LeetCode 14 — Longest Common Prefix

1. Problem Title & Link

- **Title:** LeetCode 14 — Longest Common Prefix
- **Link:** https://leetcode.com/problems/longest-common-prefix/

2. Problem Statement (Short Summary)

Given an array of strings strs, find the **longest prefix** that is **common to all strings**.

Example:

["flower", "flow", "flight"] → "fl"

If no common prefix exists, return "".

3. Examples (Input → Output)

Example 1

Input: ["flower","flow","flight"]

Output: "fl"

Example 2

Input: ["dog","racecar","car"]

Output: ""

Example 3

Input: ["a"]

Output: "a"

4. Constraints

- 1 <= strs.length <= 200
- 0 <= strs[i].length <= 200
- All strings contain lowercase English letters.

5. Core Concept (Pattern / Topic)

String Scanning + Horizontal Matching

We compare the prefix with each string and shrink it until it matches.

This is a **simple but powerful string pattern** used in interviews.

6. Thought Process (Step-by-Step Explanation)

❌ **Brute Force**

Try all substrings of first string → check each with all strings → O(n * m^2). Slow.

✔ **Optimal (Horizontal Scan)**

1. Take prefix = first string
2. Compare with next string
3. Shrink prefix until the next string starts with it
4. Continue for all strings
5. Remaining prefix is the answer

Why it works:

- Prefix can only shrink, never grow
- Efficient: at most all characters compared once

7. Visual / Intuition Diagram

For:

["flower", "flow", "flight"]

Start:

prefix = "flower"

Compare with "flow":

flower ❌

flowe ❌

flow ✔ → prefix = "flow"

Compare with "flight":

flow ❌

flo ❌

fl ✔ → prefix = "fl"

Final answer: "fl"

8. Pseudocode

```
prefix = strs[0]

for each string s in strs[1:]:
    while s does not start with prefix:
        remove last character from prefix
        if prefix is empty:
            return ""

return prefix
```

9. Code Implementation

✅ Python

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        prefix = strs[0]

        for s in strs[1:]:
            while not s.startswith(prefix):
                prefix = prefix[:-1]
                if prefix == "":
                    return ""
        return prefix
```

✅ Java

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        String prefix = strs[0];

        for (int i = 1; i < strs.length; i++) {
            while (!strs[i].startsWith(prefix)) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) return "";
            }
        }
        return prefix;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(n × m) where m = length of shortest string
Space	O(1)

Fast and efficient.

11. Common Mistakes / Edge Cases

❌ Not handling empty string array

❌ Not returning "" if prefix becomes empty

❌ Comparing character by character manually (slower)

❌ Forgetting to update prefix correctly

Edge cases:

- [""] → output ""
- ["a"] → output "a"
- ["abc", ""] → output ""

12. Detailed Dry Run (Step-by-Step Table)

Input:

["flower", "flow", "flight"]

Step	Compare With	Current Prefix	Action
Start	—	"flower"	Initial
1	"flow"	"flower"	Not match → shrink
		"flowe"	Not match
		"flow"	Match → keep
2	"flight"	"flow"	Not match →
		"flo"	Not match
		"fl"	MATCH ✔
End	—	"fl"	Answer

Output:

"fl"

13. Common Use Cases

- URL prefix matching
- File path prefix grouping
- Autocomplete systems
- Matching version prefixes

14. Common Traps

- Shrinking wrong side (must shrink end)
- Using substring inefficiently in loops
- Not checking start-with condition correctly

15. Builds To (Related Problems)

- LC 28 — StrStr (string search)
- LC 58 — Length of Last Word

- LC 242 — Valid Anagram
- LC 49 — Group Anagrams

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Horizontal Scanning	O(n × m)	O(1)	Best for interviews
Vertical Scanning	O(n × m)	O(1)	Also good
Sorting + Compare First & Last	O(n log n)	O(1)	Very elegant
Trie (Prefix Tree)	O(n × m)	O(m)	Overkill for small constraints

17. Why This Solution Works (Short Intuition)

The longest common prefix must be a prefix of the first string.
By shrinking it while checking all strings, we guarantee correctness.

18. Variations / Follow-Up Questions

- What if we need **longest common suffix**?
- What if we need **common prefix among millions of strings**?
- How to solve with **Trie**?
- How to solve if strings are **very large**?