Find one extra character

Two strings are given of lengths n and n+1. The second string contains all the characters of the first string, but there is one extra character. The task is to find the extra character in the second string.

Note: The characters in strings can be in any order.

Examples :

Input: str1 = "abcd";
str2 = "cbdae";
Output: e
Explanation: str2 contains all the element of str1 with one extra alphabet 'e'.

Input: str1 = "kxml";
str2 = "klxml";
Output: l
Explanation: str2 contains all the element of str1 with one extra alphabet 'l'.

Method 1: Sorting

A simple method is to sort both of the arrays, and then start comparing characters one by one from start. If we found a mismatch, then the mismatched character from the second string is the answer as it has one extra character. If we don't find a mismatch, then the last character in the second string is the answer.

Note: Java doesn't provide a sort function for the String class. We need to convert the String to a character array and then use Arrays.sort().

```java
// Importing the Arrays class

import java.util.Arrays;



class GFG {

    public static char findExtra(String s1, String s2) {

        // Convert strings to character arrays

        char[] a1 = s1.toCharArray();

        char[] a2 = s2.toCharArray();



        // Sort both arrays

        Arrays.sort(a1);

        Arrays.sort(a2);



        // Compare characters

        int n = a1.length; // Length of the smaller string

        for (int i = 0; i < n; i++) {

            if (a1[i] != a2[i]) {

                return a2[i]; // Return the extra character

            }

        }
        // If no mismatch, the extra character is the last one
        return a2[n];
    }

    public static void main(String[] args) {
        String s1 = "abcd";
        String s2 = "abcde";
        System.out.println("Extra character: " + findExtra(s1, s2));
    }
}
```

Output

```
Extra character: e
```

Time Complexity: O(n*log*n).
Auxiliary Space: O(n).

Method 2: Counting

We can use frequency counting. We have assumed that the input strings have only lowercase English alphabets.

- Create a `count` array of size 26 to store the frequency of characters (assuming only lowercase alphabets).
- Traverse `s1` and decrement the count for each character.
- Traverse `s2` and increment the count for each character.
- The character with a count of `1` is the extra character.

Here's the code implementation of above approach:

```java
import java.util.*;

class GfG {

    static char findExtra(String s1, String s2)
    {
        int count[] = new int[26];

        int n = s1.length();

        for(int i=0; i<n; i++)
        {
            count[s2.charAt(i) - 'a']++;
            count[s1.charAt(i) - 'a']--;
        }
        count[s2.charAt(n) - 'a']++;

        for(int i=0; i<26; i++)
        {
            if(count[i] == 1)
                return (char)(i + 'a');
        }

        return 0;
    }

    public static void main(String args[])
    {
        String s1 = "abcd";
        String s2 = "cbdae";

        System.out.println(findExtra(s1, s2));
    }
}
```

Output

```
e
```

Time Complexity: O(N).
Auxiliary Space: O(1).

Method 3: Bitwise-XOR Operator

Using Bitwise Operators. If we combine all characters in both of the strings, we can see that every character will appear in multiples of 2 and there is only one character that will not have any duplicates.

- Initialize a variable `res` to store the XOR result.
- XOR all characters in `s1` and `s2`.
- The result after XORing will be the extra character, as XOR of two identical characters cancels them out.

Here's the code implementation of above approach:

```
import java.util.*;

class GfG {

    static char findExtra(String s1, String s2)
    {
        int res = 0;

        int n = s1.length();

        for(int i=0; i<n; i++)
        {
            res = res^s2.charAt(i)^s1.charAt(i);
        }

        res = res^s2.charAt(n);

        return (char)res;
    }

    public static void main(String args[])
    {
        String s1 = "abcd";
        String s2 = "cbdae";

        System.out.println(findExtra(s1, s2));
    }
}
```

Output

```
e
```

Time Complexity: O(N).

Auxiliary Space: O(1).