Stack – Reverse String Using Stack

1. Introduction

Reversing a string is a common problem in Data Structures and Algorithms.

Using a **stack** to reverse a string is a classic example that demonstrates the **LIFO (Last In, First Out)** principle of stacks.

This problem clearly shows how stack behavior naturally helps in reversing data.

---

2. Why Use Stack for Reversing a String?

A stack is ideal for string reversal because:

- The **last character pushed** will be the **first one popped**
- This automatically reverses the order
- It provides a simple and logical solution
- Helps understand practical stack usage

---

3. Basic Idea Behind String Reversal Using Stack

The idea is simple:

- Push each character of the string onto the stack
- Pop characters one by one from the stack
- Store or print the popped characters
- The popped order gives the reversed string

---

4. Stack Principle Used

The solution is based on:

```
LIFO → Last In, First Out
```

When characters are pushed in normal order and popped later, they come out in **reverse order**.

---

5. Logic for Reversing String Using Stack (Plain English)

1. Create an empty stack
2. Traverse the string from left to right
3. Push each character onto the stack
4. Once all characters are pushed, start popping
5. Store each popped character into a new string
6. Continue until the stack becomes empty
7. The new string is the reversed string

---

6. Step-by-Step Example

Original string:

```
"HELLO"
```

Push operation:

```
Push H
Push E
Push L
Push L
Push O
```

Stack:

```
Top → O
      L
      L
      E
      H
```

Pop operation:

```
Pop → O
Pop → L
Pop → L
Pop → E
Pop → H
```

Reversed string:

```
"OLLEH"
```

---

7. Visualization of the Process

```
Original:  H → E → L → L → O
Stack:     H E L L O
Pop order: O → L → L → E → H
Result:    OLLEH
```

---

8. Time and Space Complexity

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

Where n is the length of the string.

---

9. Advantages of Using Stack

- Easy to understand
- Demonstrates stack operations clearly
- Works for any type of string
- Educational and concept-friendly

---

10. Limitations

- Requires extra memory
- Not the most optimized method
- Slower compared to in-place reversal methods

---

11. Real-World Applications

- Undo/Redo operations
- Palindrome checking
- Expression evaluation
- Text processing
- Interview and academic problems

---

12. Comparison with Other Methods

Method	Space Used	Complexity
Stack method	Extra space	O(n)
Two-pointer method	No extra space	O(n)
Built-in reverse	Optimized	O(n)

---

13. Summary

- Stack reverses string using LIFO

- Characters are pushed then popped
- Reversal happens naturally
- Time complexity is O(n)
- Excellent example of stack usage
