Q1 : Implement String Sort

Solution :

- The approach opted to tackle the problem is to use two **Arrays of Link Lists** each of size **27**.
- Say, length of the longest string is **L**. Hence, we will pad the rest of the strings with length less than **L** with **'$'** (smallest ascii value, let's say).
- Now iteratively, we will be considering one character at a time of each string from the end and storing the strings in the DSs alternately, according to the charachters.
- We will stop after the strings will be arranged on the basis of their first charachter.

Input Demo :

7 (No. of strings)
abb
Abd
sd
Sdffgr
erew
w
er

Output Demo :

abb
Abd
er
erew
sd
Sdffgr
w

Time Complexity :

L : length of longest string
N : No. of strings
Time Complexity : O(NL)

Space Complexity :

L : length of longest string
N : No. of strings
Space Complexity : O(NL)

Q2 : Given a string find the consecutive triples that have occurred most number of times

Solution :

- We are using **Trie** Data Structure to store all the triplets (consisting of 'a', 'g', 't', and 'c').
- During this process, every time we reach to depth three of any triplet, we increase the counter corresponding to that branch.
- Finally, we return the triplet(s) with the highest counter value.

Input Demo :

accaacctaccgggggaccggg (Given String)

Output Demo :

acc
ggg
(Both have frequency of 3)

Time Complexity:

N : Length of given string
Time Complexity : O(N)

Space Complexity :

As, there can be at most 64 different triplets. So, maximum space required is in the order of (3x64).
So, Space Complexity is **constant**.

Q3 : Given three strings, Determine their longest common substring

Solution :

- Firstly, we have extracted all the suffixes of all the three strings and sorted them.
- Then, we traverse over the suffix array and find all the ranges **i..j** where there is at least one suffix from each given string, and find out the **longest common prefix** of the first and last suffix in that range.
- The pefix having the maximum length is our solution.

Input Demo :

abababccabbba
aababcabbba
aaababcaabbba

Output Demo :

ababc
abbba
(Both are of length 5)

Time Complexity:

N1 : Length of 1st string
N2 : Length of 2nd string
N3 : Length of 3rd string
N : N1 + N2 + N3
M : MAX(N1, N2, N3)
Time Complexity : O(NlogN + MN)

Space Complexity :

O(MN) (To store all the strings)