In [ ]:
```
Question 1
```

In [ ]:
```
Easy 1

Given a string s consisting of words and spaces, return the length of the last word ir
A word is a maximal
substring consisting of non-space characters only.
Constraints:
•1 <= s.length <= 104
•s consists of only English letters and spaces ' '.
•There will be at least one word in s.
```

CODE:

In [ ]:
```
def length_of_last_word(s):
    words = s.split()
    if not words:
        return 0
    return len(words[-1])
# Example usage:
s = "Hello World"
result = length_of_last_word(s)
print(result)
```

In [ ]:
```
EXPLANATION:
```

In [ ]:
```
Certainly! Let's break down the logic and algorithm of the code for finding the length
1. Split the String:
   - The first step is to split the input string into words. This is done using the `s
   - The `split` method without any arguments splits the string based on whitespace by
2. Access the Last Word:
   - Once the string is split, we can access the last word in the list.
   - In Python, indexing with `-1` refers to the last element in a list.
3. Calculate and Return Length:
   - After obtaining the last word, the code calculates its length using the `len` fur
   - The length of the last word is the desired result, so the function returns this l
4. Example:
   - Let's consider the string "Hello World". After splitting, we get the list `['Hell
   - Accessing the last element with `[-1]` gives us `'World'`.
   - The length of the last word, in this case, is 5, which is then returned by the fu
```

In [ ]:
```
Medium 1

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two gi
According to the definition of LCA on Wikipedia: “The lowest common ancestor is define
Constraints:
•The number of nodes in the tree is in the range [2, 105].
•-109 <= Node.val <= 109
•All Node.val are unique.
•p != q
•p and q will exist in the BST.
```

CODE:

In [ ]:
```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def lowest_common_ancestor(root, p, q):
    if p.val > q.val:
        p, q = q, p
    while root:
        if root.val > q.val:
            root = root.left
        elif root.val < p.val:
            root = root.right
        else:
            return root
# Example usage:
root = TreeNode(5)
root.left = TreeNode(3, TreeNode(2), TreeNode(4))
root.right = TreeNode(6, None, TreeNode(7))
# Find the LCA of nodes with values 2 and 4
p = TreeNode(2)
q = TreeNode(4)
result = lowest_common_ancestor(root, p, q)
print(result.val)
```

In [ ]:
```
EXPLANATION:
```

In [ ]:
```
To find the lowest common ancestor (LCA) of two nodes in a binary search tree (BST), y
1. Start from the Root:
   - Begin traversing the tree starting from the root.
2. Navigate Down the Tree:
   - At each node, compare the values of the given nodes `p` and `q` with the current
   - If both `p` and `q` are greater than the current node's value, it means both node
   - If both `p` and `q` are less than the current node's value, it means both nodes a
3. LCA Condition:
   - If one of `p` or `q` is smaller and the other is larger than the current node's v
   - This is because in a BST, the left subtree of a node contains values smaller thar
4. Repeat Until LCA is Found:
   - Keep traversing down the tree until you find the LCA.
```

In [ ]:
```
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.
Constraints:
•0 <= s.length <= 5 * 104
•s consists of lowercase English letters only.
```

CODE:

In [ ]:
```
def shortest_palindrome(s):
    if not s:
        return s
    extended = s + "#" + s[::-1]
    lps = [0] * len(extended)
    j = 0
```

```
for i in range(1, len(extended)):
        while j > 0 and extended[i] != extended[j]:
            j = lps[j - 1]
 if extended[i] == extended[j]:
            j += 1
 lps[i] = j
    palindrome_length = lps[-1]
    return s[palindrome_length:][::-1] + s
# Example usage:
s = "abcd"
result = shortest_palindrome(s)
print(result)
```

In [ ]:
```
EXPLANATION:
```

In [ ]:
```
To find the shortest palindrome by adding characters in front of a given string `s`, y

Here's the step-by-step explanation:
1. Create Modified String:
   - Construct a new string `new_s` by concatenating the reversed `s` with a special c
2. Build Prefix Array (KMP):
   - Build the prefix array (also known as the LPS array - Longest Proper Prefix which
3. Find Palindromic Prefix:
   - The value at the last index of the prefix array indicates the length of the longe
4. Construct Result:
   - The characters from index `len_palindrome` to the end in the reversed `s` need to
```