```java
import java.io.*;
import java.util.Arrays;
import java.util.*;

class Solution{
    // to sort the strings in lexicographically non-decreasing order.
    public void sortLexo(String[] arr){
        // CASE_INSENSITIVE_ORDER: would do case insensitive sort
        Arrays.sort(arr, String.CASE_INSENSITIVE_ORDER);
        for(int i=0;i<arr.length;i++){
            System.out.print(arr[i]+" ");
        }
    }

    // to sort the strings in lexicographically non-increasing order.
    public void sortLexoreverse(String[] arr){
        // Collections.reverseOrder(): reverse the order of sorting
        Arrays.sort(arr, Collections.reverseOrder());
        for(int i=0;i<arr.length;i++){
            System.out.print(arr[i]+" ");
        }
    }

//Helper method To count the distinct character in a string
    public int countDistinct(String s)
{
    // Initialize map
    Map<Character, Integer> m = new HashMap<>();
    for(int i = 0; i < s.length(); i++)
    {
        // Count distinct characters : Create a hashmap for the given string
        // if(the map already contains the character key then increment its va
lue by 1)
        if (m.containsKey(s.charAt(i)))
        {
            m.put(s.charAt(i),
            m.get(s.charAt(i)) + 1);
        }
        // else( create a new key-
value pair inside hashmap and initialize value to 1)
        else
        {
            m.put(s.charAt(i), 1);
        }
    }
    // size of hashmap would give the number of distinct character in the stri
ng ( Keys are always unique)
    return m.size();
```

```
}
// to sort the strings in non-
decreasing order of the number of distinct characters present in them. If two
strings have the same
// number of distinct characters present in them, then the lexicographically s
maller string should appear first.
public void sort_by_number_of_distinct_characters(String[] arr){
    // Using a new Comparator: we can implement our own condition for sorting.
Arrays.sort(arr, new Comparator<String>()
    {
        public int compare(String a, String b)
        {
            if (countDistinct(a) ==
                countDistinct(b))
            {

                // Check if size of string 1
                // is same as string 2 then
                // return false because s1 should
                // not be placed before s2
                return (b.length() - a.length());
            }
            else
            {
                return (countDistinct(a) -
                        countDistinct(b));
            }
        }
    });

    // Printing the output
    for(int i = 0; i < arr.length; i++)
    {
        System.out.print(arr[i] + " ");
    }

}
// to sort the strings in non-decreasing order of their lengths.
// If two strings have the same length, then the lexicographically smaller str
ing should appear first.
public void sort_by_length(String[] s){
    int n = s.length;
    for (int i=1 ;i<n; i++)
    {
        String temp = s[i];

        // Insert s[j] at its correct position
        int j = i - 1;
```

```
        while (j >= 0 && temp.length() < s[j].length())
        {
            s[j+1] = s[j];
            j--;
        }
        s[j+1] = temp;
    }

    // Printing the output.
    for (int i=0; i<n; i++)
        System.out.print(s[i]+" ");
}

}
```