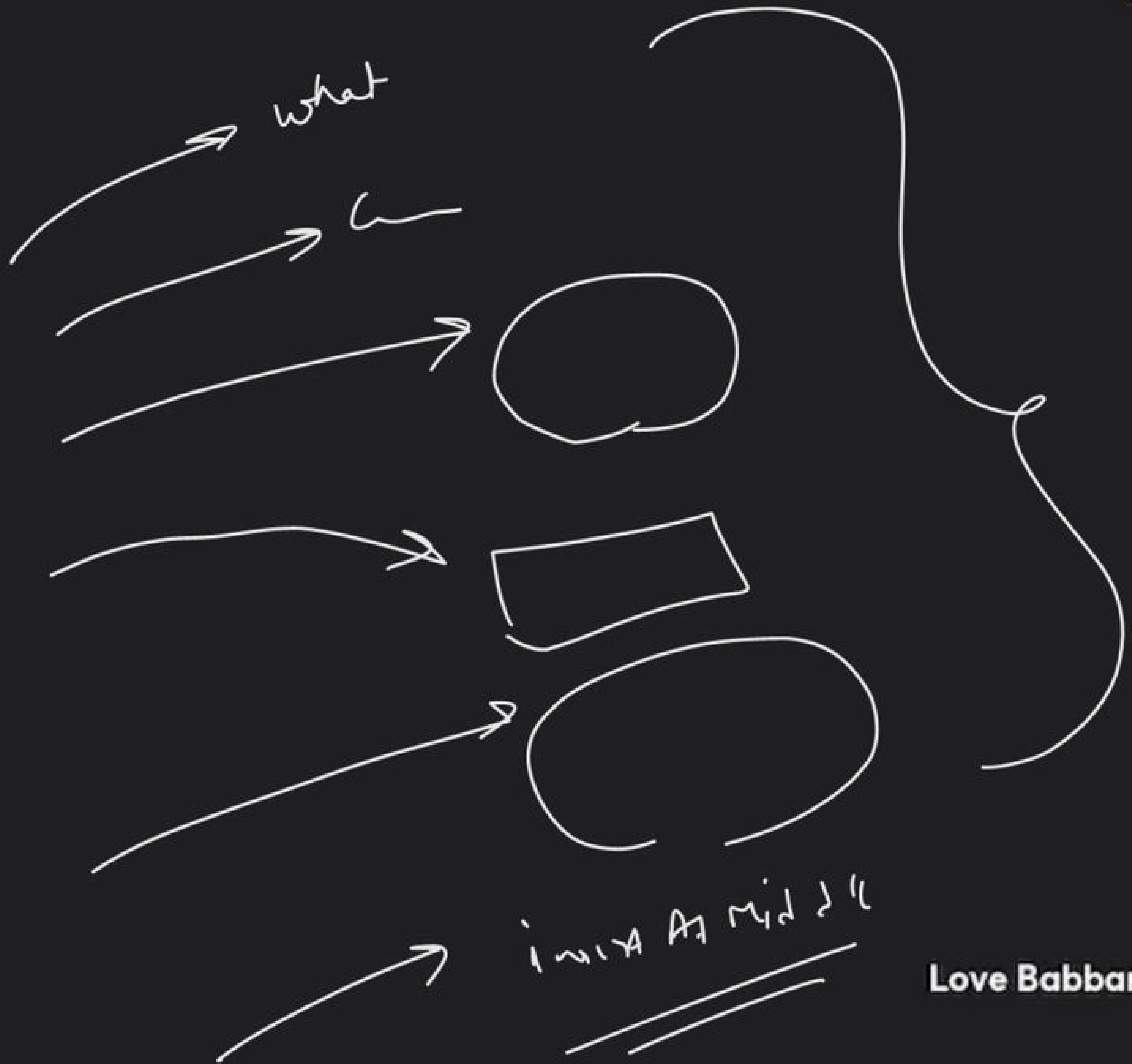
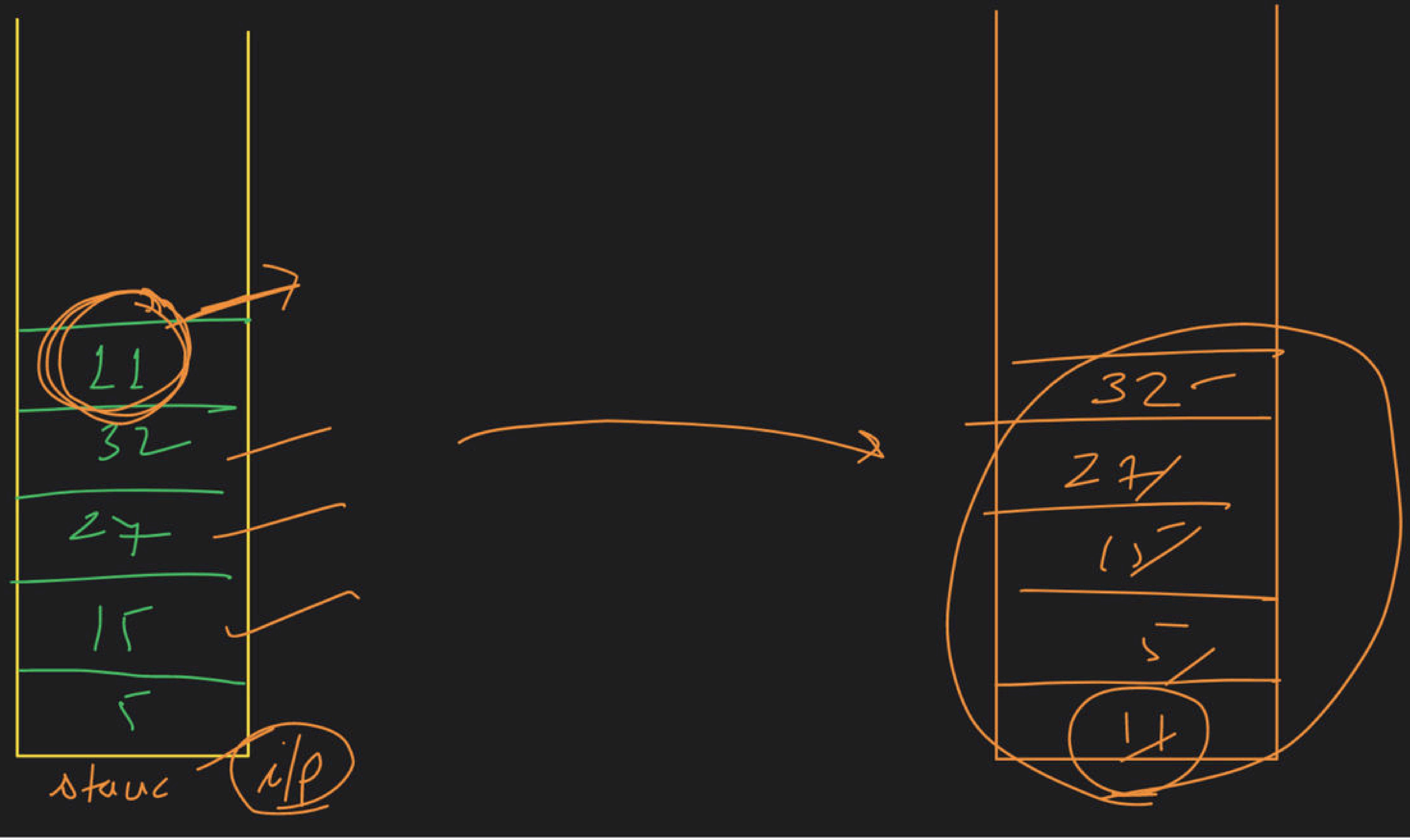
Stack Class - 2

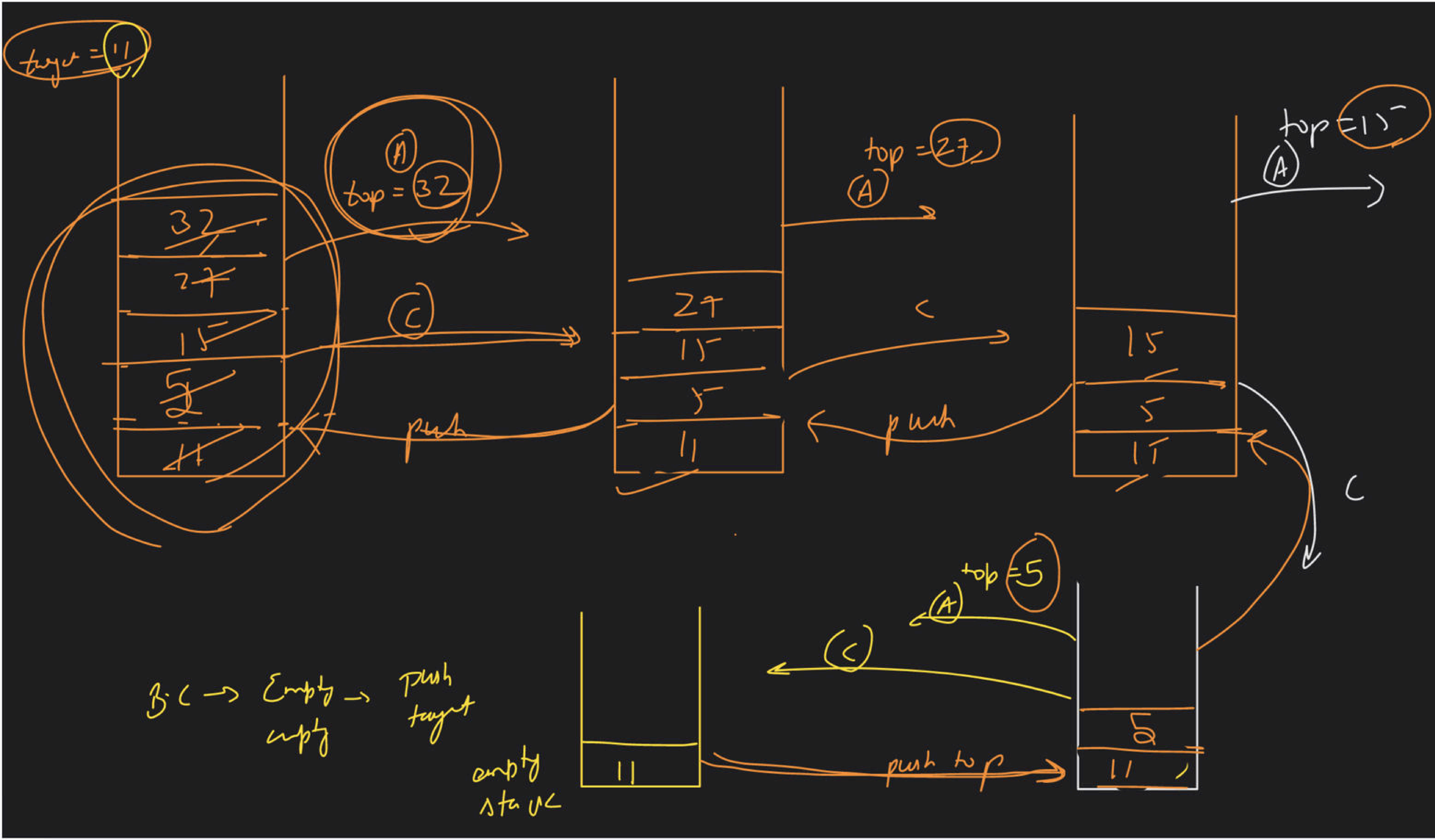
Special class



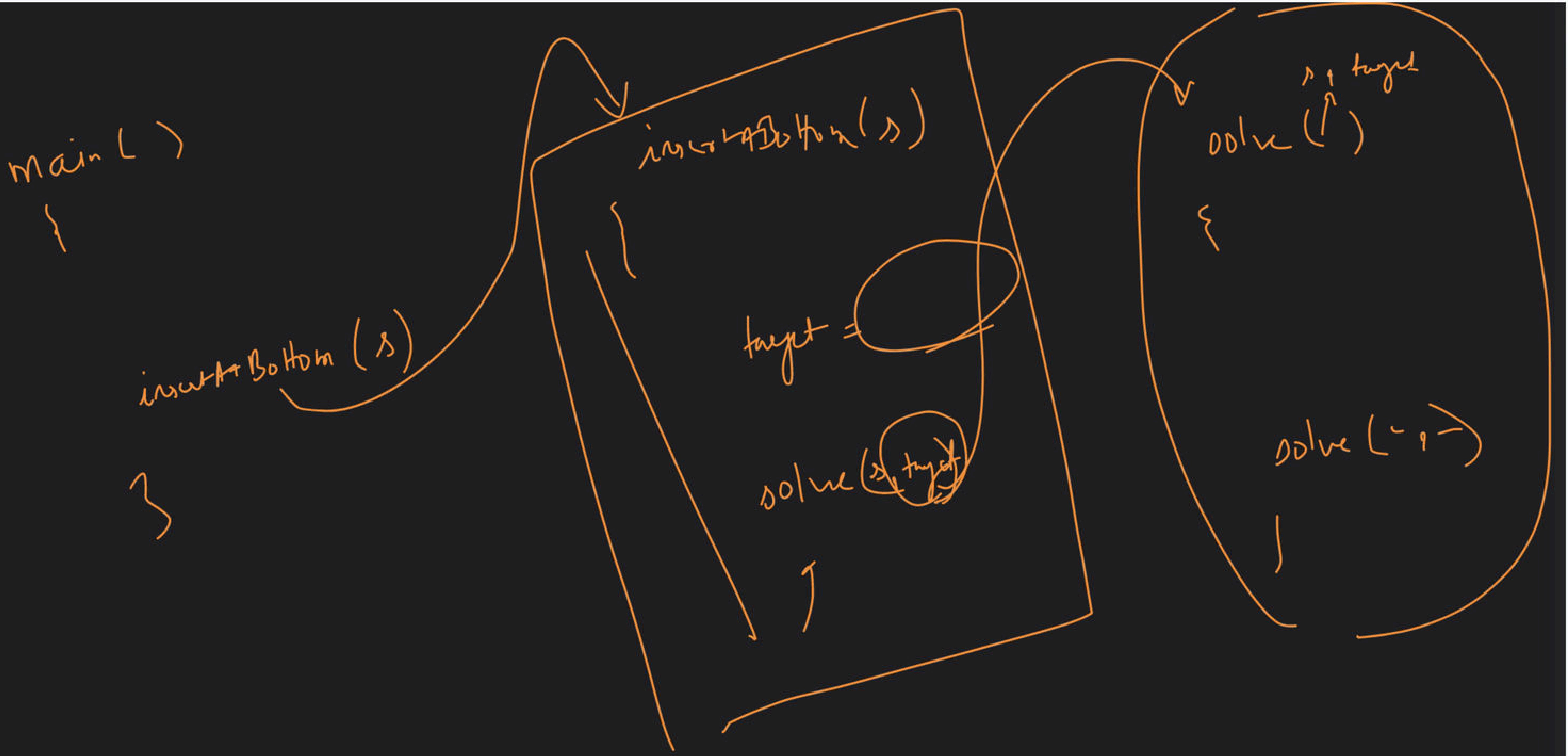
Love Babbar • Apr 26, 2023

insert At Bottom:-



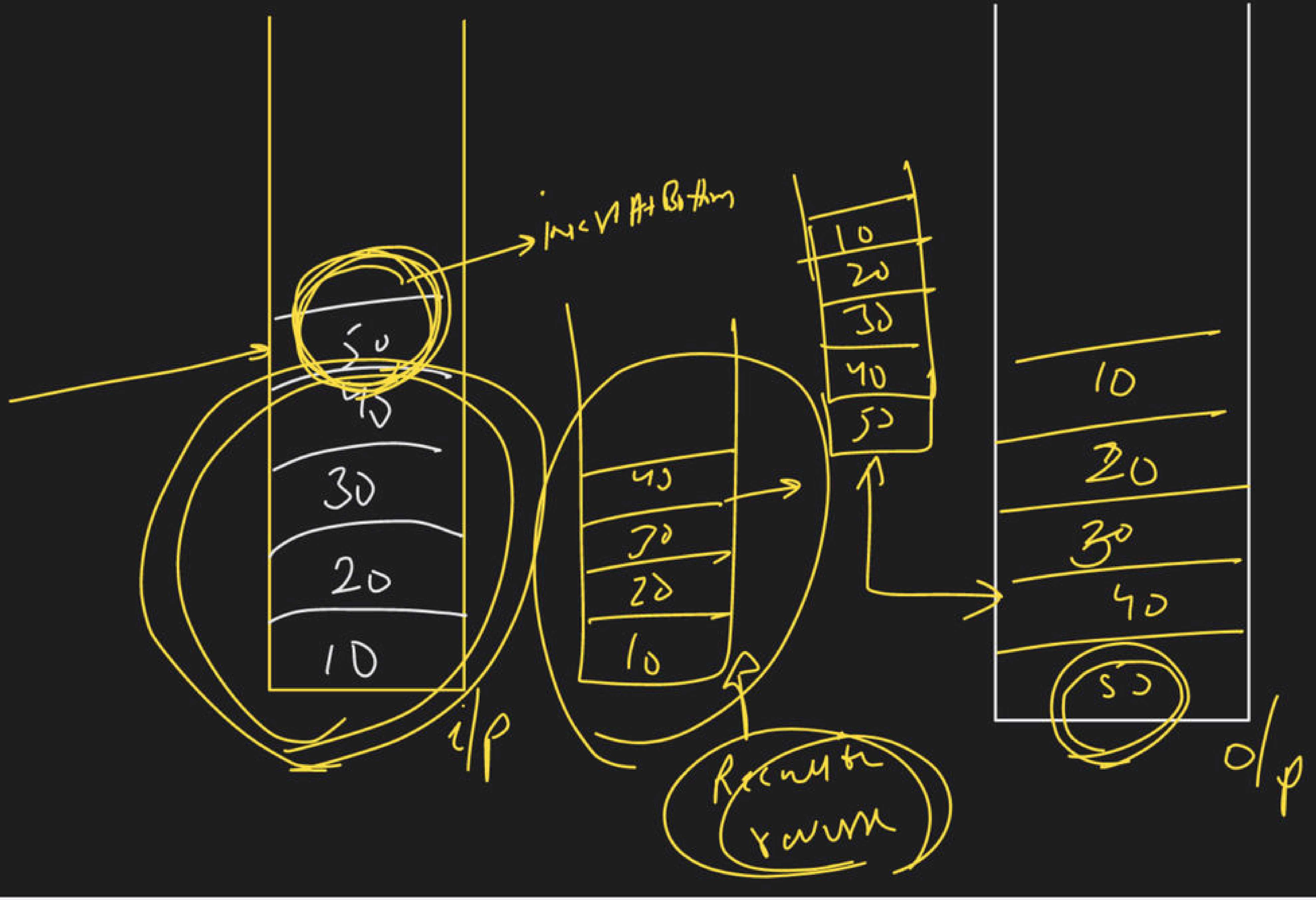


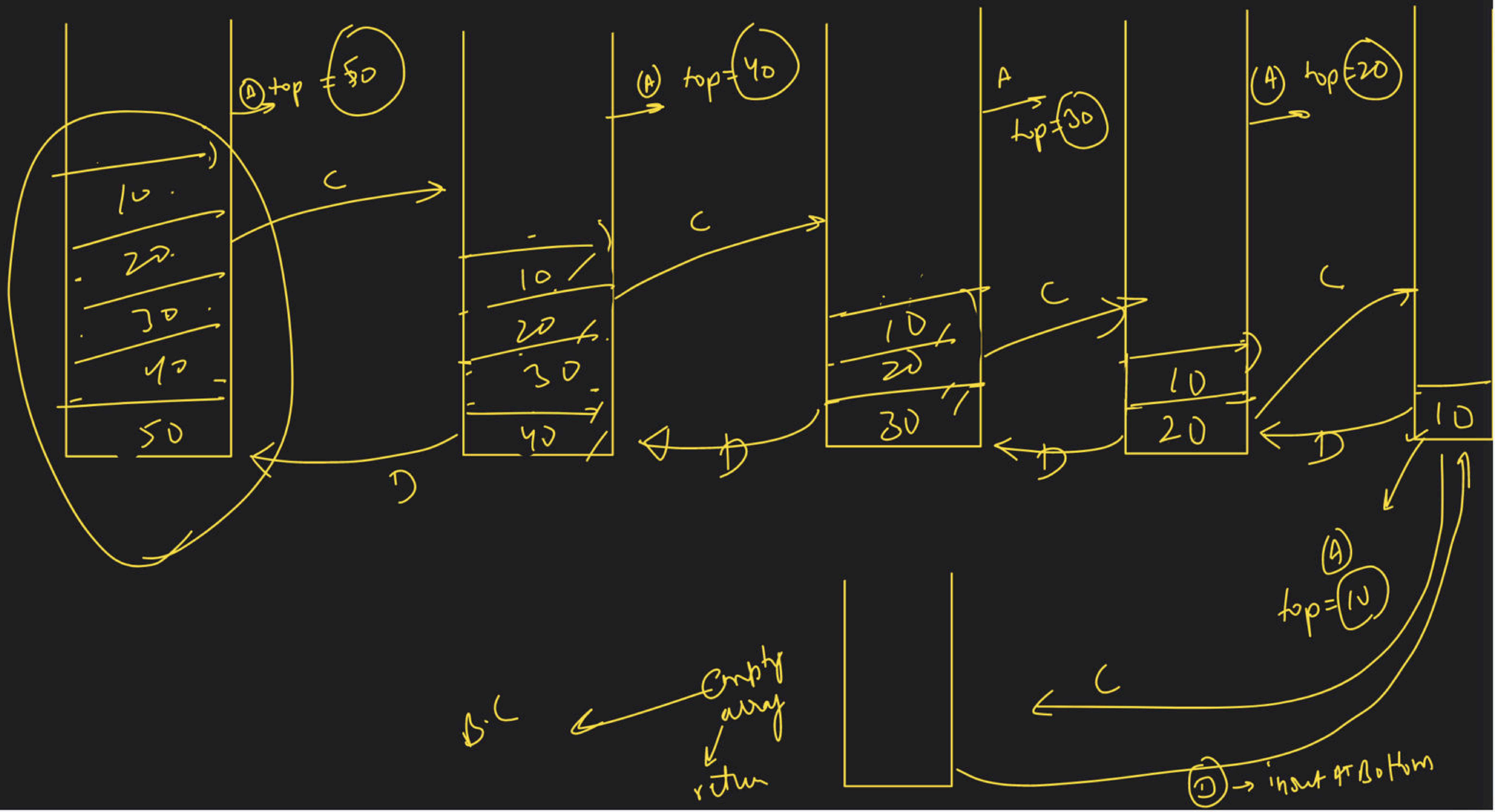
B.C → Empty → push target
empty


main ( )
{
insertAtBottom (s)
}
insertAtBottom (s)
{
target =
solve (s, target)
}
solve ( )
s , target
{
solve (_ , _)
}

→ Reverse a Stack

2 min





(1) top = 50
10
20
30
40
50
C
D
(A) top = 40
10
20
30
40
C
D
A
top = 30
10
20
30
C
D
(A) top = 20
10
20
C
D
10
(A)
top = 10
C
(1) → insert at Bottom
Empty array
return
B.C

Algo

A top =

B pop

C recursive call

↳ reverse stack

4 insertAtBottom

↳ top

→ Valid Parenthesis → Bracket

→ exp →

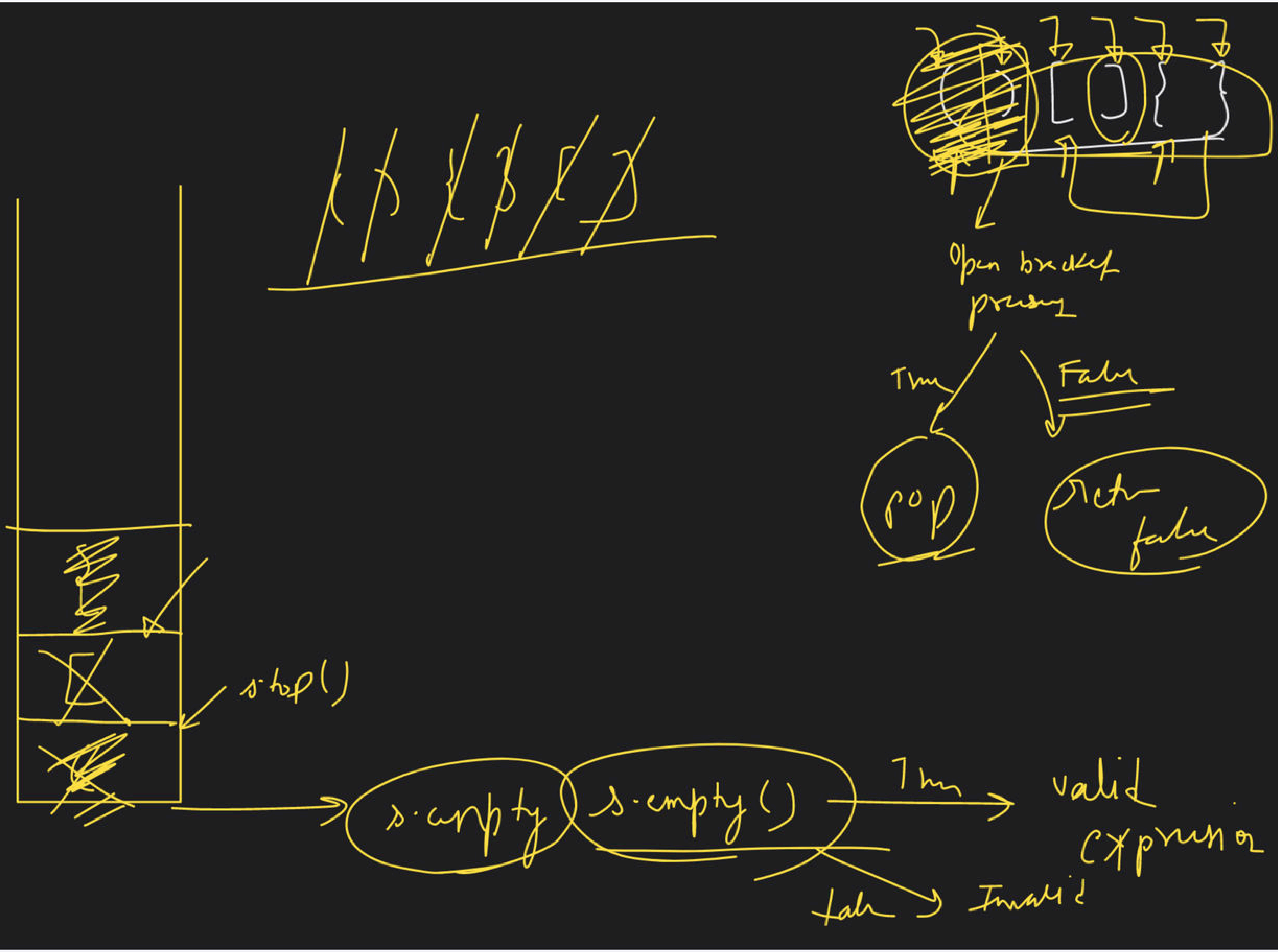
( )

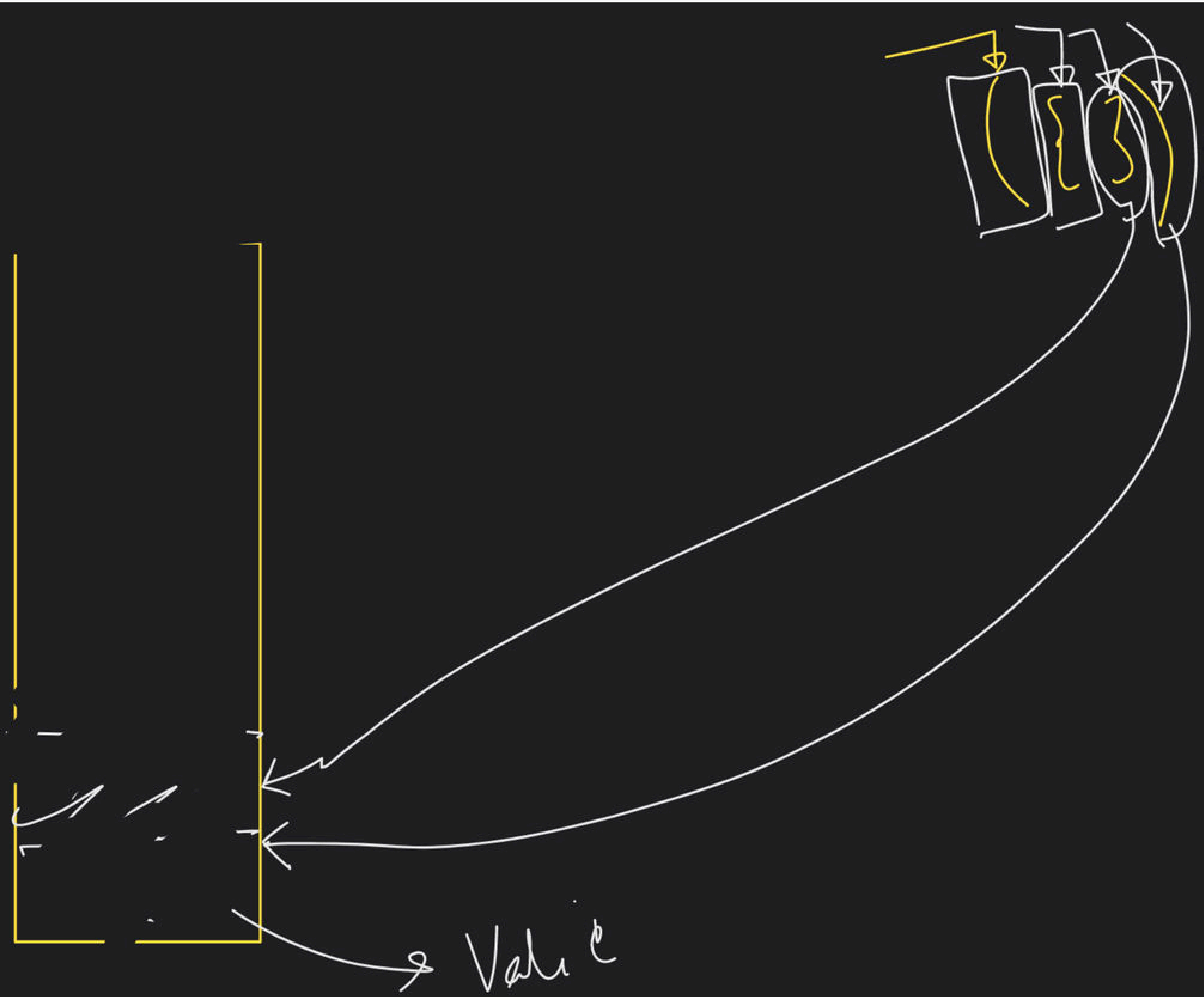
{ }

[ ]

valid → ( ) { } [ ]

1. ([ ]) → true
2. ([{ }]) → true
3. ([} {]) → false
4. } { → false

( ) [( )] { }
( ) { } [ ]
Open bracket present
True
False
pop
return false
s.top()
s.empty
s.empty()
True
valid expression
false
Invalid


({ })
Valid

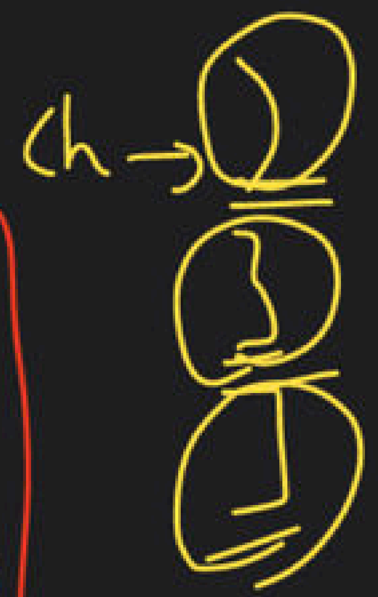
```
stack<char> st;

for (int i=0; i<s.length(); i++)
{
    char ch = s[i];
    if (ch=='(' || ch=='{' || ch=='[')
    {
        st.push(ch);
    }
    else {
        if(!st.empty())
        {
            → logic
        }
    }
    else → st → empty
    {
        return false;
    }
}
```

ch → opening Bracket

ch → closing bracket

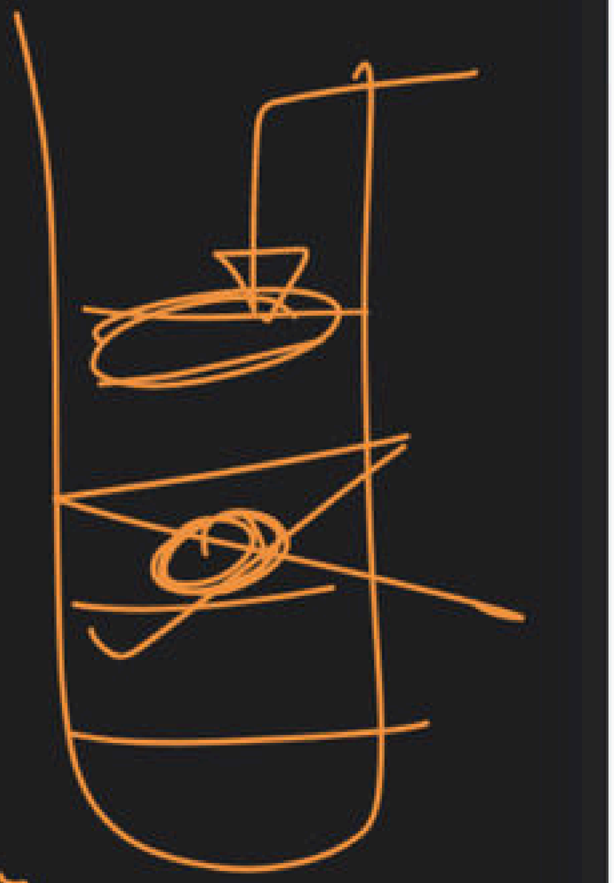
if(st.empty())
↳ valid → return true;

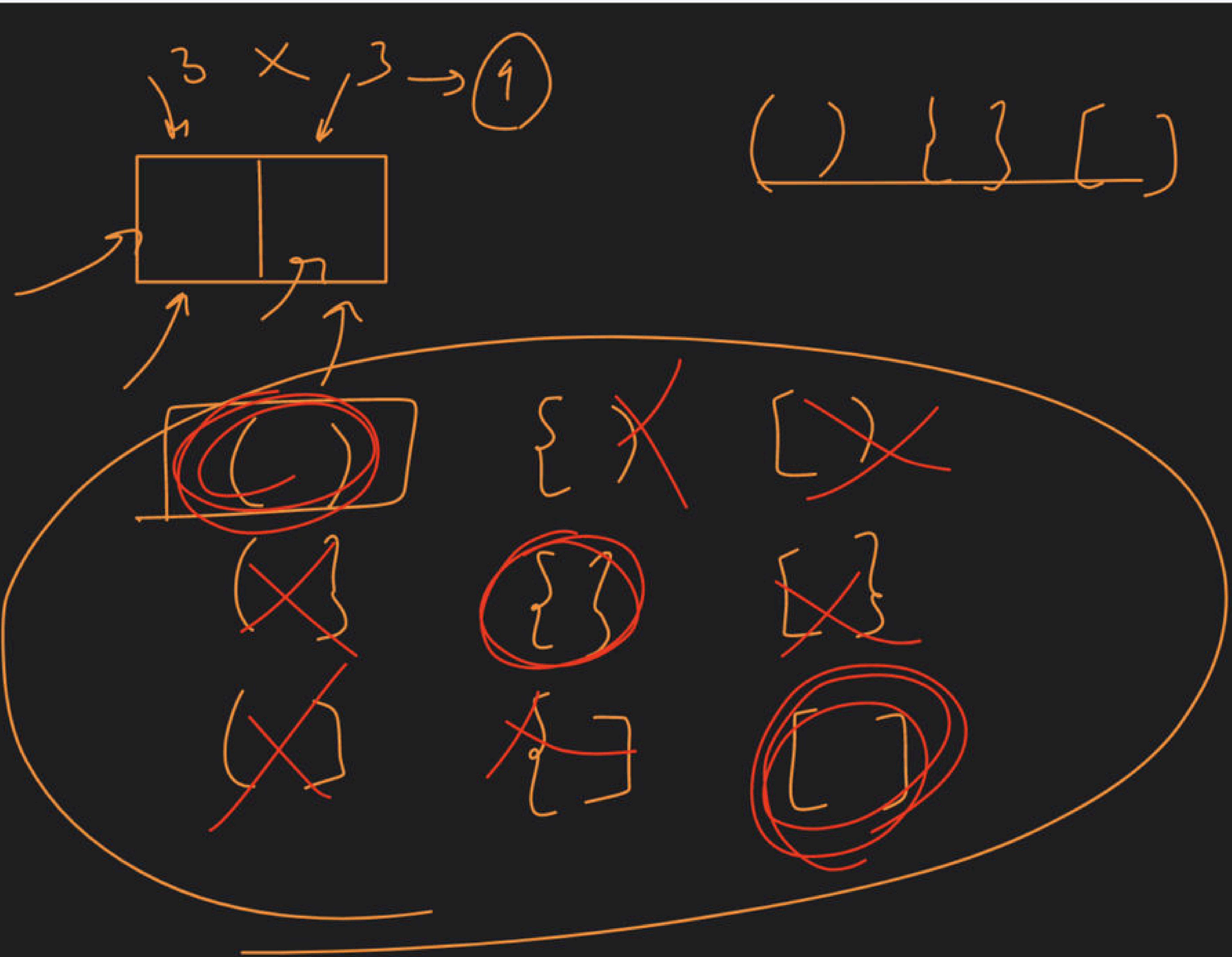
else
↳ invalid → return false



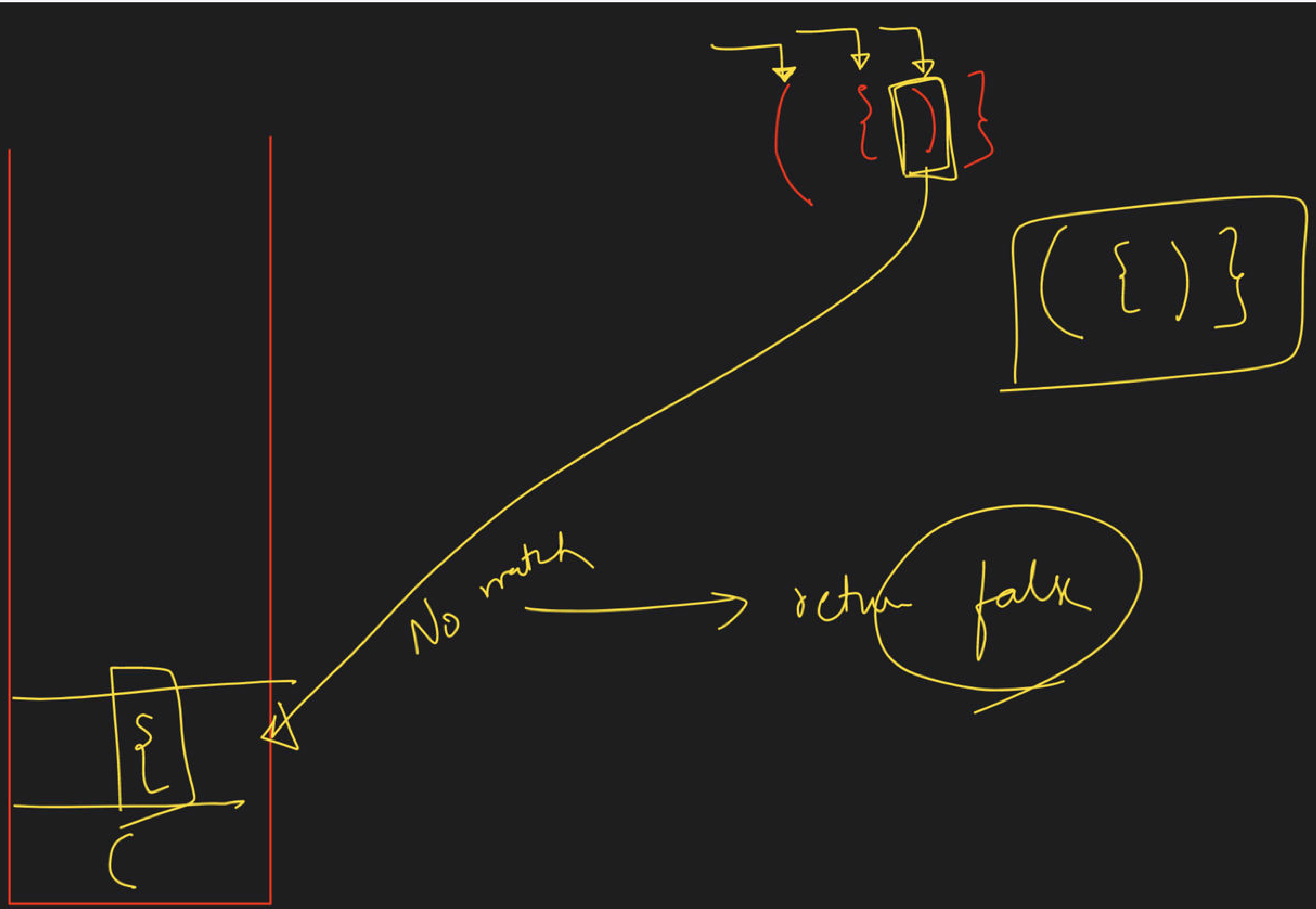
~~tag~~ Logic → int topch = st.top()

```
if (ch == ')' && topch == '(' )  → st.pop
else if (ch == '}' && top4h == '{' )  → st.pop()
else if (ch == ']' && top == '[' )  → st.pop()
                                              matching
else → not matching
{
    return false
}
```





3 × 3 → 9
( ) { } [ ]
( )
{)
[ )
(}
{ }
[}
( ]
{]
[ ]


({) }
({) }
{
(
No match
return false

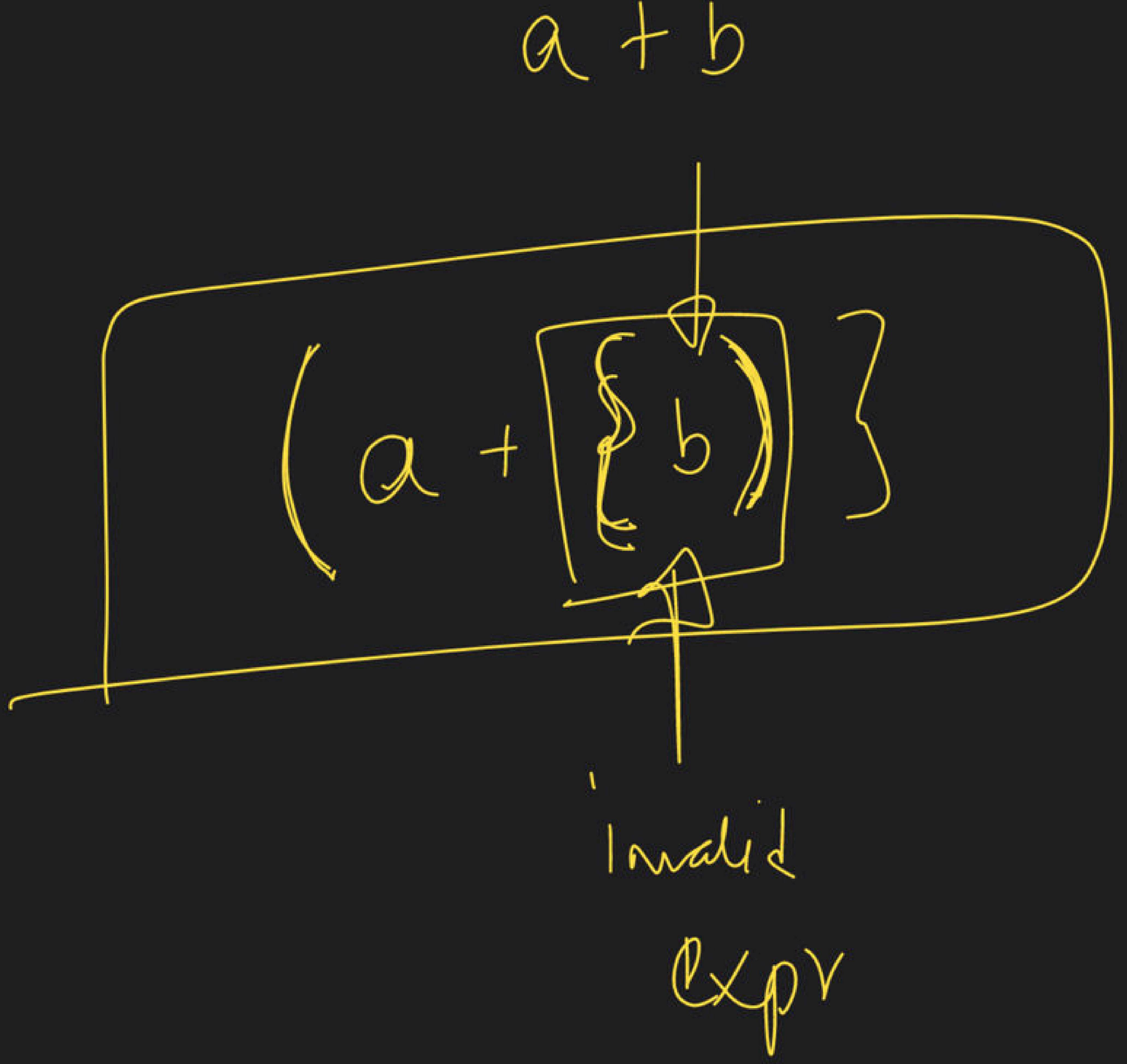
valid

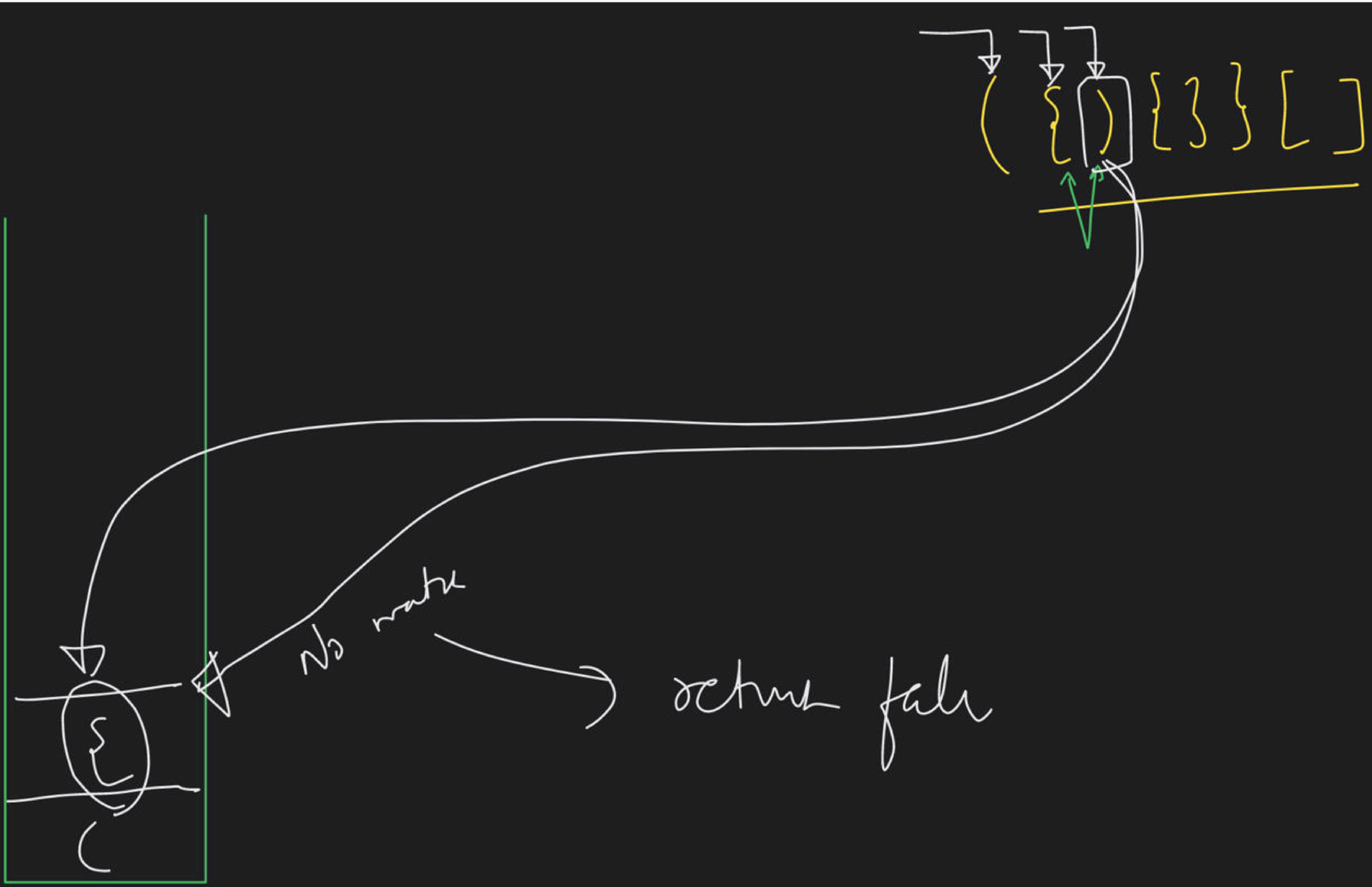
( )

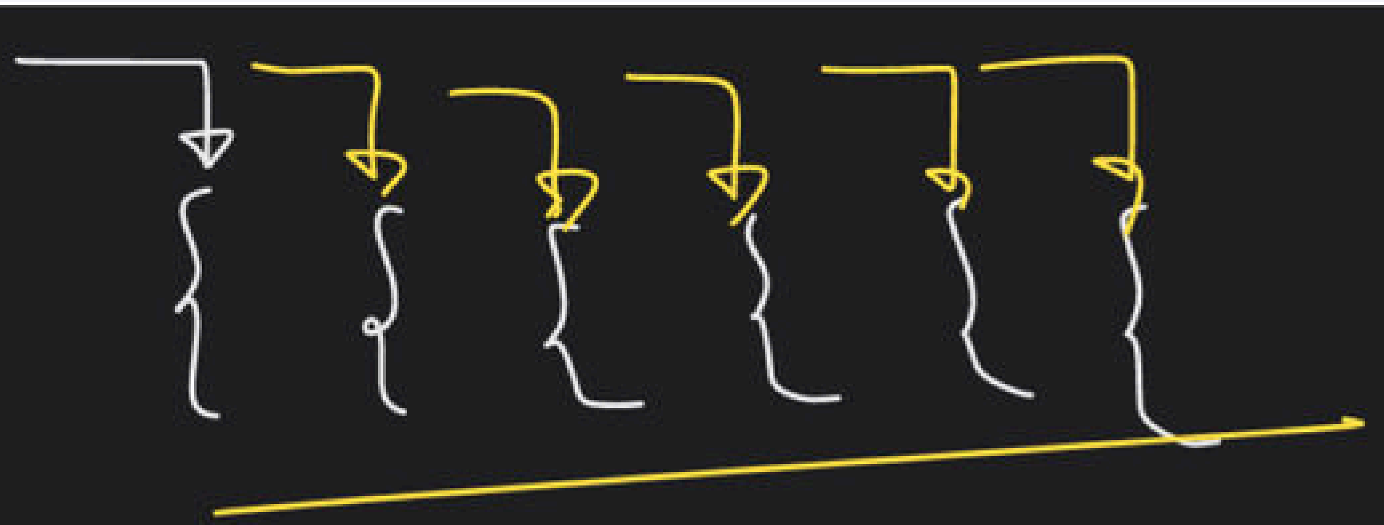
( )

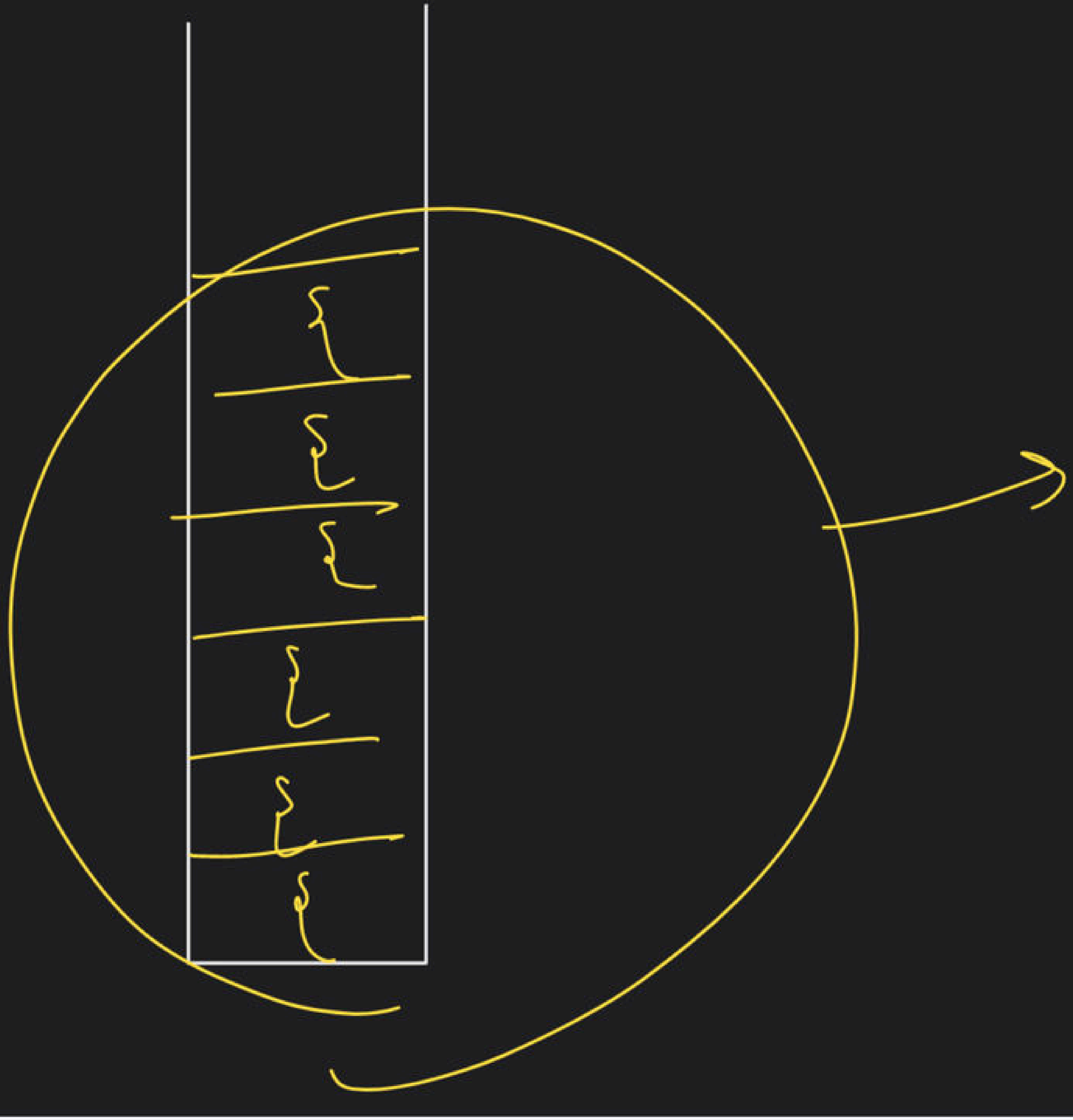
{ }

[ ]




({) { } } [ ]
{
(
No match
return false

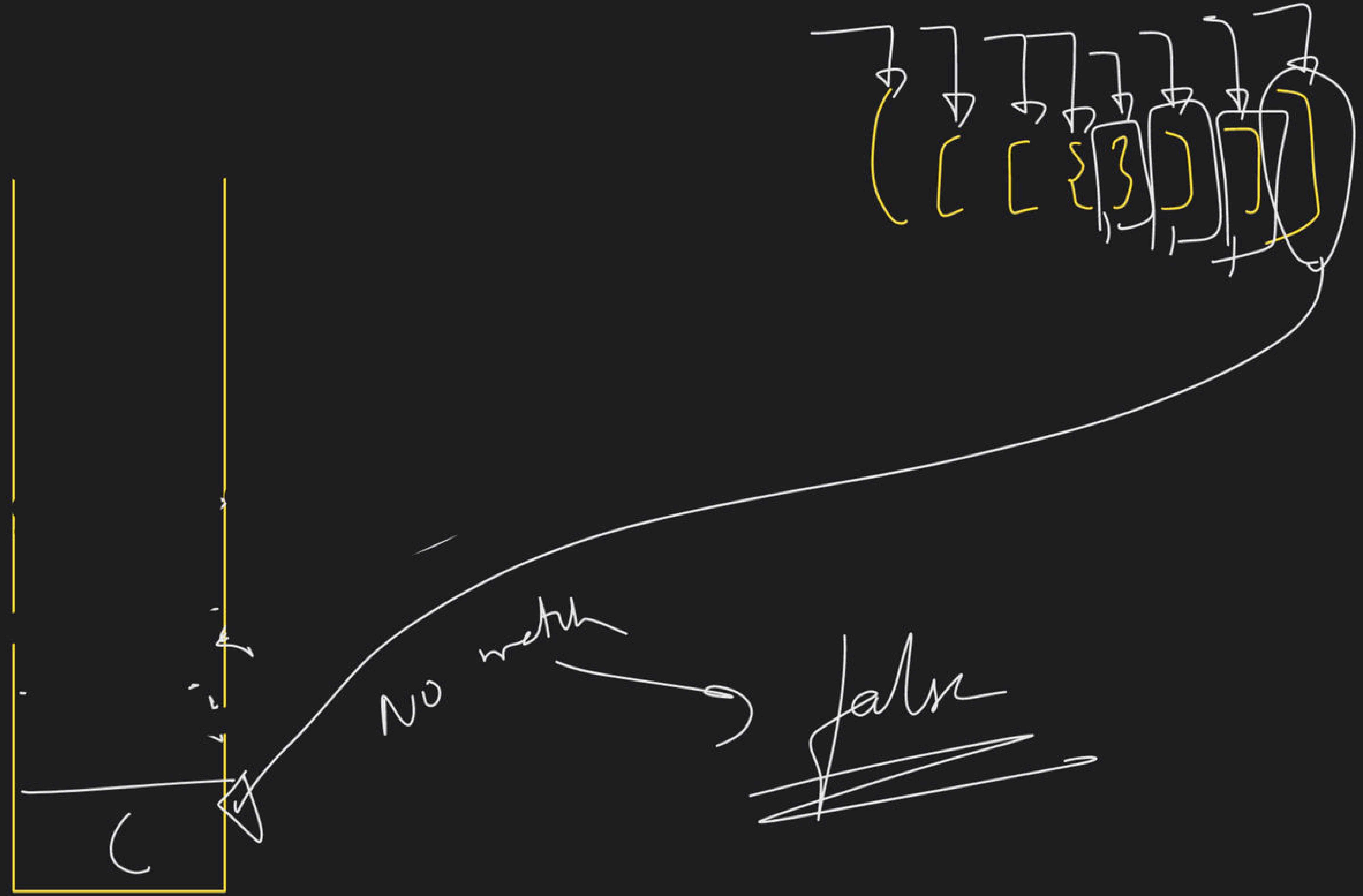


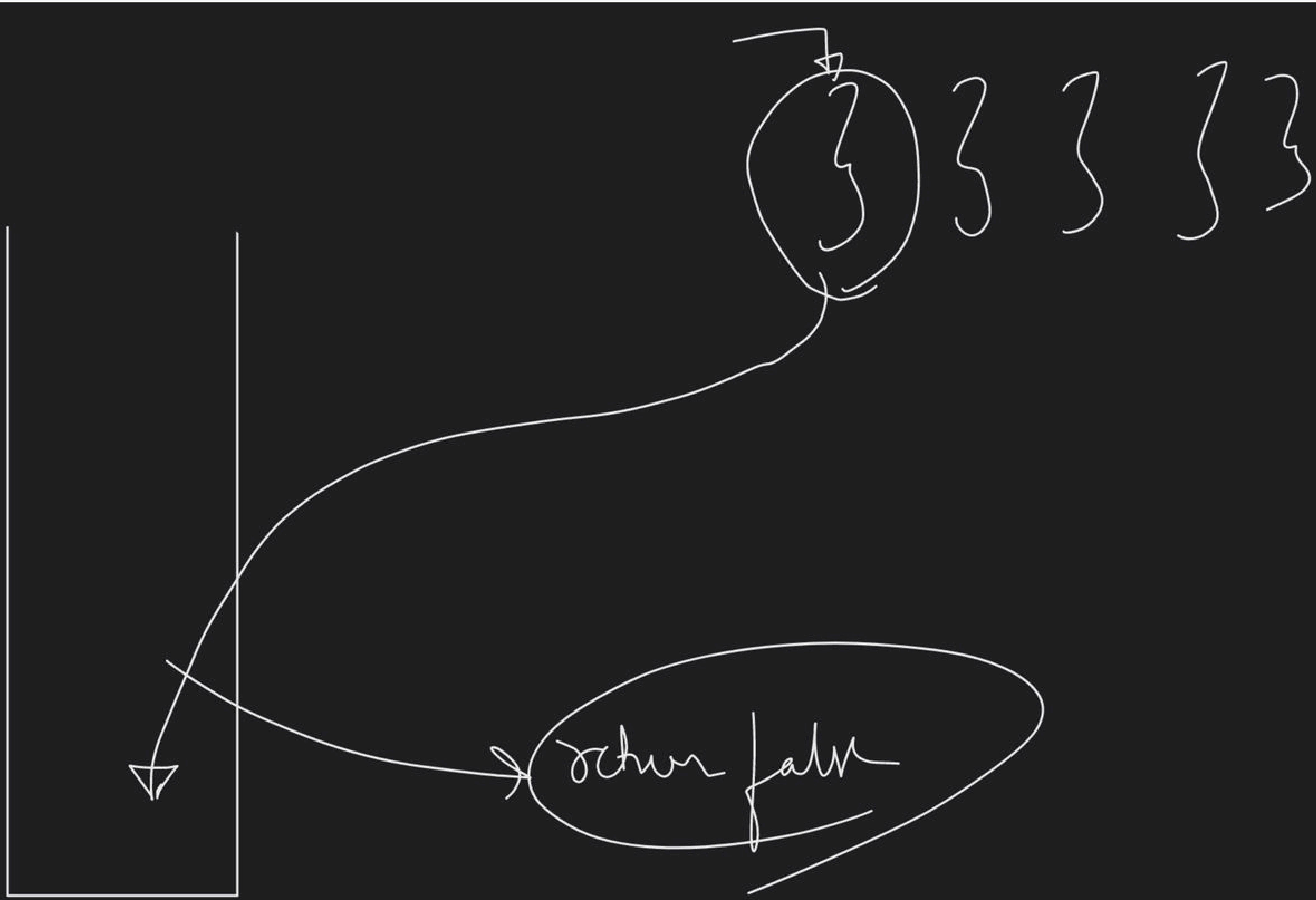


Empty → No → Invalid exp

↓

return false


([[{ }]])
(
No match
false


} } } } }
return false

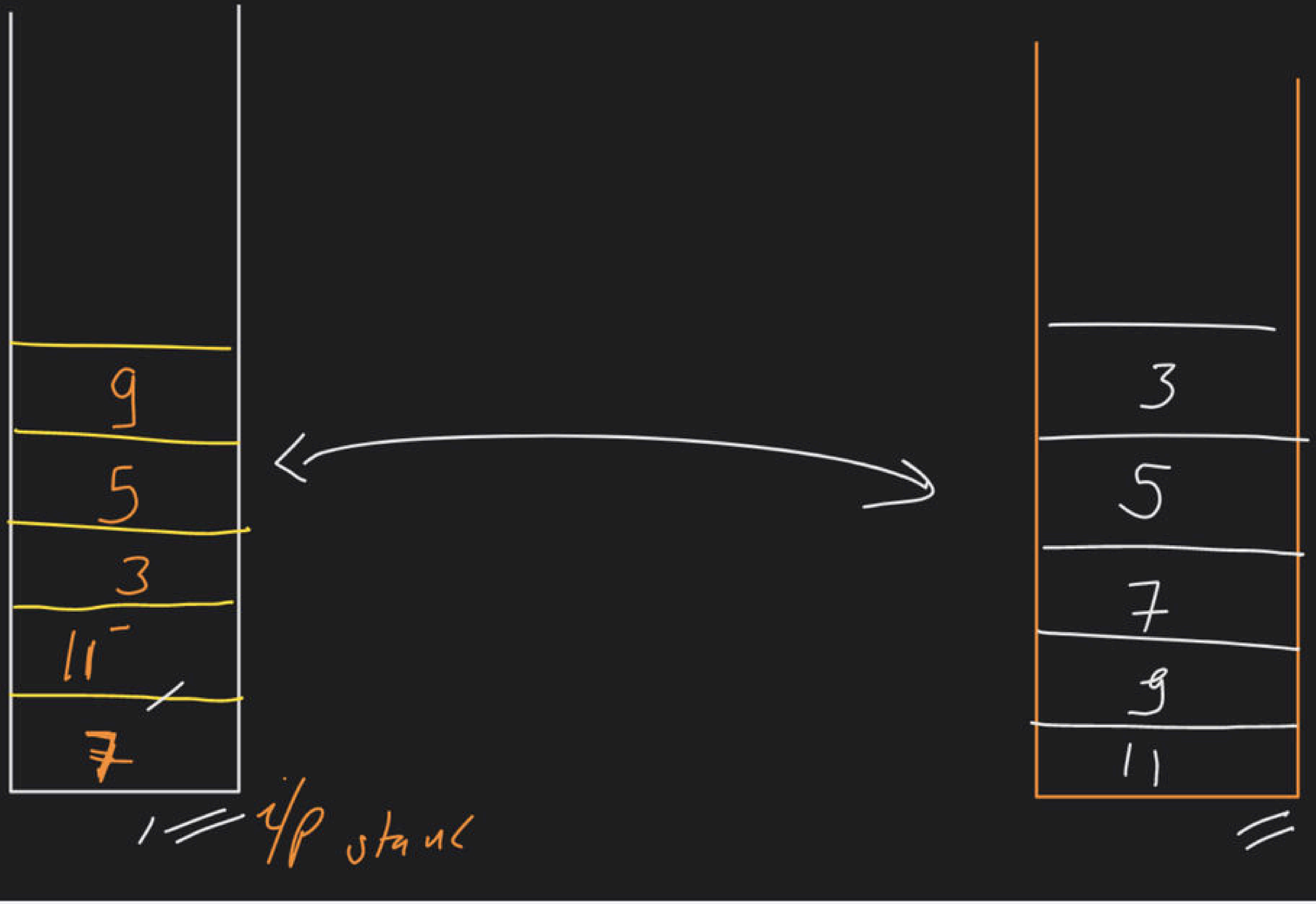
Sort a Stack

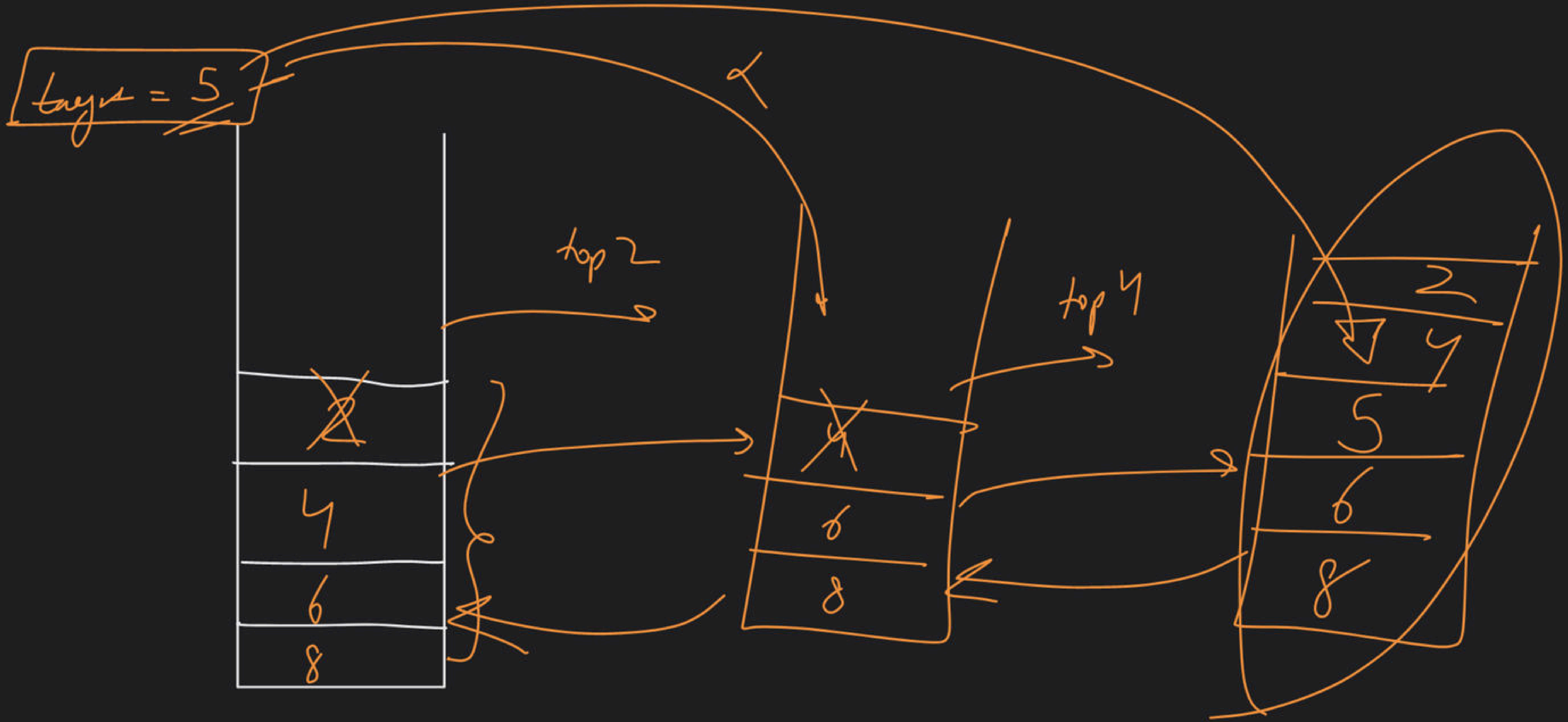
using recursion

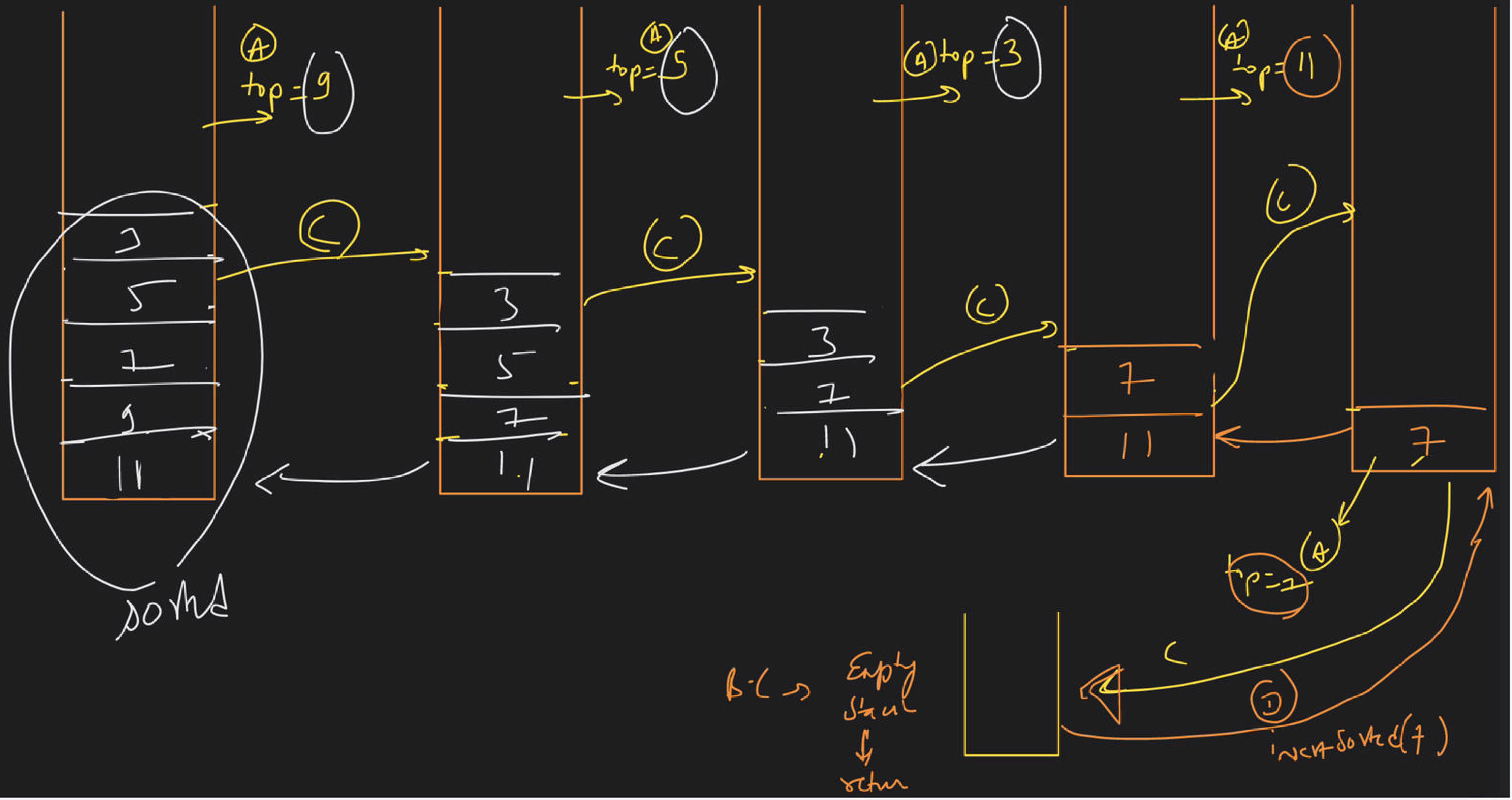
2 min

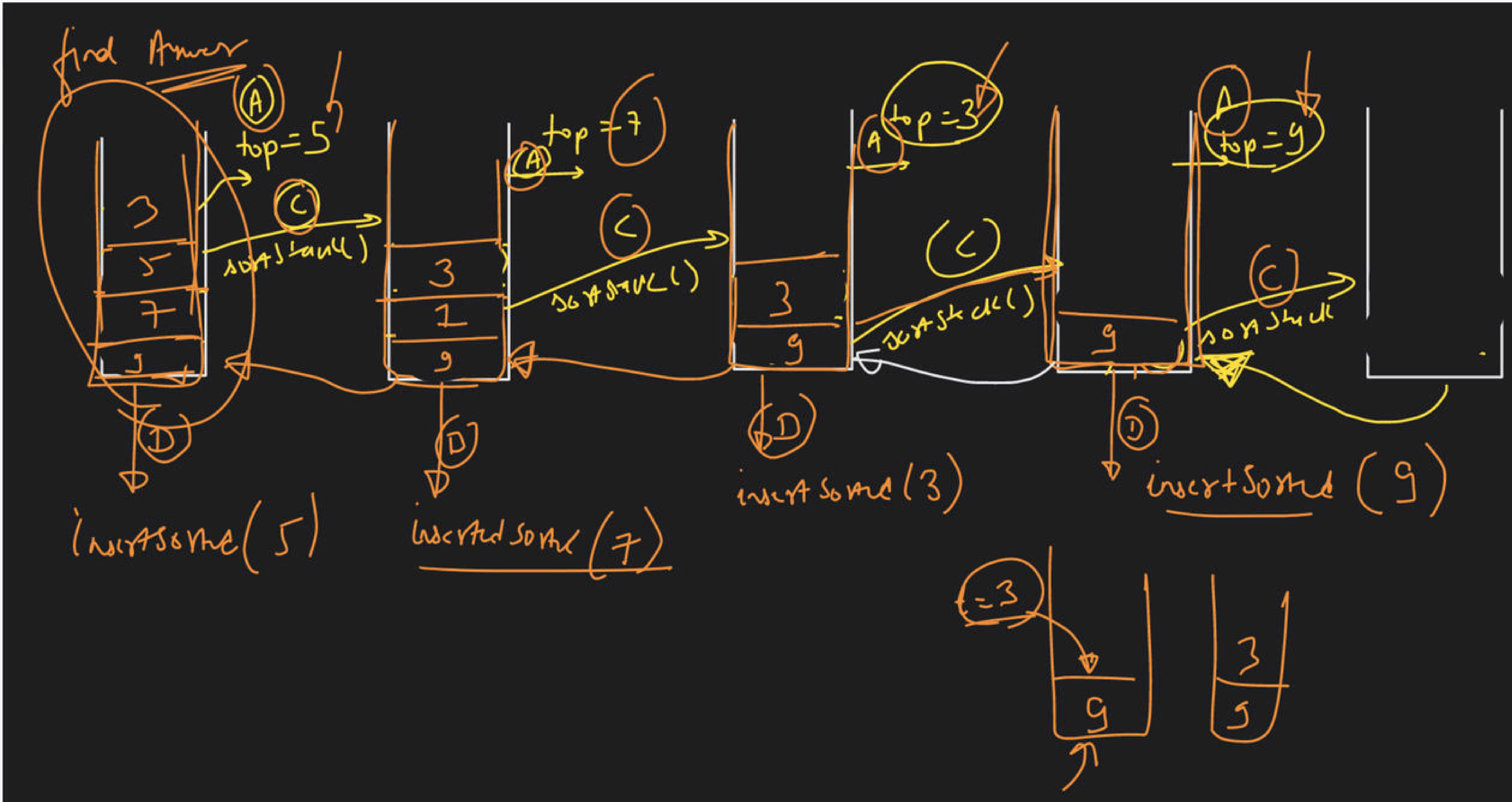
1st task
→ insertSorted()

2nd task
→ sortStack()

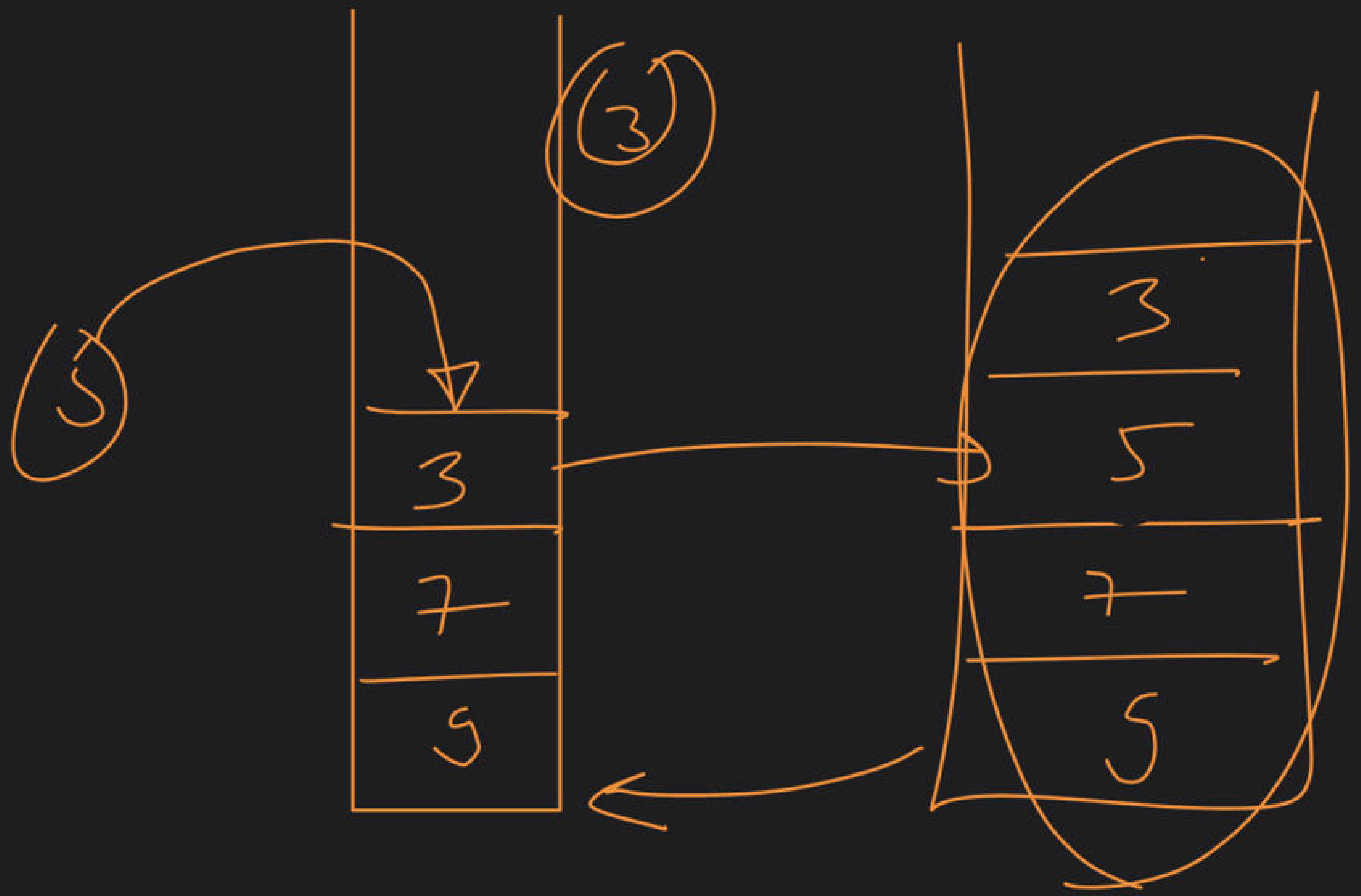


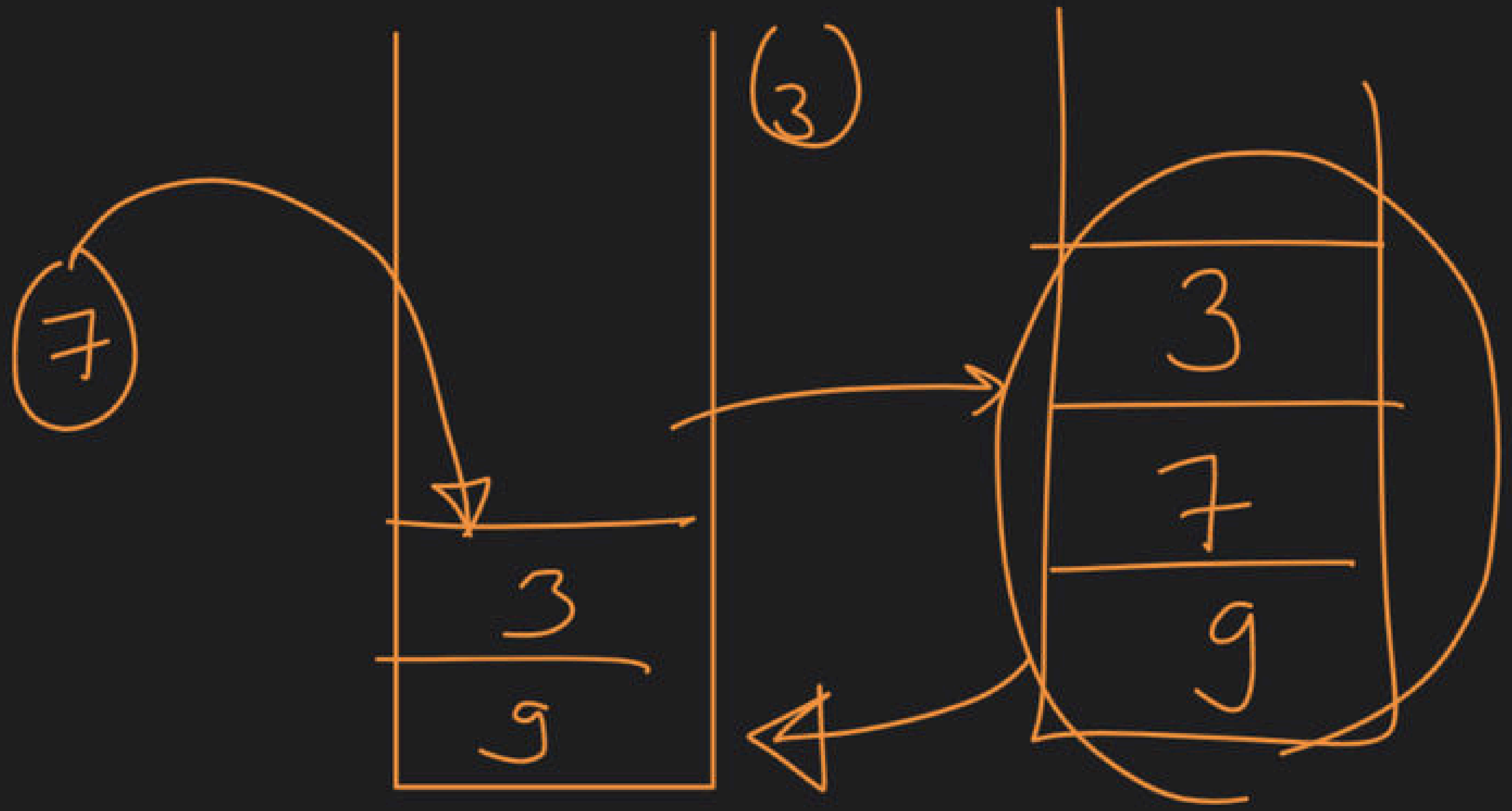

target = 5
<
top 2
top 4
2
4
6
8
4
6
8
2
4
5
6
8

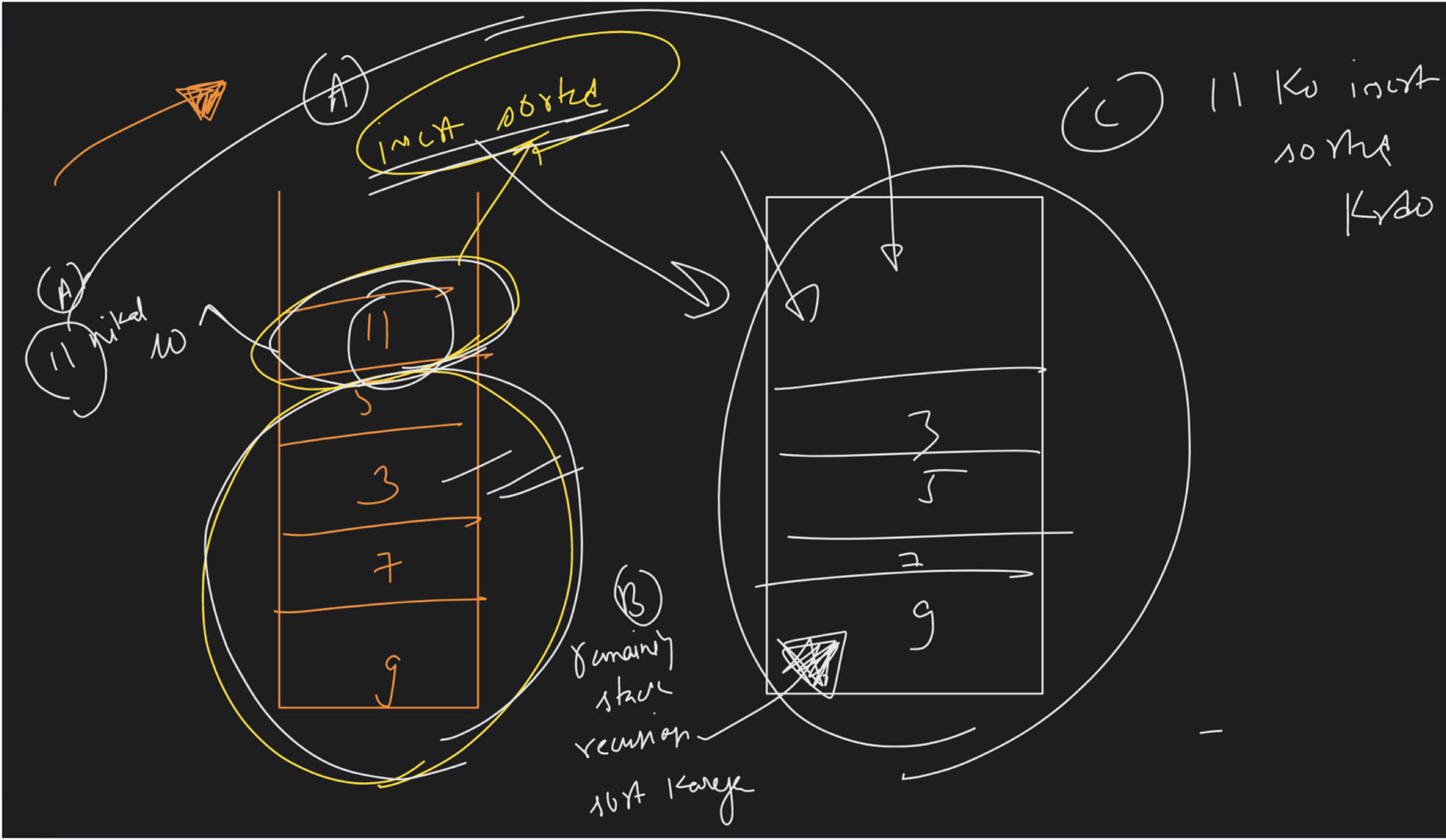

A
top = 9
C
3
5
7
9
11
sorted
A
top = 5
C
3
5
7
11
A
top = 3
C
3
7
11
A
top = 11
C
7
11
7
A
top = 7
C
D
insertsorted(7)
B·C -> Empty stack
↓
return



```
B·C
if(s.empty())
    return;
```


5
3
3
7
9
3
5
7
5


7
3
3
9
3
7
9


A
incha sorted
A
11 nikal
10
11
5
3
7
9
B
remaining stack recursion sort karega
3
5
7
9
C
11 ko insert sorted kardo

Reload

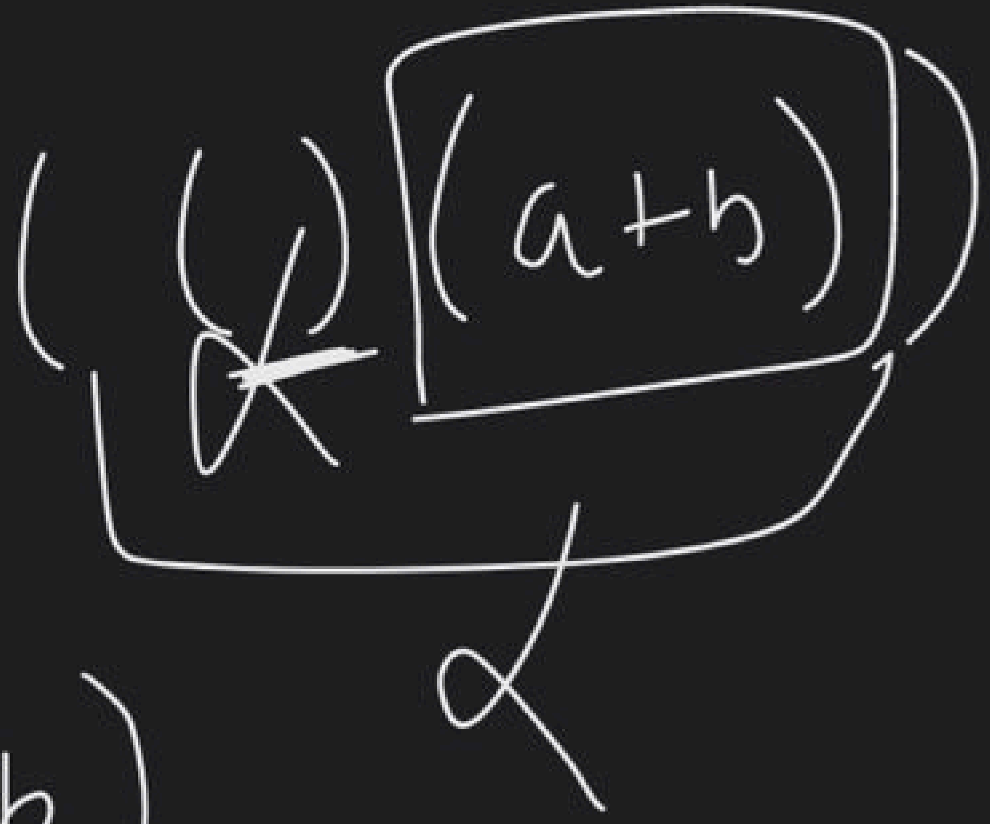
Remove Redundant Brackets

$((a+b)*c)$

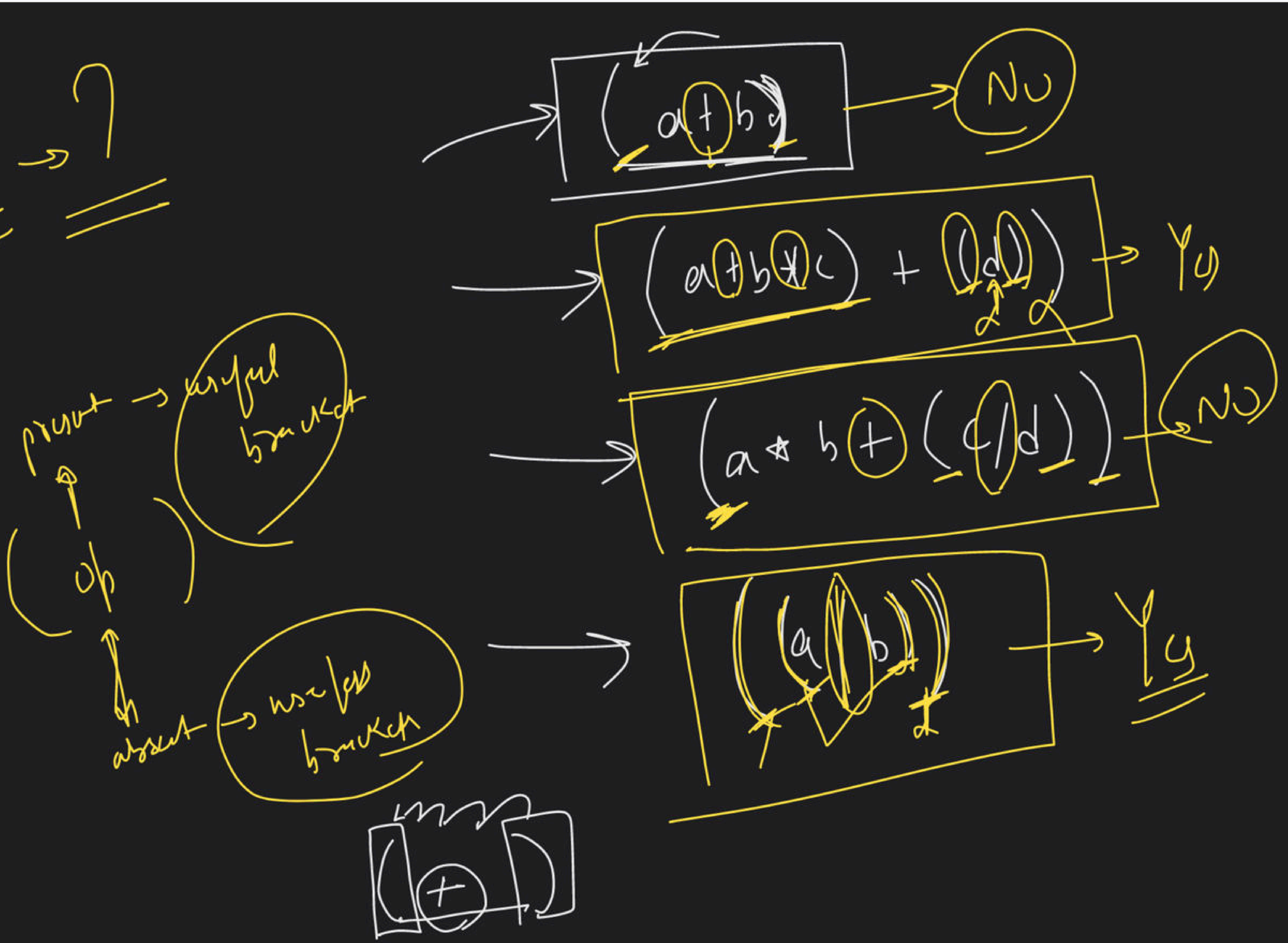
i/p → expression

$(a+b)$

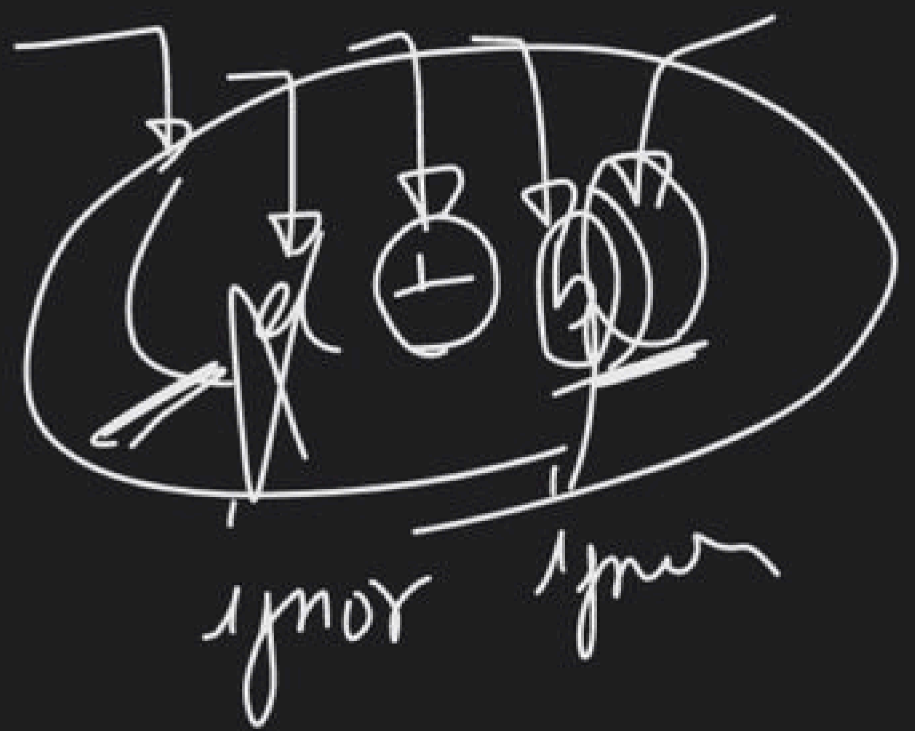
$(\boxed{(a+b)})$

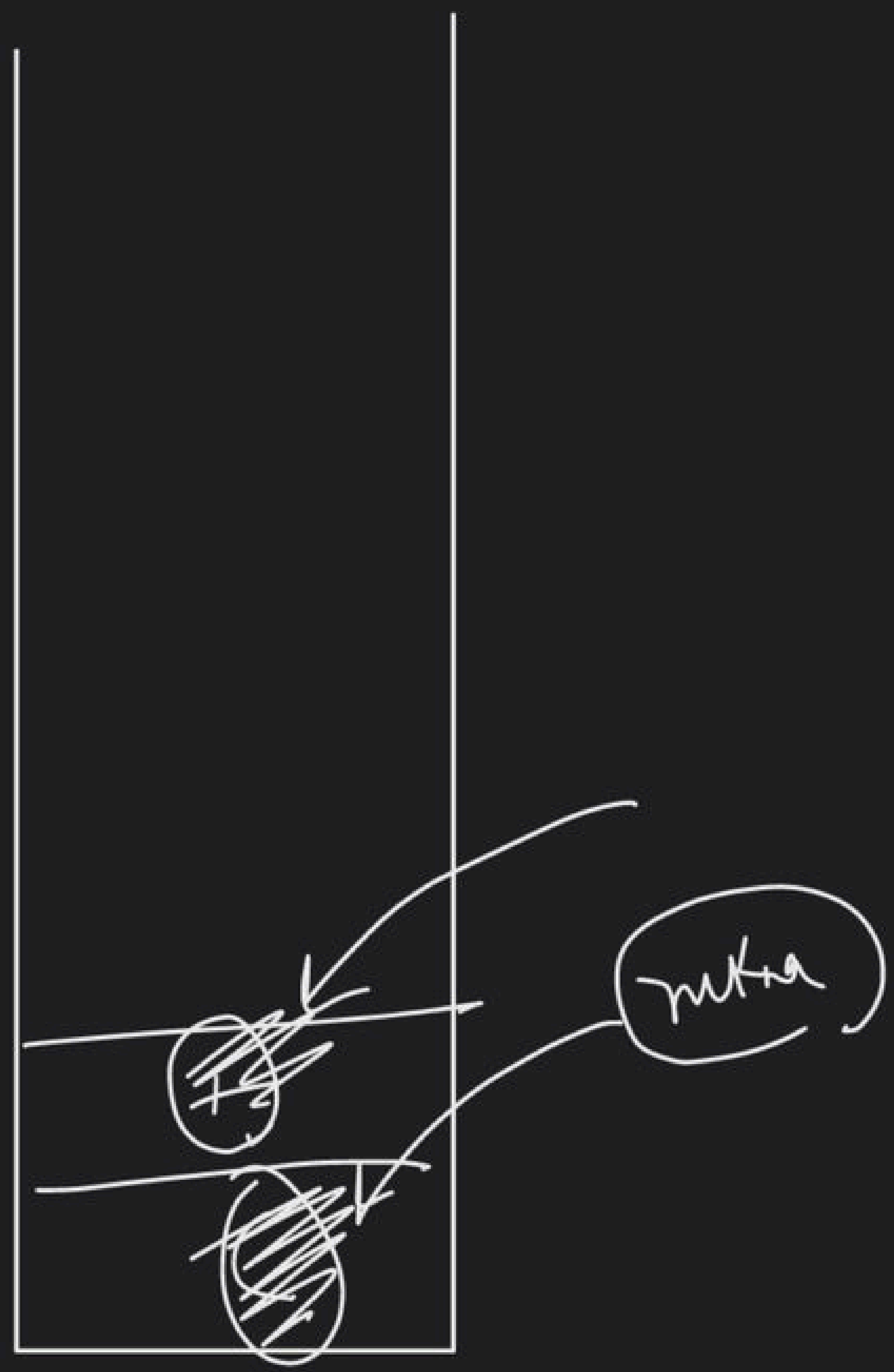


logic → ?


present → useful bracket
(op)
absent → useless bracket
(a+b) → No
(a+b*c) + ((d)) → Yes
(a*b+(c/d)) → No
((a)(b)) → Yes
(+)

operator → stack



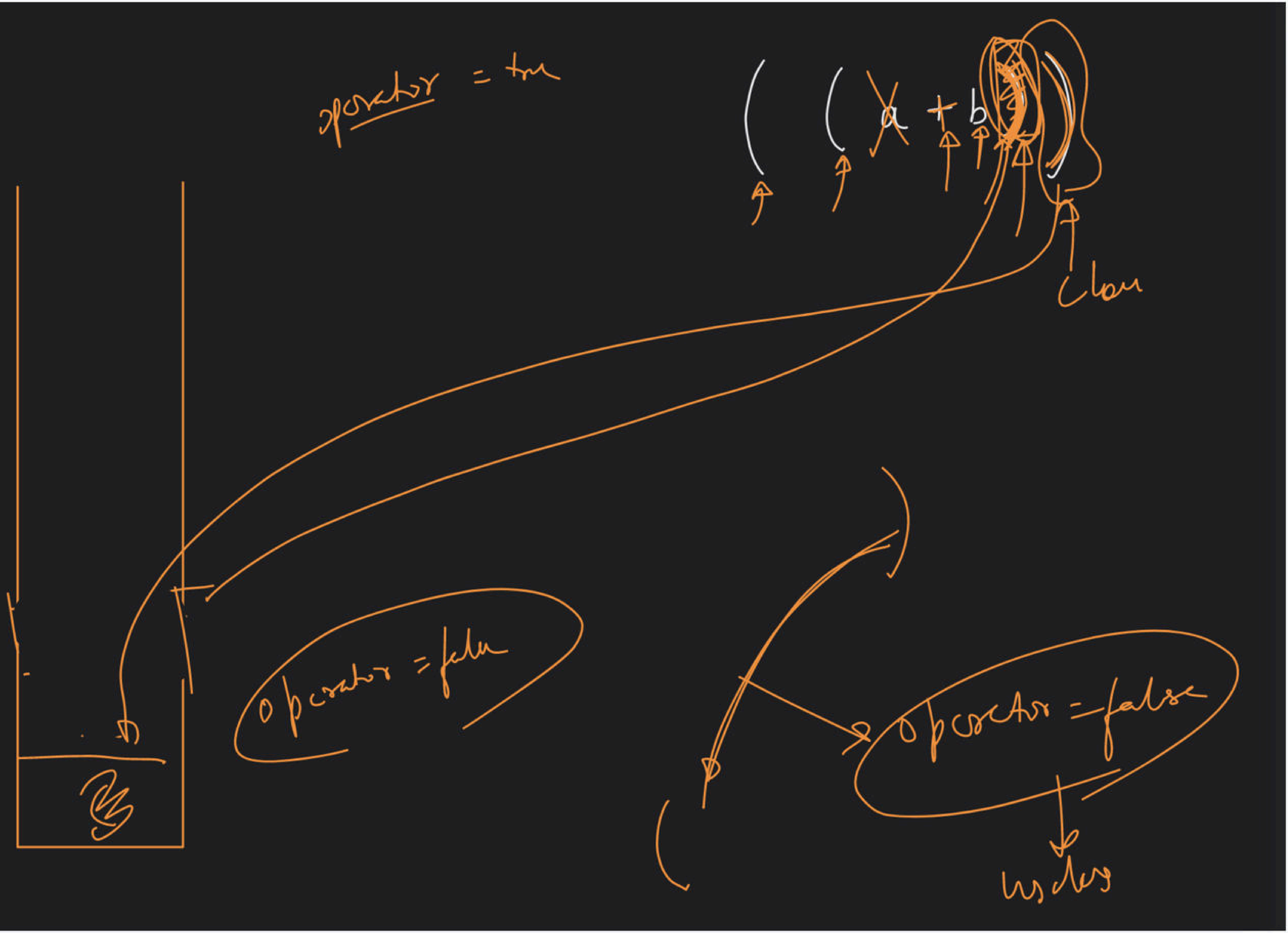


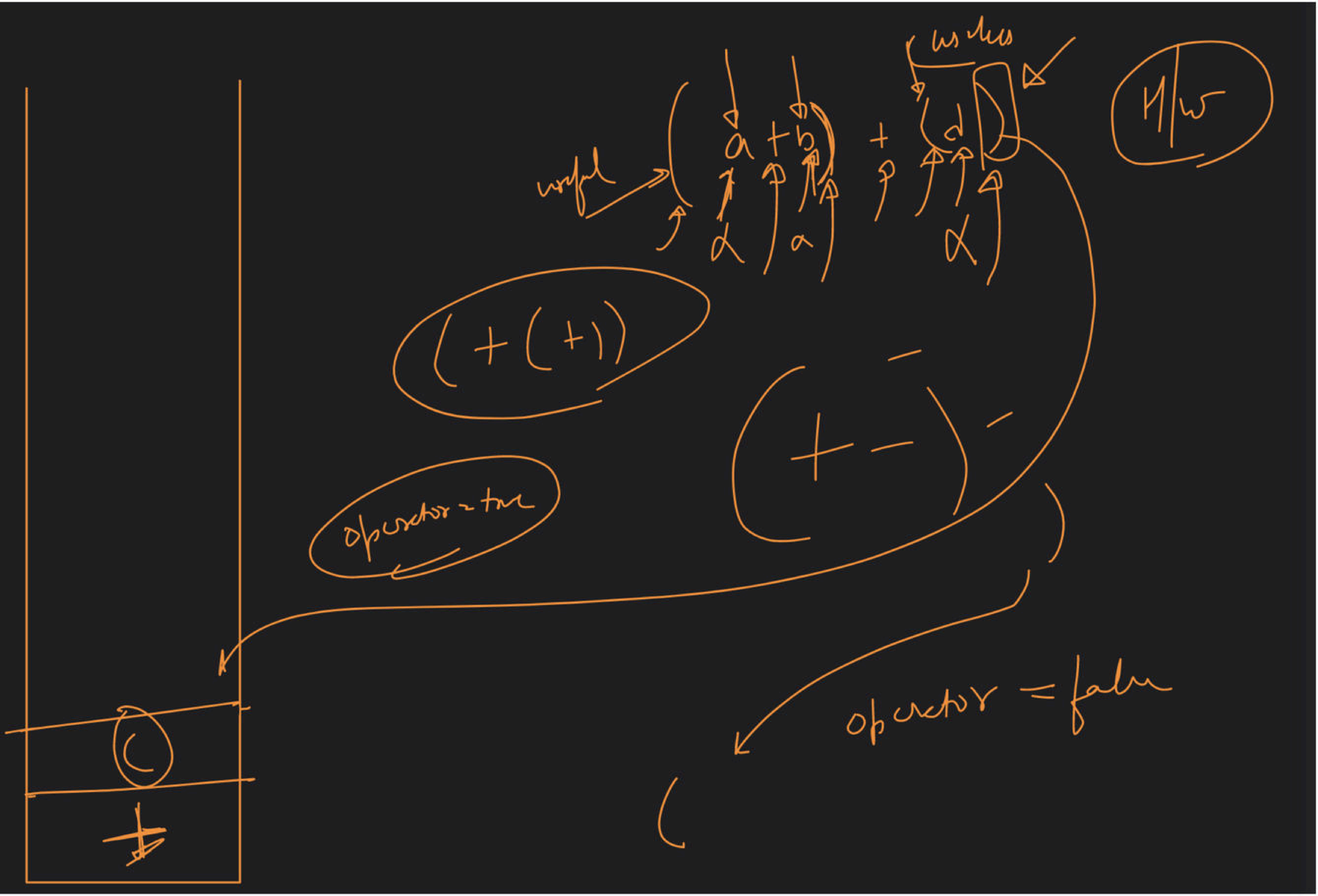
closing bracket
↓
stack
↓
till
opening
bracket

+ - * /

(+ → push

a → z → ignor


operator = true
((a + b))
close
operator = false
operator = false
useless


useless
useful
(a + b) + (d)
α a α
H/W
(+ (+))
operator = true
(+ -) -
operator = false
(
(
+