Lab 3: Radix Sort

You may work with another student on this exercise. If so, make sure you write both names in the header.

You have studied **Radix Sort**, an algorithm that has running time **O(n*k)**, where n is the number of elements and k is the number of digits. Write a program that sorts an array of integers. Capacity, number of elements, and digits are left to your choice.

Requirements:

- You may use **ONLY 2 arrays of integers** to manipulate the numbers.
- On the console, display the original arrays and all subsequent passes.

  Example:

  453 675 841
  841 453 675
  841 453 675
  453 675 841

- MAKE IT SIMPLE → Code the whole algorithm in the main function so that you can clearly see if it is an O(n*k) algorithm.
- Make it more efficient by using a **WHILE** loop that also checks if the array to which you are copying the numbers is filled up. For example, if the sequence of numbers is 32**4**, 12**3**, and 65**2**, and you are treating the right-most digit, then there is no need to check for 5-9, because the only digits are 2, 3, and 4, which will also indicate that the array where the numbers have been copied is full.
- Which functions should you create? You need to figure out how the application should be implemented.
- What about test cases? Create at least 4 test cases that differ in digits and number of elements.

NOTE: Write the code without worrying about efficiency. You will eventually see that, even if you have nested loops, you will still achieve an O(n*k) notation.

After writing, testing, and cleaning up your code, prove that your implementation has a running time O(n* k) by writing comments on the right side of the statements; indicate the running time for blocks of statements that lead to the final O(n * k).