<https://leetcode.com/problems/faulty-keyboard/description/>

Your laptop keyboard is faulty, and whenever you type a character 'i' on it, it reverses the string that you have written. Typing other characters works as expected.

You are given a **0-indexed** string s, and you type each character of s using your faulty keyboard.

Return *the final string that will be present on your laptop screen.*

**Example 1:**

**Input:** s = "string"

**Output:** "rtsng"

**Explanation:**

After typing first character, the text on the screen is "s".

After the second character, the text is "st".

After the third character, the text is "str".

Since the fourth character is an 'i', the text gets reversed and becomes "rts".

After the fifth character, the text is "rtsn".

After the sixth character, the text is "rtsng".

Therefore, we return "rtsng".

**Example 2:**

**Input:** s = "poiinter"

**Output:** "ponter"

**Explanation:**

After the first character, the text on the screen is "p".

After the second character, the text is "po".

Since the third character you type is an 'i', the text gets reversed and becomes "op".

Since the fourth character you type is an 'i', the text gets reversed and becomes "po".

After the fifth character, the text is "pon".

After the sixth character, the text is "pont".

After the seventh character, the text is "ponte".

After the eighth character, the text is "ponter".

Therefore, we return "ponter".

**Constraints:**

1 <= s.length <= 100

s consists of lowercase English letters.

s[0] != 'i'

**Attempt 1: 2025-07-12**

**Solution 1: String (10 min)**

class Solution {

    public String finalString(String s) {

        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < s.length(); i++) {

            if(s.charAt(i) == 'i') {

                sb.reverse();

            } else {

                sb.append(s.charAt(i));

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(n\*k) where n is string length and k is number of 'i's (each reverse is O(n))

Space Complexity: O(n) for the StringBuilder

**Refer to Deepseek**

**Problem Statement**

Your laptop keyboard is faulty, and whenever you type a character 'i', it reverses the string typed so far. Given a string s representing what you intend to type, return the string that will actually appear on screen.

**Approaches**

**Simulation Approach**:

Iterate through each character in the input string

When encountering 'i', reverse the current string

Otherwise, append the character to the string

**Optimized Approach**:

Use a deque (double-ended queue) to efficiently handle reversals

Track whether the string should be in normal or reversed state

Append characters to front or back based on current state

**Solution Code (Simulation Approach)**

class Solution {

public String finalString(String s) {

StringBuilder result = new StringBuilder();

for (char c : s.toCharArray()) {

if (c == 'i') {

result.reverse();

} else {

result.append(c);

}

}

return result.toString();

}

}

**Solution Code (Optimized Deque Approach)**

class Solution {

public String finalString(String s) {

Deque<Character> deque = new ArrayDeque<>();

boolean reverse = false;

for (char c : s.toCharArray()) {

if (c == 'i') {

reverse = !reverse;

} else {

if (reverse) {

deque.addFirst(c);

} else {

deque.addLast(c);

}

}

}

StringBuilder result = new StringBuilder();

while (!deque.isEmpty()) {

result.append(reverse ? deque.removeLast() : deque.removeFirst());

}

return result.toString();

}

}

**Explanation**

**Simulation Approach**

**Simple Implementation**: Directly follows the problem description

**Time Complexity**: O(n\*k) where n is string length and k is number of 'i's (each reverse is O(n))

**Space Complexity**: O(n) for the StringBuilder

**Optimized Deque Approach**

**Efficient Reversals**: Uses a boolean flag to track reversal state

**Insertion Logic**:

When not reversed: append to end (addLast)

When reversed: append to front (addFirst)

**Final Construction**: Reads from front or back based on final reverse state

**Time Complexity**: O(n) - each operation is O(1)

**Space Complexity**: O(n) for the deque

**Example Walkthrough**

Input: "string"

's' → append → "s"

't' → append → "st"

'r' → append → "str"

'i' → reverse → "rts"

'n' → append (reversed state) → "nts"

'g' → append (reversed state) → "gnts"Output: "gnts"

The optimized deque approach is more efficient for cases with many 'i' characters, while the simulation approach is simpler to implement for small inputs.

**Refer to**

[L345.Reverse Vowels of a String (Ref.L344,L1119)](note://CA879E2192A94099996642D7560B6FB9)