Solution: Reverse Words in a String

Let's solve the Reverse Words in a String problem using the Two Pointers pattern.

We'll cover the following

- Statement
- Solution
  - Time complexity
  - Space complexity

Statement

Given a sentence, reverse the order of its words without affecting the order of letters within the given word.

Constraints:

- The sentence contains English uppercase and lowercase letters, digits, and spaces.
- $1 \leq \texttt{sentence.length} \leq 10^4$

- The order of the letters within a word is not to be reversed.

> **Note:** The input string may contain leading or trailing spaces or multiple spaces between words. The returned string, however, should only have a single space separating each word. Do not include any extra spaces.

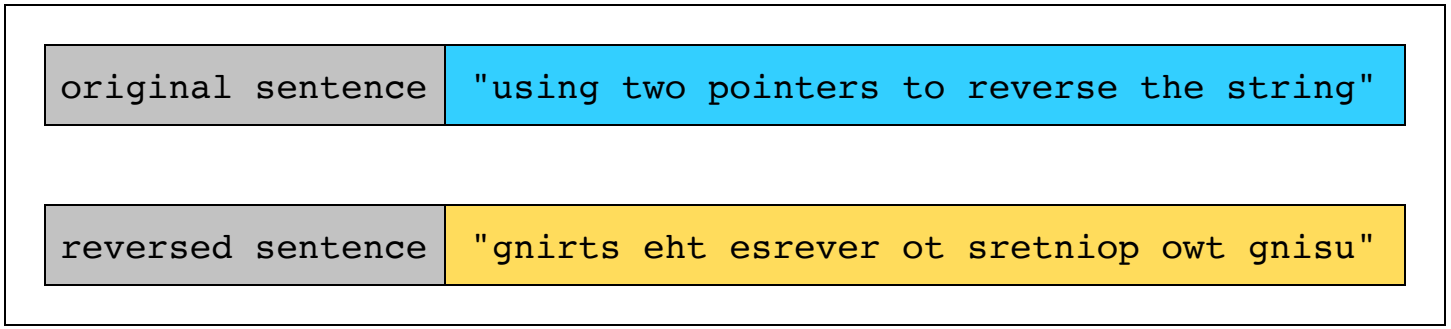
Solution

The essence of this algorithm is to reverse the words in a string by utilizing a two-step method using two pointers, effectively manipulating the string in place. The entire string is initially reversed, which correctly repositions the words in reverse order. But due to this reversal of the string, the corresponding characters of words are also reversed. To correct the order of the characters, iterate through the string and identify the start and end of each word, considering space as a delimiter between words. Set two pointers at the start and end of the word to reverse the characters within that word. By repeating this process for each word, the method effectively achieves the goal without requiring additional memory.

> **Note:** The solution mimics an in-place reversal by converting the string to a mutable list of characters, modifying it in place, and then joining it back into a string.

Now let's look at an example to understand the detailed workflow of this algorithm:

1. **Reverse the entire string:** First, we reverse the entire string. This action places the words in the desired order, but each word is itself reversed (i.e., the characters in each word are in the reverse order).



2. **Reverse each word individually:** Next, we go through the string and reverse each word individually to correct their character order.

Here's how we can implement the approach above:

1. **Reversing the entire sentence:**
    I. We first reverse the complete string. This step places the words in their correct positions relative to each other, although each word is backward.
2. **Reversing each word:**

I. Iterate through the reversed sentence:

   i. Use two pointers, `start` and `end`, both initially set to 0. The first pointer, `start`, marks the beginning of a word, and the second pointer, `end`, finds the end of a word.

   ii. When the end of a word is found (either a space or the end of the string), we reverse the characters in that word in place.

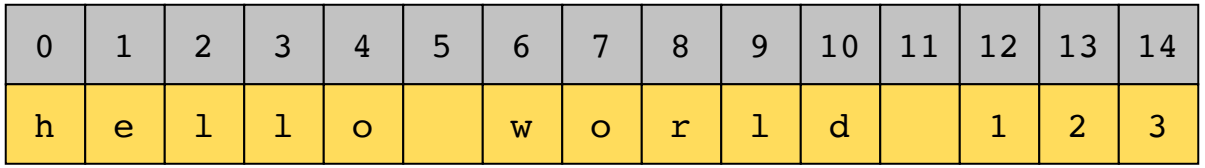
   iii. After reversing, we update `start` to point to the beginning of the next word.

   iv. Now, we'll repeat this process for the next word. At the end of all iterations, we get the reversed words in the string.

   v. We repeat this process for all words in the string.

The following illustration shows these steps in detail:

Let's reverse the sentence "hello world 123".



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
h	e	l	l	o		w	o	r	l	d		1	2	3

We can see the code of this solution below.

Solution | Commented Solution

```
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```
class Solution {
    // A function that reverses characters from startRev to endRev in place
    private static void strRev(char[] str, int startRev, int endRev) {
        while (startRev < endRev) {
            char temp = str[startRev];
            str[startRev] = str[endRev];
            str[endRev] = temp;
            startRev++;
            endRev--;
        }
    }

    public static String reverseWords(String sentence) {
        sentence = sentence.replaceAll("\\s+", " ").trim();

        char[] charArray = sentence.toCharArray();
        int strLen = charArray.length - 1;

        strRev(charArray, 0, strLen);

        for (int start = 0, end = 0; end <= strLen; ++end) {
            if (end == strLen || charArray[end] == ' ') {
                int endIdx = (end == strLen) ? end : end - 1;
```

Run | Ask AI Mentor | Save | Reset

Reverse Words in a String

Time complexity

It takes $O(n)$ time to remove the leading and trailing whitespace and replace multiple spaces with a single space in the sentence, where n is the length of the sentence. Additionally, the array is traversed twice in

$O(n + n) = O(n)$ time. Therefore, the overall time complexity of this solution is $O(n)$.