LeetCode 20 — Valid Parentheses

1. Problem Title & Link

20. Valid Parentheses

🔗 https://leetcode.com/problems/valid-parentheses/

2. Problem Summary (Short & Clear)

Given a string s containing only:

() [] {}

Return **true** if the string is a *valid* parentheses sequence.

A string is valid when:

- Every opening bracket has a matching closing bracket
- They close in the **correct order**
- No partial mismatches like "(]" or "{)"
- Stack logic naturally applies

3. Examples

Example 1

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

Example 2

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

Example 3

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

Example 4

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

Example 5

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

4. Constraints

- 1 <= s.length <= 10^4
- Contains only '(){}[]'
- Must validate efficiently in **O(n)**

5. Thought Process (Step-by-Step)

❤️ **Key Idea — Use a Stack**

- Push opening brackets: (, {, [
- When you see a closing bracket:), },]
    - Stack must not be empty
    - Top of stack must match its pair
- If mismatch → return false
- At end → stack must be empty

This is a classic LIFO (Last In, First Out) pattern.

Mapping Closing → Opening

) → (
} → {
] → [
Use a dictionary for constant-time matching.

6. Pseudocode

```
stack = empty
map = { ')':'(', ']':'[', '}':'{' }

for each char c in s:
    if c is opening:
        push to stack
    else:
        if stack empty → false
        if top != map[c] → false
        pop stack

return stack empty
```

7. Code Implementation

Python Solution

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        match = {')': '(', ']': '[', '}': '{'}

        for ch in s:
            # Opening bracket
            if ch in "([{":
                stack.append(ch)
            else:
                # Closing bracket with no matching opening
                if not stack or stack[-1] != match[ch]:
                    return False
                stack.pop()

        # Valid only if stack is empty
        return len(stack) == 0
```

Java Solution

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        Map<Character, Character> match = Map.of(
            ')', '(',
            ']', '[',
            '}', '{'
        );

        for (char ch : s.toCharArray()) {
            // Opening bracket
            if (match.containsValue(ch)) {
                stack.push(ch);
            }
            else { // Closing bracket
                if (stack.isEmpty() || stack.peek() != match.get(ch)) {
                    return false;
                }
                stack.pop();
            }
        }

        return stack.isEmpty();
    }
}
```

8. Time & Space Complexity

Metric	Complexity	Why
Time	O(n)	One pass over string
Space	O(n)	Stack (worst case all '(')

9. Dry Run

Input:

s = "{[]}"

Step	Char	Stack	Action
1	{	[{]	push
2	[	[{ []]	push
3	]	[{]	pop (matched '[')
4	}	[]	pop (matched '{')

Stack empty → **Valid**

Failure Example

Input: "([)]"

Char	Stack	Reason
(	(	push
[	([	push
)	([	mismatch → expected (but top is [→ **False**

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds
Stack for Matching Pairs	Balanced brackets, syntax checking, HTML tags	Forgetting empty-stack check, wrong mapping, not clearing stack	LC 150, LC 394, LC 32, Expression Parsing

11. Common Mistakes

- Not checking if stack is empty before popping
- Wrong mapping direction
- Forgetting to return stack empty at end
- Using if-else chains instead of a mapping table

12. Variations / Follow-Ups

- LC 32: Longest Valid Parentheses
- LC 150: Evaluate Reverse Polish Notation
- LC 224/227: Basic Calculator
- HTML/XML tag validator
- "Minimum invalid parentheses to remove" problems