**Solutions for Finding Anagrams**

**University:** Lund University

**Students:** Xhesina Hita, Davide Gotti

**Course:** Programming for Data Science

**Investigated problem:** selecting the most efficient algorithm for finding anagrams.

In this project, we focused on identifying anagrams, which are two strings that are composed of the same characters, although in a different order. To keep things simple, we assume that the strings consist only of lowercase alphabetical characters.

Our objective is to create a Boolean function that, given two strings, returns True if they are anagrams of each other. We find four distinct solutions for finding anagrams in both R and Python, with the goal of identifying the algorithm that minimizes execution time.

To measure the time efficiency of the algorithm from a theoretical point of view, we will employ the Big O notation, a mathematical framework used to quantify how an algorithm's resource usage and run time increase with larger input data (length of the word).

**Implemented solutions:**

* **Solution 1:** we perform an algorithm that checks if both stings have the same character frequencies, marking matched characters in the x2 string to ensure identical counts.
* **Solution 2:** in this case we sort the two strings and check if the two ordered strings are the same.
* **Solution 3:** we perform an algorithm that builds all the possible anagrams for the x1 string and then compares all of them with x2. If it finds a match, then they are anagrams.
* **Solution 4:** we perform an algorithm that counts the frequencies of the characters in x1 string and compares these frequencies with the x2 string. If they are the same, then they are anagrams.

**Conclusions:**

From the results we observe that in Python with small n (small words) it seems that the theory doesn’t match the results, but as n increases it seems that the results are congruent. As regards R, even with small values of n, the last algorithm (solution 4) which has complexity O(n) turns out to be the fastest, so the results are congruent to the theory.

If we compare the two software, we transform the microseconds outputs in R in such a way as to compare them with seconds (the Python outputs), we find that Python is faster.