1.Easy\06.Check_for_rotated_string.cpp

```cpp
/*
QUESTION: Rotate String
Given two strings s and goal, return true if and only if s can become goal after some number of shifts on s.

A shift on s consists of moving the leftmost character of s to the rightmost position.

Example:
Input: s = "abcde", goal = "cdeab"
Output: true

Input: s = "abcde", goal = "abced"
Output: false

Approach:
- First, we check if the lengths of the two strings `s` and `goal` are equal. If not, they cannot be rotated versions of each other, so we return `false`.
- Then, we concatenate `s` with itself to create a new string `concat`.
- We check if `goal` is a substring of `concat`. If it is, that means `s` can be transformed into `goal` by performing some number of left shifts, so we return `true`. Otherwise, we return `false`.

CODE:-
*/

bool rotateString(string s, string goal) {
    if (s.size() != goal.size())
        return false;
    if ((s + s).find(goal) == string::npos)
        return false;
    return true;
}

/*
Time Complexity: The time complexity of this approach is O(N^2), where N is the length of the input strings `s` and `goal`. This is because the `find` function is used to search for the substring `goal` within the concatenated string, which has a time complexity of O(N^2).
Space Complexity: The space complexity is O(N), where N is the length of the input string `s`. This is because we create a new string `concat` by concatenating `s` with itself.
*/

```