Leetcode-14

👨‍💻 What's the Goal?

This C++ code is for the **Leetcode problem: "Longest Common Prefix"**.

📜 **Given**: A list of strings.

🎯 **Goal**: Find the **longest common prefix** (LCP) string shared among all of them.

Example:

```
Input: ["flower", "flow", "flight"]
Output: "fl"
```

🔍 Line-by-Line Explanation

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
```

👆 Declares a class `Solution` and defines a method `longestCommonPrefix` which takes a vector of strings as input.

```
        int n = strs.size();
        if (strs.size() == 1) return strs[0];
```

📌 Checks the size of the input:

- If there's only **one string**, it **returns it directly** because it's obviously the common prefix with itself.

```
        sort(strs.begin(), strs.end());
```

🔀 **Key move**: The strings are **sorted lexicographically** (i.e., dictionary order).

Why?

After sorting:

- The **first** and **last** strings in the vector will be the **most different**.
- So, the common prefix of **these two** is guaranteed to be the common prefix of the **entire array**.

Example:

Before sort: `["flower", "flow", "flight"]`

After sort: `["flight", "flow", "flower"]`

So we now only need to compare `"flight"` and `"flower"`.

```
string first = strs[0];
string last = strs[n - 1];
string s = "";
```

Stores the **first** and **last** strings (after sorting), and initializes an empty string `s` to build the result.

```
for (int i = 0; i < (min(first.size(), last.size())); i++) {
    if (first[i] == last[i]) {
        s += first[i];
    }
    else return s;
}
```

👉 **Compare character-by-character** of the first and last strings:

- If the characters match → add to result `s`
- Else → break and return the prefix found so far

This works because only the **first few characters may be common**, and as soon as a mismatch is found, we know the prefix ends.

```
        return s;
    }
};
```

Finally, return the prefix `s`.

⚙️ Time Complexity

- Sorting: `O(N log N)` where N = number of strings
- Comparing first & last strings: `O(M)` where M = length of shortest string

💡 In practice, this is often fast for small-to-medium input sizes.

🧠 Summary

This code:

1. Sorts the strings
2. Compares only the first and last strings
3. Finds the common prefix between those two — which is also the prefix for the rest

---