LINEAR DATA STRUCTURES AND ALGORITHMS.

L02-03: Problems, Algorithms and Programs.

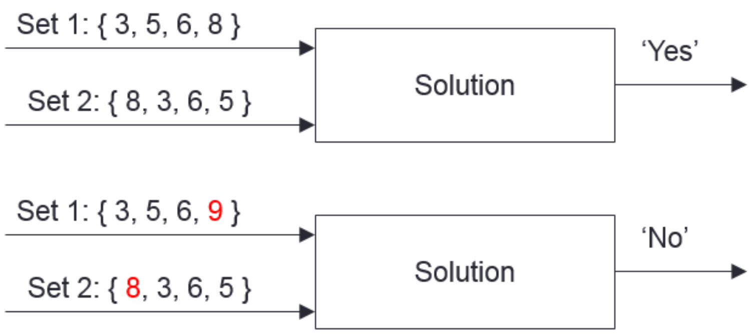
BACKGROUND.

We say two sets of numbers are *anagrams* if the number of set 1 can be constructed as a permutation of the numbers of set 2 and viceversa. Examples:

- Scenario A: Set 1 = { 3, 5, 6, 8} and Set 2 = { 8, 3, 6, 5} are anagrams, as each set can reorder its numbers to look exactly the same as the other set.
- Scenario B: Set 1 = { 3, 5, 6, 9} and Set 2 = { 8, 3, 6, 5} are not anagrams, as there is no way each set can be reordered to get the same numbers as the other set.

PROBLEM SPECIFICATION.

Provided 2 sets of n numbers { A_1, A_2, A_3, …, A_n } and { B_1, B_2, B_3, …, B_n }, come up with a solution to check whether the two sets are anagrams.



In our lecture 'Problems, Algorithms and Programs' we have seen three possible algorithms to be used as a formal specification of the solution:

1. Algorithm **anagrams1** (slides 39-41): Based on removing matching numbers between sets.
2. Algorithm **anagrams2** (slides 42-47): Based on sorting the numbers of both sets and compare them.
    - It uses the additional algorithms **bubble_sort** and **are_equal**.
3. Algorithm anagrams3 (slides 48-49): Based on counting the digit appearances of each set and compare them.
    - It uses the additional algorithms **appearances** and **are_equal**.

PRACTISE.

The folder contains the implementation of the algorithm **anagrams2** (and its additional algorithms **bubble_sort** and **are_equal)** in the following programming paradigms: