QUESTION 1:

Easy 1

Given a string s consisting of words and spaces, return the length of the last word in the string. A word is a maximal substring consisting of non-space characters only.

Example 1: Input: s = "Hello World" Output: 5 Explanation: The last word is "World" with length 5.

Example 2: Input: s = "fly me to the moon" Output: 4 Explanation: The last word is "moon" with length 4.

Example 3: Input: s = "luffy is still joyboy" Output: 6 Explanation: The last word is "joyboy" with length 6.

Constraints: --> 1 <= s.length <= 104 --> s consists of only English letters and spaces ' '. --> There will be at least one word in s.

PYTHON CODE:

In [ ]:

```
s=input()
k= s.split(" ")
print(len(k[-1]))
```

ALGORITHM:

1. Accept a string input from the user and store it in variable s.
2. Split the input string s into a list of words using space as the delimiter and store it in variable k.
3. Retrieve the last word in the list k using k[-1].
4. Calculate the length of the last word using the len() function.
5. Print the length of the last word as the output.

LOGIC:

1. Input: The user provides a string input.
2. Splitting: The split(" ") function divides the string into a list of words wherever it encounters a space. For instance, if the input is "Hello there, how are you?", it'll be split into ["Hello","there,", "how", "are", "you?"].
3. Identifying the Last Word: k[-1] refers to the last element in the list k, which is the last word in the original string.
4. Measuring Length: len(k[-1]) calculates the length of the last word in the input string.
5. Display : The length of the last word is then printed as the output.

QUESTION 2:

Medium 3

Constraints: m == matrix.length n == matrix[i].length 1 <= m, n <= 300 matrix[i][j] is '0' or '1'.

Given an m x n binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

Example 1: Input: matrix = [["0"]] Output: 0

Example 2: Input: matrix = [["0","1"],["1","0"]] Output: 1

Example 3: Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]] Output: 4

PYTHON CODE:

In [ ]:

```python
def find(matrix):
    if not matrix or not matrix[0]:
        return 0

    m, n = len(matrix), len(matrix[0])
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    max_side = 0

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if matrix[i - 1][j - 1] == "1":
                dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]) + 1
                max_side = max(max_side, dp[i][j])

    return max_side ** 2
```

LOGIC AND ALGORITHM:

1. Starting Point: You have a grid of 1s and 0s representing a matrix.
2. Checking Matrix: First, it ensures the matrix exists and isn't empty.
3. Dynamic Programming Array: Creates an extra grid (DP array) that's a bit larger than the original matrix and fills it with zeros.
4. Traversing the Matrix: Goes through each cell of the matrix (starting from index 1, not 0).
5. For Each '1' Cell: If the cell contains '1', it tries to form a square using this cell as the bottom-right corner.

   Looks at the nearby cells (above, left, and diagonally top-left), checks their values, and calculates a potential square size at the current cell.

6. Determining Square Size: Increases this potential square size by 1 to represent the current cell's square size, ensuring the square can be formed.

7. Tracking Maximum Side: Keeps track of the largest square's side encountered while traversing.

8. Return: Finally, it returns the area (size) of the largest square found by squaring the value of the maximum side. This represents the largest square area in the matrix consisting of '1's.

QUESTION 3:

Hard 2

You are given a string s. You can convert s to a palindrome by adding characters in front of it. Return the shortest palindrome you can find by performing this transformation.

Example 1: Input: s = "aacecaaa" Output: "aaacecaaa"

Example 2: Input: s = "abcd" Output: "dcbabcd"

Constraints: 0 <= s.length <= 5 * 104 s consists of lowercase English letters only.

PYTHON CODE:

In [ ]:

```python
def shortestPalindrome(s):
    if not s:
        return ""

    rev_s = s[::-1]
    concat_str = s + "#" + rev_s

    # Construct the KMP table
    kmp_table = [0] * len(concat_str)
    j = 0

    for i in range(1, len(concat_str)):
        while j > 0 and concat_str[i] != concat_str[j]:
            j = kmp_table[j - 1]

        if concat_str[i] == concat_str[j]:
            j += 1

        kmp_table[i] = j

    return rev_s[:len(s) - kmp_table[-1]] + s


s = input()
print(shortestPalindrome(s))
```

LOGIC AND ALGORITHM:

1. Prepare for the Palindrome: First, get the letters or a word.
2. Reverse and Combine: Write these letters backward and join them with the original letters using a special mark (#), making a long mixed-up string.
3. Creating a Secret Table: Make a table to help us understand if there are any repeating patterns in this mixed-up string.
4. Magic Tablework: Look at each letter in this mixed-up string.

   If there Is a pattern, mark it down in the table.
5. Building the Shortest Palindrome: Use this table to figure out how many letters to add at the start of the original word.

   Grab these letters from the backward version and put them at the beginning of the original word.
6. Show Off the Result: Present this special word (a palindrome) made by adding letters to the start of the original word.