<https://leetcode.com/problems/reverse-vowels-of-a-string/>

Given a string s, reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.

**Example 1:**

Input: s = "hello"

Output: "holle"

**Example 2:**

Input: s = "leetcode"

Output: "leotcede"

**Constraints:**

1 <= s.length <= 3 \* 10^5

s consist of **printable ASCII** characters.

**Attempt 1: 2023-02-22**

**Solution 1: Two Pointers (10 min)**

class Solution {

    public String reverseVowels(String s) {

        char[] chars = s.toCharArray();

        int i = 0;

        int j = chars.length - 1;

        while(i < j) {

            if(isVowel(chars[i]) && isVowel(chars[j])) {

                swap(chars, i, j);

                i++;

                j--;

            } else if(!isVowel(chars[i]) && isVowel(chars[j])) {

                i++;

            } else if(isVowel(chars[i]) && !isVowel(chars[j])) {

                j--;

            } else {

                i++;

                j--;

            }

        }

        return new String(chars);

    }

    private void swap(char[] chars, int i, int j) {

        char tmp = chars[i];

        chars[i] = chars[j];

        chars[j] = tmp;

    }

    private boolean isVowel(char c) {

        if(c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' || c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {

            return true;

        }

        return false;

    }

}

Time Complexity: O(n)

Space Complexity: O(n), in Java, we need to convert the string to a char array that would take O(N) space,

and therefore the space complexity for Java would be O(N).

**Refer to**

<https://leetcode.com/problems/reverse-vowels-of-a-string/solutions/2484211/reverse-vowels-of-a-string/>

**Overview**

This problem is an extension to the problem [344 Reverse String](https://leetcode.com/problems/reverse-string/). In this problem, we have to reverse only the vowels instead of every character as in the original problem.

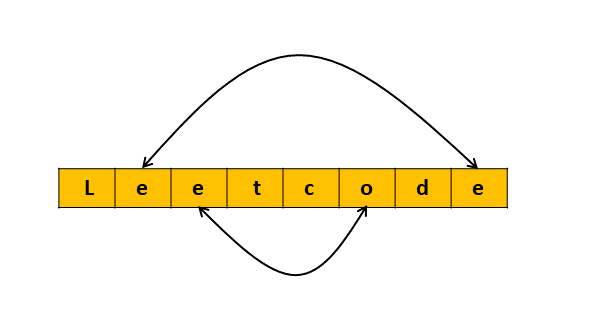
**Approach 1: Two Pointers**

**Intuition**

we will initialize two pointers, one pointer (referred as left) pointing to the left end of the input string and the other pointer (named as right) pointing to the right end of the string.

The only difference compared to the problem [344 Reverse String](https://leetcode.com/problems/reverse-string/) is that we don't want to swap all characters instead we want to swap just the vowels. So the left and right pointers as we discussed should be pointing to the vowels only.

To achieve this, we would initialize a left pointer to 0 and keep incrementing it until we get a vowel. Similarly, we initialize the right pointer to the last index and keep decrementing it until it points to a vowel. At each such iteration where both the pointers are pointing to the vowel, we would swap the characters at these pointers.



**Algorithm**

Initialize the left pointer start to 0, and the right pointer end to s.size() - 1.

Keep iterating until the left pointer catches up with the right pointer:

Keep incrementing the left pointer start until it's pointing to a vowel character.

Keep decrementing the right pointer end until it's pointing to a vowel character.

Swap the characters at the start and end.

Increment the start pointer and decrement the end pointer.

Return the string s.

**Implementation**

class Solution {

    // Return true if the character is a vowel (case-insensitive)

    boolean isVowel(char c) {

        return c == 'a' || c == 'i' || c == 'e' || c == 'o' || c == 'u'

            || c == 'A' || c == 'I' || c == 'E' || c == 'O' || c == 'U';

    }

    // Function to swap characters at index x and y

    void swap(char[] chars, int x, int y) {

        char temp = chars[x];

        chars[x] = chars[y];

        chars[y] = temp;

    }

    public String reverseVowels(String s) {

        int start = 0;

        int end = s.length() - 1;

        // Convert String to char array as String is immutable in Java

        char[] sChar = s.toCharArray();

        // While we still have characters to traverse

        while (start < end) {

            // Find the leftmost vowel

            while (start < s.length () && !isVowel(sChar[start])) {

                start++;

            }

            // Find the rightmost vowel

            while (end >= 0 && !isVowel(sChar[end])) {

                end--;

            }

            // Swap them if start is left of end

            if (start < end) {

                swap(sChar, start++, end--);

            }

        }

        // Converting char array back to String

        return new String(sChar);

    }

};

**Complexity Analysis**

Here, N is the length of the string s.

Time complexity: O(N)

It might be tempting to say that there are two nested loops and hence the complexity would be O(N^2). However, if we observe closely the pointers start and end will only traverse the index once. Each element of the string swill be iterated only once either by the left or right pointer and not both. We swap characters when both pointers point to vowels which are O(1) operation. Hence the total time complexity will be O(N).

Note that in Java we need to convert the string to a char array as strings are immutable and hence it would take O(N) time.

Space complexity: O(N)

In C++ we only need an extra temporary variable to perform the swap and hence the space complexity is O(1). However, in Java, we need to convert the string to a char array that would take O(N) space, and therefore the space complexity for Java would be O(N).