CTIS264 COMPUTER ALGORITHMS
PYTHON LABSTUDY 3

A good example problem for showing algorithms with different orders of magnitude is the classic anagram detection problem for strings.

One string is an **anagram of another** if the second is simply a rearrangement of the first.

For example, **'heart'** and **'earth'** are anagrams. The strings **'python'** and **'typhon'** are anagrams as well.

For the sake of simplicity, we will assume that the two strings in question are of equal length and that they are made up of symbols from the set of 26 lowercase alphabetic characters. Our goal is to write Boolean functions that will take two strings and return whether they are anagrams.

Question 1:
Checking Off
Our first solution to the anagram problem will check to see that each character in the first string actually occurs in the second. If it is possible to "checkoff" each character, then the two strings must be anagrams. Checking off a character will be accomplished by replacing it with the special Python value **None**. However, since strings in Python are immutable, the first step in the process will be to convert the second string to a list. Each character from the first string can be checked against the characters in the list and if found, checked off by replacement.

Question 2:
Sort and Compare

Another solution to the anagram problem will make use of the fact that even though **s1** and **s2** are different, they are anagrams only if they consist of exactly the same characters. So, if we begin by sorting each string alphabetically, from a to z, we will end up with the same string if the original two strings are anagrams.
In Python we can use the built-in **sort** method on lists by simply converting each string to a list at the start.

Question 3: Count and Compare

Solve the anagram problem by taking the advantage of the fact that any two anagrams will have the same number of a's, the same number of b's, the same number of c's, and so on. Count the number of times each character occurs and keep them in a list of 26 counters, one for each possible character. In the end, if the two lists of counters are identical, the strings must be anagrams.

Question 4: Analyse efficiencies

Analyse the question 1, 2, and 3 algorithms with worst case and best case. Find the asymptotic efficiency classes for the worst cases of these algorithms and compare them. Write your analysis results in a text file.

Note: (Assume that Python sort function has efficiency class or $O(n \log n)$)

Send all of your python codes in a single .py file.
Send the answer of the 4th question as a text file.