Stack – Valid Parentheses

1. Introduction

The **Valid Parentheses** problem is a classic application of the **Stack** data structure.

It checks whether an expression containing parentheses is **balanced and correctly ordered**.

This problem helps learners understand:

- Stack usage
- Matching pairs
- Order validation

---

2. What is Valid Parentheses?

An expression is said to have **valid parentheses** if:

- Every opening bracket has a corresponding closing bracket
- Brackets are closed in the **correct order**

Common types of brackets:

```
( )  { }  [ ]
```

---

3. Examples

Valid Expressions

```
()
()[]{}
{[()]}
```

Invalid Expressions

```
(
(]
([)]
```

4. Why Stack is Used?

Stack is used because:

- It follows **LIFO (Last In, First Out)**
- The most recent opening bracket must be closed first
- Perfectly matches the parentheses matching logic

5. Basic Idea Behind the Solution

- Push opening brackets onto the stack
- When a closing bracket appears:
  - Check the top of the stack
  - If it matches, pop it
  - Otherwise, expression is invalid
- At the end:
  - Stack should be empty

6. Logic for Valid Parentheses (Plain English)

1. Create an empty stack
2. Traverse the string character by character
3. If opening bracket is found, push it onto the stack
4. If closing bracket is found:
   - If stack is empty → invalid

- Pop top element and check matching

5. After traversal:
    - If stack is empty → valid
    - Else → invalid

---

7. Visualization Example

Expression:

```
{[()]}
```

Steps:

```
Push {
Push [
Push (
Pop (
Pop [
Pop {
```

Stack is empty → Valid

---

8. Mismatch Scenario Example

Expression:

```
([)]
```

Steps:

```
Push (
Push [
Encounter ) → mismatch
```

Result → Invalid

9. Time and Space Complexity

Aspect	Complexity
Time Complexity	O(n)
Space Complexity	O(n)

Where n is the length of the string.

10. Edge Cases

- Empty string → Valid
- Only opening brackets → Invalid
- Only closing brackets → Invalid
- Single character → Invalid

11. Advantages of Stack-Based Approach

- Efficient
- Easy to understand
- Works for multiple bracket types
- Linear time complexity

12. Limitations

- Requires extra space
- Only checks structural correctness
- Does not evaluate expressions

13. Real-World Applications

- Syntax checking in compilers
- Code editors (bracket highlighting)
- Expression evaluation
- HTML/XML tag validation
- Interview and competitive programming problems

---

14. Summary

- Valid parentheses require proper matching and order
- Stack is ideal due to LIFO nature
- Push opening, pop on closing
- Final stack must be empty
- Time complexity is O(n)
