155. Min Stack

Easy 8346 635 Add to List Share

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

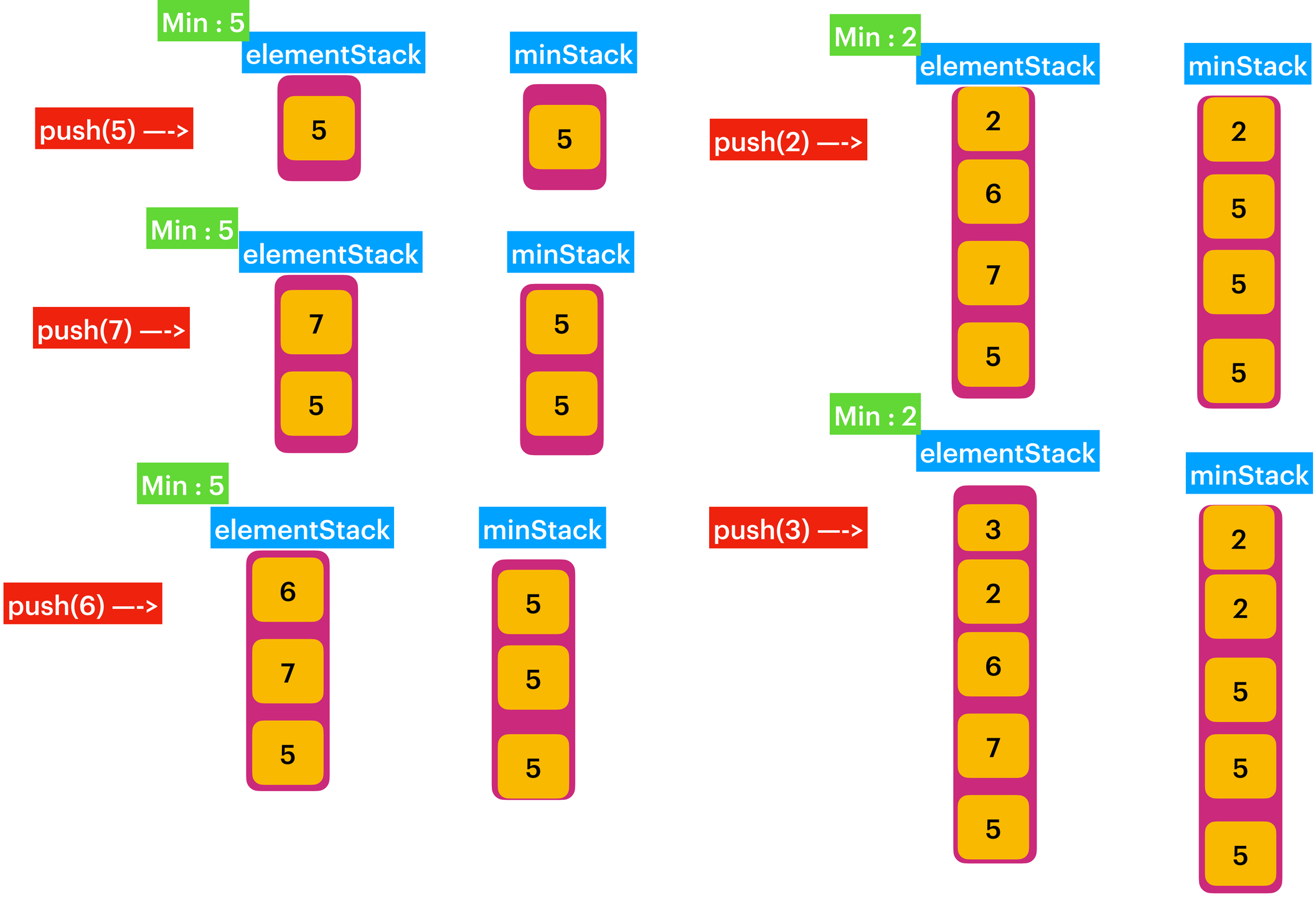
- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

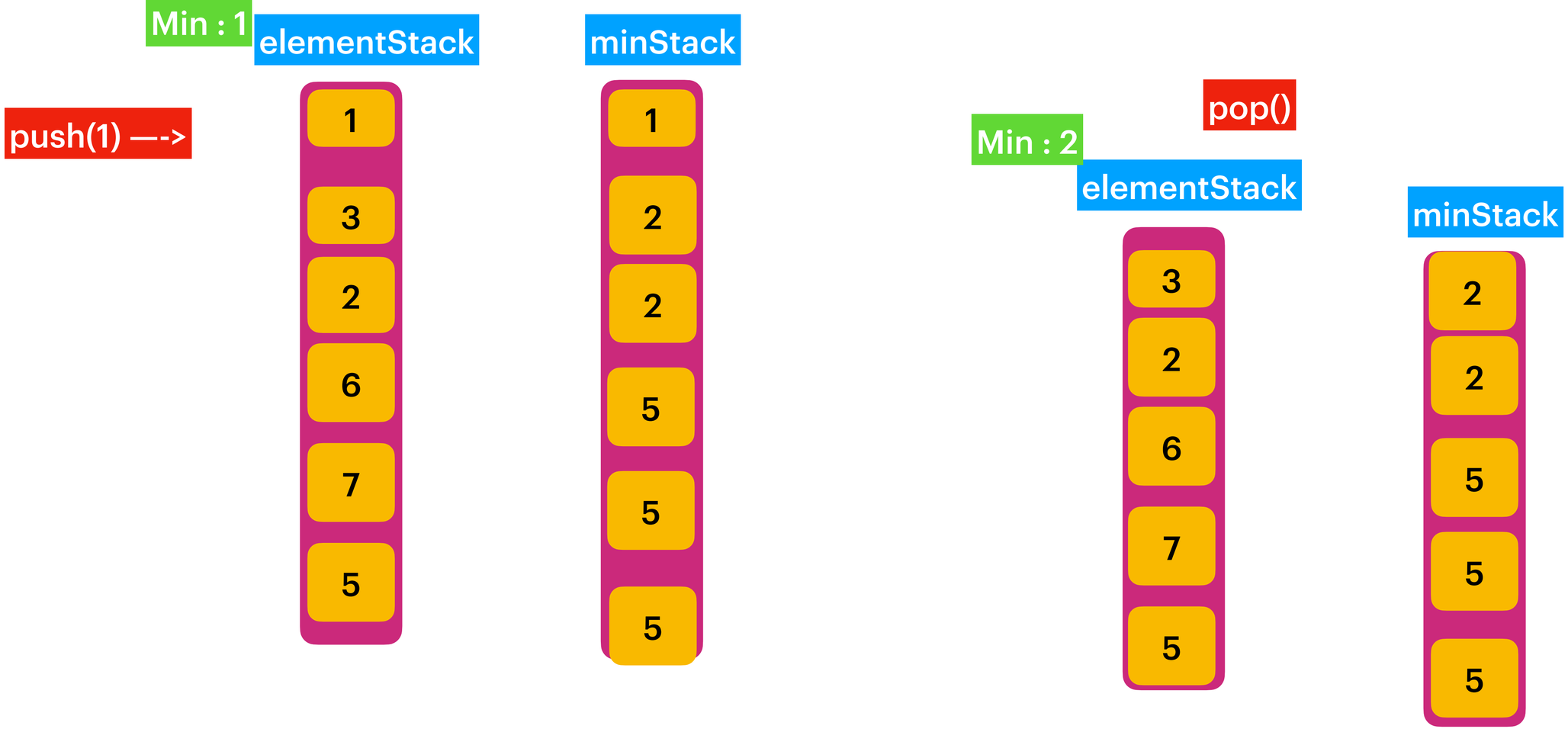
Example 1:

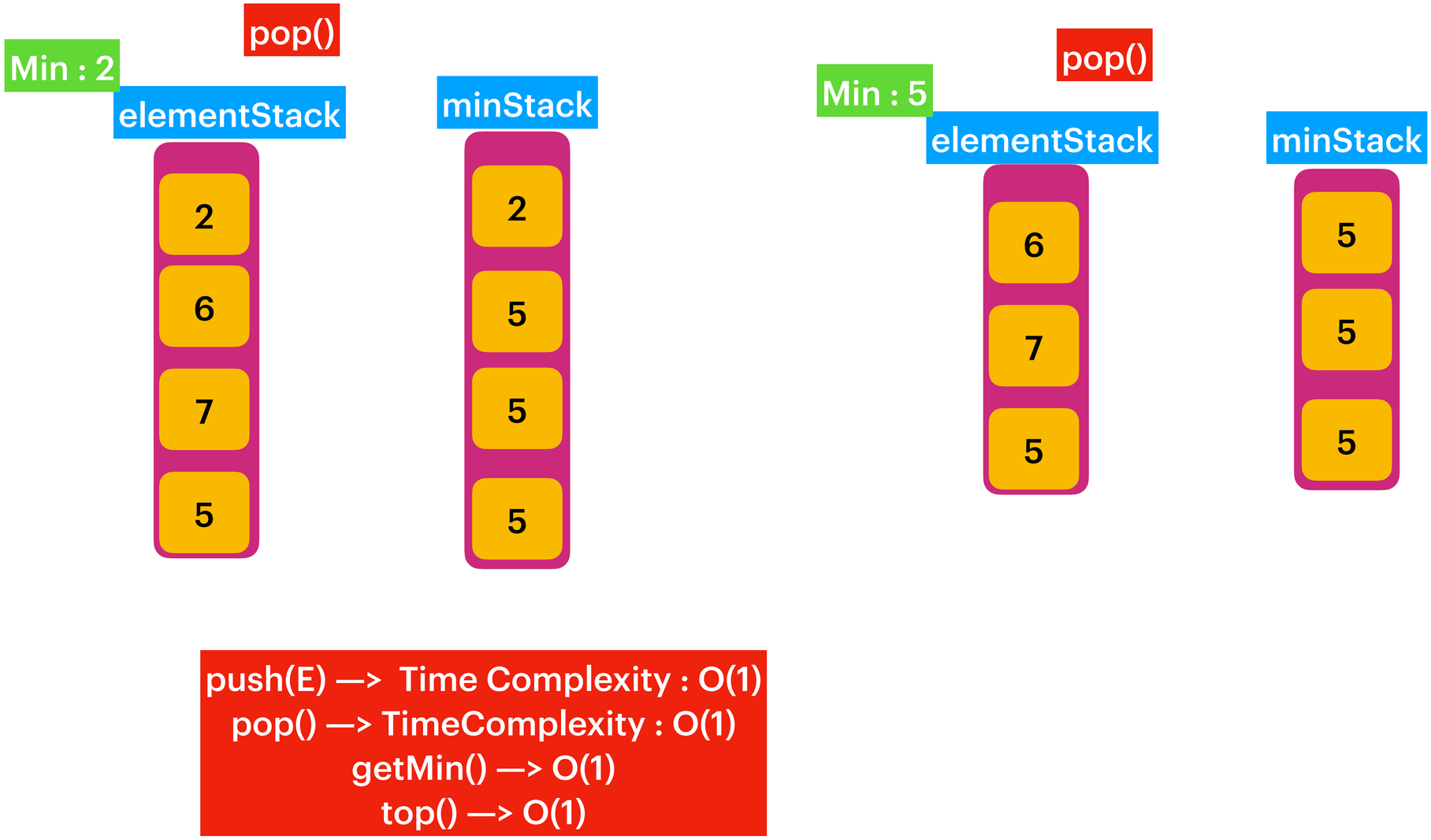
```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$
- Methods `pop`, `top` and `getMin` operations will always be called on **non-empty** stacks.
- At most $3 * 10^4$ calls will be made to `push`, `pop`, `top`, and `getMin`.


Min : 5
elementStack
minStack
push(5) —->
5
5
Min : 5
elementStack
minStack
push(7) —->
7
5
5
5
Min : 5
elementStack
minStack
push(6) —->
6
7
5
5
5
5
Min : 2
elementStack
minStack
push(2) —->
2
6
7
5
2
5
5
5
Min : 2
elementStack
minStack
push(3) —->
3
2
6
7
5
2
2
5
5
5


Min : 1
elementStack
minStack
push(1) —->
1
3
2
6
7
5
1
2
2
5
5
5
pop()
Min : 2
elementStack
minStack
3
2
6
7
5
2
2
5
5
5



push(E) —> Time Complexity : O(1)
pop() —> TimeComplexity : O(1)
getMin() —> O(1)
top() —> O(1)

20. Valid Parentheses

Easy 13517 611 Add to List Share

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Constraints:

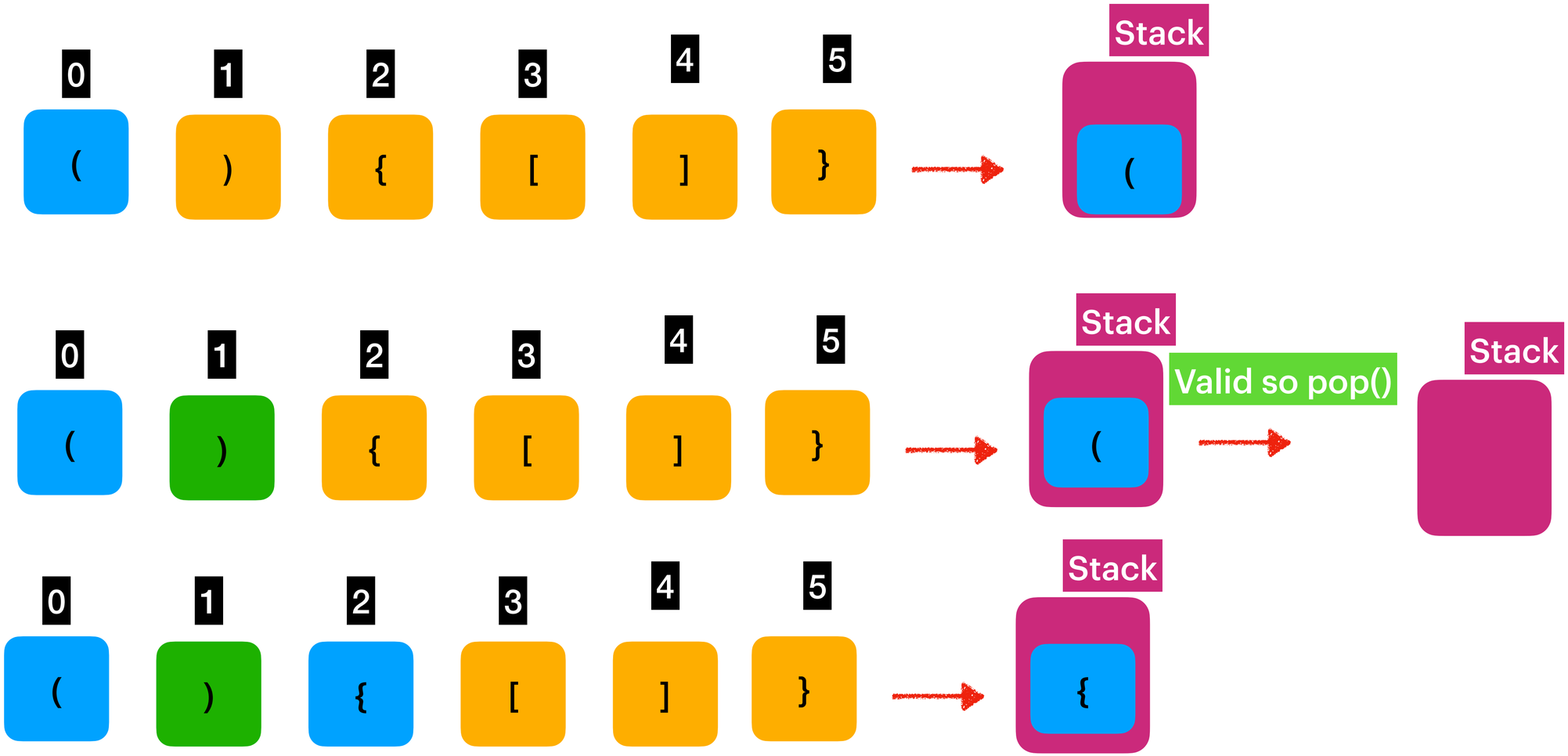
- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

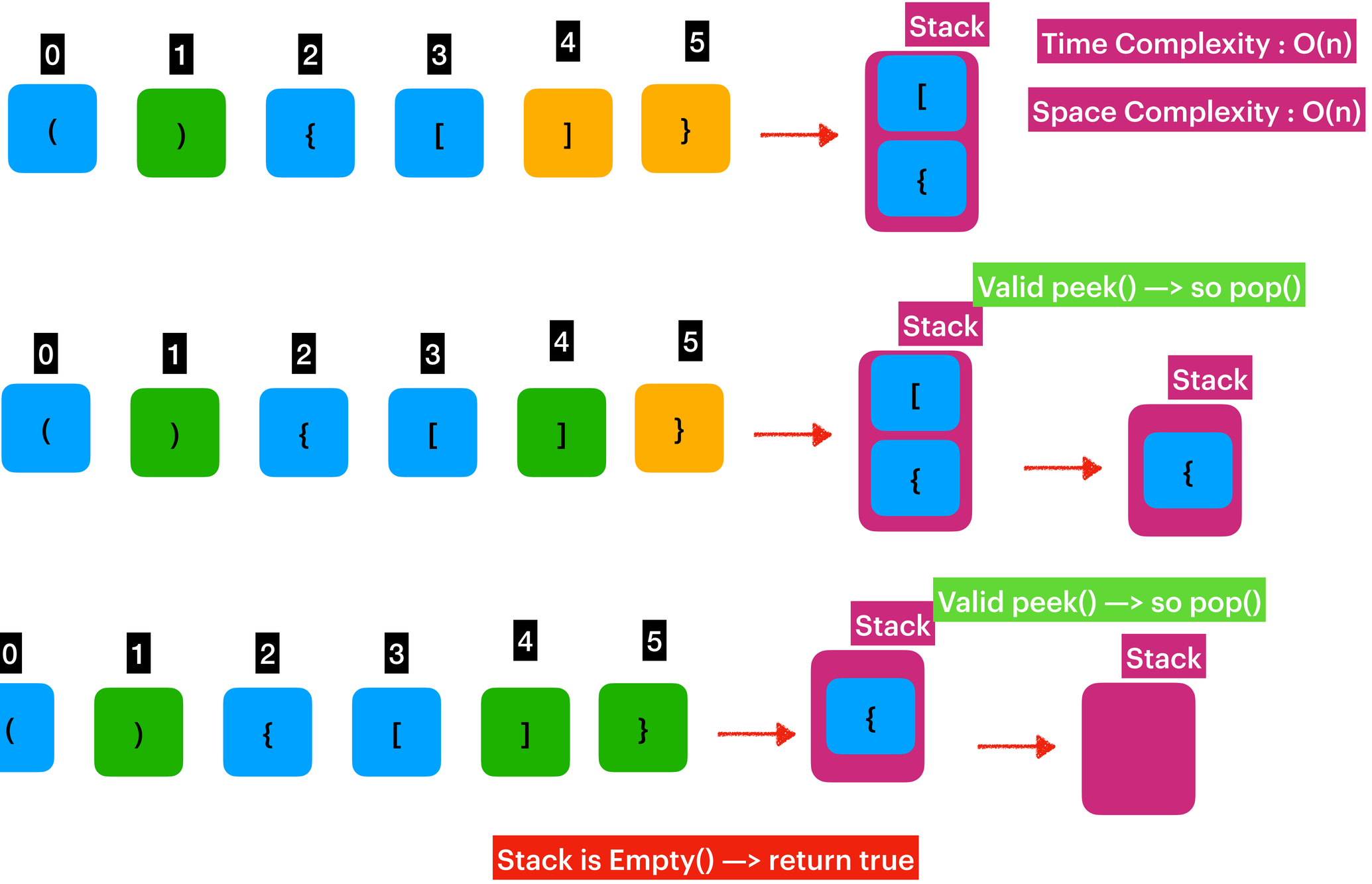
Algorithm :

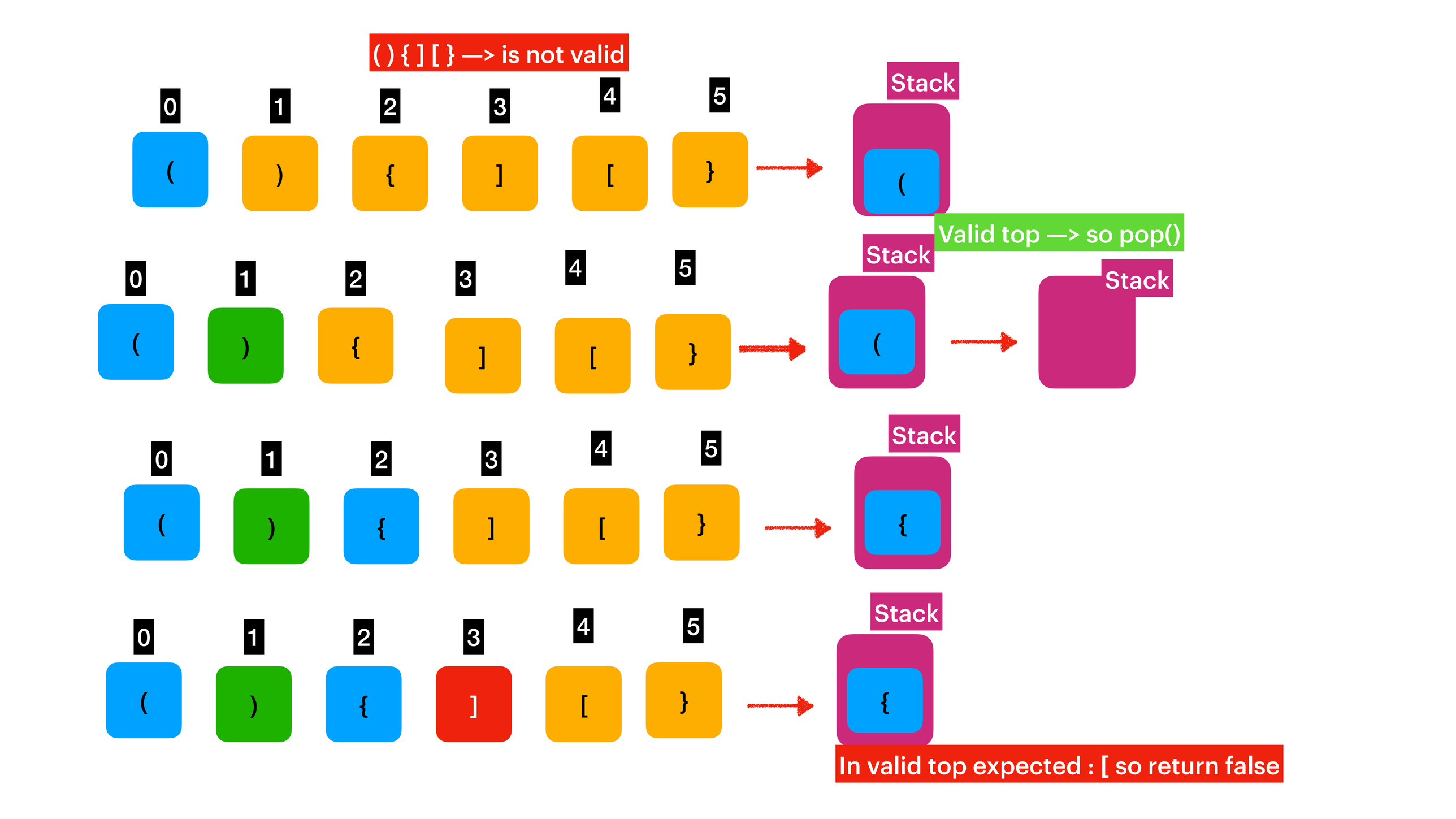
When ever we find either “(“ or “{“ or “[“ —> Push into the stack.
Whenever we find either “)“ or “}“ or “]“
the element in the stack should be equivalent “(“ or “{“ or “[“
if so pop() it otherwise return false.

Base Check : After processing all the elements return true if the stack is Empty.

( ) { [ ] } —> is valid




0
1
2
3
4
5
(
)
{
[
]
}
Stack
[
{
Time Complexity : O(n)
Space Complexity : O(n)
Valid peek() —> so pop()
Stack
Stack
Valid peek() —> so pop()
Stack
Stack
Stack is Empty() —> return true


( ) {] [} —> is not valid
0
1
2
3
4
5
(
)
{
]
[
}
Stack
(
Valid top —> so pop()
Stack
Stack
(
0
1
2
3
4
5
(
)
{
]
[
}
Stack
{
0
1
2
3
4
5
(
)
{
]
[
}
Stack
{
In valid top expected : [so return false

921. Minimum Add to Make Parentheses Valid

Medium 2612 148 Add to List Share

A parentheses string is valid if and only if:

- It is the empty string,
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

You are given a parentheses string `s`. In one move, you can insert a parenthesis at any position of the string.

- For example, if `s = "()))"`, you can insert an opening parenthesis to be `"(()))"` or a closing parenthesis to be `"())))"`.

Return *the minimum number of moves required to make* `s` *valid*.

Example 1:

```
Input: s = "())"
Output: 1
```

Example 2:

```
Input: s = "((("
Output: 3
```

Constraints:

- `1 <= s.length <= 1000`
- `s[i]` is either `'('` or `')'`.

“()))((" —-> need 4 braces —> () ‘(‘) ‘(‘)(‘)’ (‘)’

Algorithm :

-> Take two stacks :: openBrachStack. & closedBraceStack.

-> Whenever we find openBrace add into openStack.

-> When ever we find closedBrace ‘)’, check peek from OpenStack is it ‘(‘ if so pop() it otherwise add into closedBraceStack.

—> return openBraceStack.size() + closedBraceStack.size().

Time Complexity : O(n)
Space Complexity : O(n)

TimeComplexity : O(n)

Space Complexity : O(n)

For Explanation through Algonotes.

“()))((" —-> need 4 braces —> () ‘(‘) ‘(‘)(‘)’ (‘)’

Improvement on Space

Algorithm :

-> Take openCount, closedCount.

-> Whenever we find openBrace ‘(‘ add increment openCount.

-> When ever we find closedBrace ‘)’, check openCount > 0 if so Decrement openCount otherwise increment closedCount.

—> return openCount + closedCount.

Time Complexity : O(n)
Space Complexity : O(1)

TimeComplexity : O(n)

Space Complexity : O(1)