<https://leetcode.com/problems/reverse-string-ii/description/>

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 less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the other as original.

**Example 1:**

**Input:** s = "abcdefg", k = 2

**Output:** "bacdfeg"

**Example 2:**

**Input:** s = "abcd", k = 2

**Output:** "bacd"

**Constraints:**

1 <= s.length <= 104

s consists of only lowercase English letters.

1 <= k <= 104

**Attempt 1: 2025-07-13**

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

class Solution {

    public String reverseStr(String s, int k) {

        char[] arr = s.toCharArray();

        int n = s.length();

        for(int i = 0; i < n; i += 2 \* k) {

            int start = i;

            int end = Math.min(i + k - 1, n - 1);

            reverse(arr, start, end);

        }

        return new String(arr);

    }

    private void reverse(char[] arr, int i, int j) {

        while(i < j) {

            char tmp = arr[i];

            arr[i] = arr[j];

            arr[j] = tmp;

            i++;

            j--;

        }

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

**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 less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the others as original.

**Approach**

**Convert to Char Array**: For in-place modification

**Iterate in Steps of 2k**: Process each block of 2k characters

**Reverse First k Characters**: For each block, reverse the first k characters

**Handle Edge Cases**: When remaining characters are less than k or between k and 2k

**Solution Code**

class Solution {

public String reverseStr(String s, int k) {

char[] arr = s.toCharArray();

int n = arr.length;

for (int i = 0; i < n; i += 2 \* k) {

int left = i;

// Determine the right boundary (either i+k-1 or end of string)

int right = Math.min(i + k - 1, n - 1);

reverse(arr, left, right);

}

return new String(arr);

}

private void reverse(char[] arr, int left, int right) {

while (left < right) {

char temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}

**Explanation**

**Conversion**: Convert the string to a char array for in-place modification

**Loop Setup**: Iterate through the array in steps of 2k

**Reverse Logic**:

For each block starting at index i, reverse from i to i+k-1

If there are fewer than k characters remaining, reverse all remaining characters

**Helper Function**: The reverse method swaps characters between left and right indices

**Example Walkthrough**

Input: s = "abcdefg", k = 2

First block (0-3): reverse first 2 chars → "bacdefg"

Next block (4-7): reverse next 2 chars → "bacdfeg"

Remaining chars (6-7): only 1 char left → no change

Output: "bacdfeg"

**Complexity Analysis**

**Time Complexity**: O(n) - Each character is visited exactly once

**Space Complexity**: O(n) - For the character array (O(1) additional space)

**Refer to**

[L344.Reverse String (Ref.L345,L541)](note://WEBfd25055adcca24c731abdeb888e6e120)

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