Find the Longest Common Prefix [LeetCode](https://leetcode.com/problems/longest-common-prefix/description/)

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example: **Input:** strs = ["flower","flow","flight"]

**Output:** "fl"

**Input:** strs = ["dog","racecar","car"]

**Output:** ""

**Approach 1: Function to find the longest common prefix using the character-by-character comparison approach**

* **Functionality:**
  + Finds the longest common prefix among a set of strings by comparing characters at each position.
* **Explanation:**
  + Iterates through each character position in the first string.
  + Compares the character at the current position with the corresponding characters in other strings.
  + If a mismatch is found or if a string is shorter than the current position, breaks the loop.
  + Appends the matched character to the common prefix.
  + Returns the common prefix.
* **Time Complexity:**
  + **O(N \* M), where N is the length of the common prefix, and M is the number of strings.**
* **Space Complexity:**
  + **O(1) - Only a constant amount of space is used.**

**Approach 2: Function to find the longest common prefix using the Trie approach**

* **Classes:**
  + **TrieNode**: Represents a single node in the trie.
  + **Trie**: Represents the trie data structure.
* **Functions:**
  + **insert**: Inserts a word into the trie.
  + **longestCommonPrefix**: Finds the longest common prefix using the trie approach.
* **Explanation:**
  + Inserts each string into the trie.
  + Traverses the trie until a node has more than one child or is the end of a word.
  + Adds the characters to the common prefix during traversal.
  + Returns the common prefix.
* **Time Complexity:**
  + **O(N \* M), where N is the length of the longest word, and M is the number of strings.**
* **Space Complexity:**
  + **O(N \* M), where N is the total number of characters in all strings, and M is the number of strings.**

**Conclusion**

* Both approaches have the same time complexity (O(N \* M)), where N is the length of the common prefix, and M is the number of strings.
* The **Character-by-Character Comparison Approach** is more straightforward and may perform better for a small number of strings.
* The **Trie Approach** may be more efficient for a large number of strings as it uses a trie structure, but it comes with higher space complexity (O(N \* M)).