Check for Anagram

Given two strings, check whether two strings are an anagram of each other. Two strings are said to be an anagram of each other if they are just permutations of each other. That is, the set of characters in both the strings must be the same, only the order of characters can be different.

Examples:

Input: s1 = "listen"
s2 = "silent"
Output: Yes
Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = "aab"
s2 = "bab"
Output: No
Explanation: Characters in both the strings are not same, so they are not anagrams.

Method 1: Naive Solution

The naive solution involves sorting both strings and comparing the sorted results. If the sorted strings are identical, the two strings are anagrams. This approach is straightforward but not the most efficient due to the sorting step.

- Check if the lengths of the two strings are equal. If not, return `false`.
- Convert both strings into character arrays and sort them.
- Convert the sorted arrays back into strings and compare them.

- If the sorted strings are identical, the original strings are anagrams.

After sorting, we can use the equals() method to check if both of the strings are equal or not and return true or false accordingly.

Here's the code implementation of above approach:

```
import java.util.Arrays;

class GFG {
    public static boolean areAnagramNaive(String s1, String s2) {
        // Check if lengths of both strings are equal
        if (s1.length() != s2.length()) {
            return false;
        }
        // Convert first strings to character arrays and sort
        char[] a1 = s1.toCharArray();
        Arrays.sort(a1);
        s1 = new String(a1);
        // Convert second strings to character arrays and sort
        char[] a2 = s2.toCharArray();
        Arrays.sort(a2);
        s2 = new String (a2);
        // Compare sorted arrays
        return s1.equals(s2);
    }
    public static void main(String[] args) {
        String s1 = "abaac";
        String s2 = "aacba";

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

Output

```
true
```

Time Complexity: O(N*logN)
Auxiliary Space: O(1)

Method 2: Efficient Solution

A more efficient solution uses a frequency count to compare the characters of both strings. This approach avoids sorting and works in linear time.

- Initialize a count array of size 256 to represent all possible ASCII characters.
- Iterate through both strings simultaneously:
    - Increment the count for characters in the first string.
    - Decrement the count for characters in the second string.
- After iterating, check if all values in the count array are zero. If any value is non-zero, the strings are not anagrams.

Here's the code implementation of above approach:

```
import java.util.*;

class GfG {

    static final int CHAR = 256;

    static boolean areAnagram(String s1, String s2)
    {
        if(s1.length() != s2.length())
            return false;

        int count[] = new int[CHAR];
```

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

        for(int i=0; i<CHAR; i++)
        {
            if(count[i] != 0)
                return false;
        }

        return true;
    }

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

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

Output

```
true
```

Time Complexity: O(N)

Auxiliary Space: O(256)