Replace Spaces CodeStudio

The goal is to replace all spaces in a given input string with the string "@40". Three different approaches are provided to accomplish this task: **replaceSpaces**, **replaceSpacesUsingTempString**, and **replaceSpacesOptimized**. The task is to analyze each approach, explain the code, and provide their time and space complexity. Additionally, guidance will be given on which approach to use.

Example: Input: "Hello World! This is a new input string."

Output: "Hello@40World!@40This@40is@40a@40new@40input@40string."

Approach 1: This function replaces spaces in the input string with the string "@40" using the replace function.

- This approach iterates over the input string character by character.
- If a space character is found, it uses the **replace** function of the **string** class to replace it with the string "@40".
- **Time Complexity: O(n^2), where n is the length of the input string. In the worst case, each replacement operation takes O(n) time due to shifting of characters after the replacement.**
- **Space Complexity: O(1) as the space used is constant, regardless of the size of the input string.**

Approach 2: This function replaces spaces in the input string with the string "@40" using a temporary string.

- This approach creates a temporary string **temp** to store the modified string.
- It iterates over each character of the input string.
- If a space character is encountered, it appends the string "@40" to **temp**, otherwise, it appends the current character.
- Finally, it assigns the modified string back to the input string **str**.
- **Time Complexity: O(n), where n is the length of the input string. The loop iterates over each character in the string once.**
- **Space Complexity: O(n), where n is the length of the input string. The additional space is used to store the modified string in temp.**

Approach 3: This function optimizes the space usage and time complexity for replacing spaces in the input string.

- This approach optimizes the time complexity by counting the number of spaces in the input string upfront.
- It then calculates the new length of the string after replacing spaces.
- Memory is reserved in the input string to accommodate the additional characters.
- It replaces spaces with "@40" in-place, by shifting characters only once.
- **Time Complexity: O(n), where n is the length of the input string. It performs two passes over the string: one to count the spaces and another to replace them. The replacement is done by shifting characters only once.**
- **Space Complexity: O(1) as the space used is constant, regardless of the size of the input string. Although str.reserve(newLength) reserves additional memory, it is not considered in the space complexity calculation since it is a one-time allocation.**

Which approach to use:

- If the string is small or the time complexity is not a concern, **replaceSpaces** or **replaceSpacesUsingTempString** can be used.
- If efficiency is a concern, **replaceSpacesOptimized** should be used as it provides the best time complexity of O(n).