📓 Reverse String II - Documentation

1. Problem Statement

Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

- If there are fewer than k characters left, reverse all of them.
- If there are between k and 2k characters, reverse the first k characters and leave the rest unchanged.

Constraints:

- $1 <= s.length <= 10^4$
- s consists only of lowercase English letters.
- $1 <= k <= 10^4$

2. Intuition

The task requires reversing characters in fixed-size windows of 2k. Within each window:

- The first k characters should be reversed.
- The next k characters remain unchanged.

By processing the string in chunks of 2k, we can easily apply this rule consistently.

3. Key Observations

- The entire string is split into segments of 2k length.
- If the segment has at least k characters, reverse the first k.
- If less than k, reverse all of them.
- The second part of each 2k segment is untouched.

4. Approach

- Convert the string to a list (strings are immutable in Python).
- Iterate over the string in steps of 2k.
- For each iteration, reverse the first k characters using slicing and reversed().
- Join the list back into a string and return it.

5. Edge Cases

- If k >= len(s): reverse the entire string.
- If len(s) is not a multiple of 2k: handle the remaining characters correctly.
- If k == 1: only single characters are reversed.
- If k == len(s) // 2: test for symmetrical reversal.

6. Complexity Analysis

Time Complexity

- O(n): Each character is visited once, and reversal of k characters is O(k) — repeated n / (2k) times, totaling O(n).

Space Complexity

- O(n): Due to list conversion and string join at the end.

7. Alternative Approaches

- String slicing without list conversion: Not efficient due to immutability.
- Manual character swapping: Less readable and more error-prone than using reversed() and slicing.
- Using recursion: Not suitable here due to large input size and depth.

8. Test Cases

Input	k	Output	Explanation
"abcdefg"	2	"bacdfeg"	First 2 reversed, next 2 untouched, last 3: first 2 reversed
"abcd"	2	"bacd"	First 2 reversed, next 2 untouched
"a"	1	"a"	Single character remains same
"abcdef"	3	"cbadef"	First 3 reversed, next 3 untouched
"abcdefg"	8	"gfedcba"	Less than k left → reverse all

9. Final Thoughts

This problem is a great exercise in string manipulation and understanding index-based slicing. The main takeaway is to approach such problems by:

- Dividing the input into manageable segments.
- Applying consistent transformation logic.
- Being cautious with string immutability in Python.