2785. Sort Vowels in a String

Medium String Sorting

Problem Description

The problem requires us to modify a given string `s` by permuting only the vowels, such that the vowels are sorted based on their ASCII values, while all the consonants remain in their original positions. Both uppercase and lowercase vowels ('a', 'e', 'i', 'o', 'u') should be considered, but it's important to note that consonants account for any letter that is not a vowel. The objective is to return the new string after the vowels have been sorted and the consonants are left untouched.

For example, if `s` is "leetcode", the output should be "leotcede" because the vowels 'e','e','e','o' in `s` are sorted to 'e','e','o','e' in the new string `t`.

Intuition

The key intuition behind the solution is to separate the vowels from the consonants, sort the vowels according to their ASCII values, and then merge them back into the original string in their proper index locations.

1. **Separating Vowels from Consonants**: We go through the string `s` and create a list of vowels. This is done by checking if each character is a vowel (for simplicity, by checking if it's in the string "aeiou" after converting to lowercase to ensure that both uppercase and lowercase vowels are considered).
2. **Sorting Vowels**: Once we have a list that contains only the vowels from the original string, we sort this list. This sorted list now represents the order that the vowels should appear in the final string.
3. **Merging Vowels Back**: Keeping a separate copy of the original string allows us to know the positions of the consonants, so we can replace the vowels in this copy with the sorted vowels. We iterate through the copy of the original string and each time we encounter a vowel, we take the next vowel from our sorted vowels list and replace it.
4. **Converting to String**: Finally, we join the list into a string and return this as our final sorted string with the vowel positions permuted according to their ASCII values and consonants in their initial places.

Solution Approach

The solution approach for sorting the vowels in the string while keeping the consonants in place involves a few clear steps combining algorithmic thinking and data structures.

Algorithm:

1. **Collecting Vowels**: Iterate over the input string `s` and build a list of vowels called `vs`. This is achieved using a list comprehension that includes a character `c` only if `c.lower()` is found within the string "aeiou". This step effectively filters out consonants.
2. **Sorting Vowels**: Once we have a list of all the vowels from the string, sort this list using Python's built-in `sort()` method. The sorted `vs` list now contains the vowels in the order they should appear in the resulting string based on their ASCII values.
3. **Preparing for Re-insertion**: Copy the original string `s` into a list `cs` which will allow modifying individual characters at specific indices (since strings in Python are immutable).
4. **Inserting Sorted Vowels**: We will use two pointers—`i` iterating over the `cs` list which represents the original string's characters, and `j` which keeps track of the position in the sorted vowels list `vs`. When we encounter a vowel (as determined by `c.lower() in "aeiou"`) in `cs` at the current index `i`, we replace it with the vowel at the current index `j` from the sorted list `vs`. We then increment `j` to move to the next sorted vowel.
5. **Joining the List**: After iterating through the entire list and replacing vowels with their sorted counterparts, we join the list `cs` back into a string using `"".join(cs)`, which gives us the final string where vowels are sorted, and consonants remain in their original places.

Data Structures:

- **List for Vowels**: We use a list to keep all the vowels from the original string because lists in Python are mutable and can be sorted easily.
- **List for Modifying String**: A list (`cs`) copying the original string `s` is also created to modify the characters in-place, as strings in Python are immutable and cannot be changed after creation.
- **For-Loop with Enumeration**: The use of `enumerate` on `cs` provides both index and character in a single loop, which is efficient for tracking positions for vowel replacement.
- **Index Pointer (`j`)**: A counter variable `j` is used to traverse the `vs` list while placing sorted vowels back into `cs`.

Code Reference:

```
class Solution:
    def sortVowels(self, s: str) -> str:
        vs = [c for c in s if c.lower() in "aeiou"]  # Collecting vowels
        vs.sort()  # Sorting[](problems/sorting_summary) vowels
        cs = list(s)  # Copy string to a list
        j = 0  # Pointer for sorted vowels list
        for i, c in enumerate(cs):  # Loop through characters in original string
            if c.lower() in "aeiou":  # If character is a vowel
                cs[i] = vs[j]  # Replace with sorted vowel
                j += 1  # Move to next vowel in sorted list
        return "".join(cs)  # Return modified string as output
```

This code snippet clearly follows the approach and uses the necessary data structures to accomplish the task.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach using the string `s = "contribute"`.

1. **Collecting Vowels**: We go through each character in "contribute" and collect the vowels in the list `vs`. After iterating over the string, `vs` becomes `['o', 'i', 'u', 'e']`.
2. **Sorting Vowels**: Sorting the list `vs` gives us `['e', 'i', 'o', 'u']`.
3. **Preparing for Re-insertion**: We convert the original string `s` into a list `cs` which will let us modify individual characters. So, `cs` becomes `['c', 'o', 'n', 't', 'r', 'i', 'b', 'u', 't', 'e']`.
4. **Inserting Sorted Vowels**: As we iterate over `cs`, when we find a vowel, we replace it with the next vowel from the sorted list `vs`. After processing, `cs` now looks like `['c', 'e', 'n', 't', 'r', 'i', 'b', 'o', 't', 'u']`.
5. **Joining the List**: Finally, we join `cs` back into a string to get the output `ceotributn`.

Following these steps with our example, the original string `contribute` transforms into `ceotributn` where the vowels appear in sorted order based on their ASCII values while all consonants remain in their original positions.

Solution Implementation

Python

```
class Solution:
    def sortVowels(self, s: str) -> str:
        # Initialize an array to hold the vowels from the string
        vowels = [c for c in s if c.lower() in "aeiou"]

        # Sort the vowels array in alphabetical order
        vowels.sort()

        # Convert the input string to a list to enable modifications
        characters = list(s)

        # Initialize a counter for the vowels array index
        vowel_index = 0

        # Iterate through the characters of the string
        for i, c in enumerate(characters):
            # Check if the character is a vowel
            if c.lower() in "aeiou":
                # Replace the vowel in the characters array with the sorted one
                characters[i] = vowels[vowel_index]
                # Increment the vowel index to move to the next sorted vowel
                vowel_index += 1

        # Join the characters back to form the modified string and return
        return "".join(characters)
```

Java

```
class Solution {
    // Method to sort vowels in a given string
    // Vowels in the original string are replaced with vowels in sorted order
    public String sortVowels(String s) {
        // List to store vowels
        List<Character> vowels = new ArrayList<>();
        // Convert the string to a character array
        char[] chars = s.toCharArray();
        // Iterate over the character array
        for (char c : chars) {
            // Convert character to lower case to handle both cases
            char lowerCase = Character.toLowerCase(c);
            // Check if the character is a vowel
            if (lowerCase == 'a' || lowerCase == 'e' || lowerCase == 'i' || lowerCase == 'o' || lowerCase == 'u') {
                // Add vowel to the list
                vowels.add(c);
            }
        }
        // Sort the list of vowels
        Collections.sort(vowels);
        // Initialize an index to keep track of sorted vowels
        int vowelIndex = 0;
        // Replace vowels in the original array with vowels in sorted order
        for (int i = 0; i < chars.length; ++i) {
            // Convert character to lower case to handle both cases
            char lowerCase = Character.toLowerCase(chars[i]);
            // Check if the character is a vowel
            if (lowerCase == 'a' || lowerCase == 'e' || lowerCase == 'i' || lowerCase == 'o' || lowerCase == 'u') {
                // Replace the vowel with a sorted vowel from the list
                chars[i] = vowels.get(vowelIndex++);
            }
        }
        // Convert character array back to string and return
        return String.valueOf(chars);
    }
}
```

C++

```
#include <algorithm> // Include algorithm header for std::sort
#include <cctype>    // Include cctype header for std::tolower

class Solution {
public:
    // Function to sort vowels in a given string `s`
    string sortVowels(string s) {
        string vowels; // Initialize a string to store the found vowels

        // Iterate over each character in the input string
        for (auto& c : s) {
            // Convert each character to lowercase for comparison
            char lowerCaseChar = std::tolower(c);

            // Check if the character is a vowel
            if (lowerCaseChar == 'a' || lowerCaseChar == 'e' ||
                lowerCaseChar == 'i' || lowerCaseChar == 'o' ||
                lowerCaseChar == 'u') {

                vowels.push_back(c); // Add the vowel to the 'vowels' string
            }
        }

        // Sort the vowels alphabetically
        std::sort(vowels.begin(), vowels.end());

        // Replace the vowels in the original string with sorted vowels
        for (int i = 0, vowelIndex = 0; i < s.size(); ++i) {
            // Check if the current character is a vowel
            char lowerCaseChar = std::tolower(s[i]);
            if (lowerCaseChar == 'a' || lowerCaseChar == 'e' ||
                lowerCaseChar == 'i' || lowerCaseChar == 'o' ||
                lowerCaseChar == 'u') {

                // Replace the vowel with the sorted vowel from 'vowels'
                s[i] = vowels[vowelIndex++];
            }
        }

        return s; // Return the modified string with sorted vowels
    }
};
```

TypeScript

```
// This function sorts the vowels in a given string while keeping the consonants in their original position
function sortVowels(inputString: string): string {
    // Define an array of all vowels, both lowercase and uppercase
    const vowels: string[] = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'];

    // Split the input string into characters, filter out vowels, and sort them
    const sortedVowels: string[] = inputString
        .split('')
        .filter(character => vowels.includes(character))
        .sort();

    // Create an array to hold the final characters in the correct order
    const sortedCharacters: string[] = [];

    // Index for iterating over the sortedVowels
    let vowelIndex: number = 0;

    // Iterate over each character of the input string
    for (const character of inputString) {
        // If the character is a vowel, use the vowel from sortedVowels (preserving original order elsewise)
        sortedCharacters.push(vowels.includes(character) ? sortedVowels[vowelIndex++] : character);
    }

    // Join the sorted characters array into a string and return it
    return sortedCharacters.join('');
}

// Usage
const input = "LeetCode";
const sorted = sortVowels(input);
console.log(sorted); // Expected output would have vowels sorted within the string
```

```
class Solution:
    def sortVowels(self, s: str) -> str:
        # Initialize an array to hold the vowels from the string
        vowels = [c for c in s if c.lower() in "aeiou"]

        # Sort the vowels array in alphabetical order
        vowels.sort()

        # Convert the input string to a list to enable modifications
        characters = list(s)

        # Initialize a counter for the vowels array index
        vowel_index = 0

        # Iterate through the characters of the string
        for i, c in enumerate(characters):
            # Check if the character is a vowel
            if c.lower() in "aeiou":
                # Replace the vowel in the characters array with the sorted one
                characters[i] = vowels[vowel_index]
                # Increment the vowel index to move to the next sorted vowel
                vowel_index += 1

        # Join the characters back to form the modified string and return
        return "".join(characters)
```

Time and Space Complexity

Time Complexity

The provided Python code consists of the following operations:

1. **Creating a list of vowels (`vs`)**: This involves iterating over each character in the string `s` to check if it is a vowel, which takes `O(n)` time where `n` is the length of the string.
2. **Sorting the list of vowels (`vs.sort()`)**: The time complexity for sorting in Python (using Timsort) is `O(n log m)` where `n` is the number of vowels in the string, which is at most `n`.
3. **Iterating over the string characters and replacing vowels (`for i, c in enumerate(cs)`)**: We iterate over the list `cs` once, which takes `O(n)` time. For each vowel, we perform a constant-time operation.

Combining these steps, the overall time complexity is `O(n) + O(n log m) + O(n)`. Since `m` is at most `n`, the time complexity can be simplified to `O(n) + O(n log n)`, which is dominated by the sorting step, leading to a final time complexity of `O(n log n)`.

Space Complexity

The space complexity is determined by the additional space used by the algorithm, not including the input itself:

1. **List of vowels (`vs`)**: At most `n` if the string consists only of vowels, so `O(n)` space.
2. **List of characters from the string (`cs`)**: We create a list of all characters, which also takes `O(n)` space.
3. **The sorted list of vowels does not use extra space** because the sorting is done in-place on the `vs` list.

Therefore, the combined space complexity is `O(n)` for storing the vowels and `O(n)` for storing the character array, which totals `O(2n)`. Simplifying this gives us `O(n)` space complexity.