Objective
To enhance problem-solving skills by exploring various scenarios associated with coding challenges and implementing efficient solutions with optimal complexity.

Approach

1. **Understanding the Problem:**
   This involves comprehensively grasping what the problem is asking. Identify the input, desired output, constraints, requirements, and any specific details provided in the problem statement.
2. **Getting the Hint:**
   Sometimes, understanding a problem requires looking for hints or clues within the problem statement itself. This may involve identifying keywords, patterns, or relationships that can guide your approach.
3. **Understanding Scenarios, Boundary Conditions, and Edge Cases:**
   - **Scenarios:** Consider all possible situations or conditions that the problem may present. Break down the problem into smaller parts or cases to understand how the solution may vary.
   - **Boundary Conditions:** Identify the limits or extreme values of inputs that may affect the solution. Understanding these helps ensure correctness under all circumstances.
   - **Edge Cases:** Similar to boundary conditions, edge cases refer to extreme or unconventional situations not covered by typical scenarios. Identifying and addressing these ensures robustness and reliability.
4. **Solving Scenarios:**
   Once you've identified different scenarios, boundary conditions, and edge cases, proceed to solve each scenario individually. Apply various techniques, algorithms, or approaches based on the nature of the problem.
5. **Understanding the Logic:**
   Finally, it's essential to understand the underlying logic behind the solution. Grasp why certain approaches or techniques were chosen, how they address the problem requirements, and how they ensure correctness, efficiency, and scalability.

Problem Statements to Solve:

1. **Longest Common Prefix:**
   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 "".
   Examples:
   - **Input:** strs = ["flower", "flow", "flight"]
     Output: "fl"
   - **Input:** strs = ["dog", "racecar", "car"]
     Output: ""
     Explanation: There is no common prefix among the input strings.
2. **Constraints:**
   1 <= strs.length <= 200
   0 <= strs[i].length <= 200
   strs[i] consists of only lowercase English letters.

3. **Incrementing a Large Integer:**
   You are given a large integer represented as an integer array `digits`, where each `digits[i]` is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. Increment the large integer by one and return the resulting array of digits.
   Examples:
   - **Input:** digits = [1, 2, 3]
     Output: [1, 2, 4]
     Explanation: The array represents the integer 123. Incrementing by one gives 123 + 1 = 124, resulting in [1, 2, 4].
   - **Input:** digits = [4, 3, 2, 1]
     Output: [4, 3, 2, 2]
     Explanation: The array represents the integer 4321. Incrementing by one gives 4321 + 1 = 4322, resulting in [4, 3, 2, 2].

4. **Longest Substring Without Repeating Characters:**

- **Input:** "ABCBC"
  Output: 3
  Explanation: The longest substring without repeating characters is "ABC", which has a length of 3.
- **Input:** "AAA"
  Output: 1
  Explanation: The longest substring without repeating characters is "A", which has a length of 1.

5. **Longest Valid Parentheses Substring:**
   - **Input:** "((()"
     Output: 2
     Explanation: The longest valid parentheses substring is (), which has a length of 2.
   - **Input:** ")()())"
     Output: 4
     Explanation: The longest valid parentheses substring is ()(), which has a length of 4

6. **Sort an Array of 0s, 1s, and 2s:**
   This is a typical array rearrangement problem where we need to sort elements based on their values (0, 1, and 2). While this can be solved using sorting approaches (with a complexity of O(n log n)), consider more efficient solutions.
   Examples:
   - **Input:** [0, 1, 2, 1, 0, 2, 1, 0]
     Output: [0, 0, 0, 1, 1, 1, 2, 2]
   - **Input:** [2, 1, 0, 1, 2, 0, 1, 0]
     Output: [0, 0, 0, 1, 1, 1, 2, 2]

7. **Smallest Window with All Distinct Characters:**
   Given a string, find the smallest window length that contains all distinct

characters of the string. For example, for `str = "aabcbcdbca"`, the result would be 4, as the smallest window is "dbca".
Examples:

- **Input:** "abca"
  Output: 4
- **Input:** "aaaa"
  Output: 1
  Explanation: In Example 1, the string contains all distinct characters, so the smallest window is the entire string.
- In Example 2, the string contains only one distinct character, so the smallest window is just "a".

So, at the end of Day 1, you should be able to:

1. Identify various inputs / outputs
2. Pseudo Code
3. Logical Implementation with explanation