CSC131: Computational Thinking
Homework Assignment # 6
Due: Beginning of class on Monday, November 27, 2015.

For this assignment, you will implement a sorting technique called **radix sort**. The solution will require the use of a data structure called **queue** where elements are always added to the end of the queue and are always removed from the front of the queue (similar to a line of people waiting to check out at a grocery store). A queue is a First-In-First-Out (FIFO for short) data structure because the first element to join the queue is the first one to leave it. A queue may be emulated using a list where the method append() can be used to add elements to the end of the list and the method pop(0) can be used to remove the element at the front. Since, the running time for pop(0) is linear, it is more efficient to use a **deque** class from the **collections** module, where both appending and popping take constant time. Notice that deque stands for double ended queue. The class **collections.deque** has the methods **pop()** and **append()** which operate on the right hand side end of the queue and the methods **popleft()** and **appendleft()** which operate on the left hand side end of the queue. So, you can either use the methods appendleft() and pop() or the methods append() and popleft() to emulate a queue. Samples of these approaches are given in the programs q1.py and q2.py on \\trace\Class\CSC-131\001_Download\DS Chapter3.

A sort is always based on some particular value, called the sort key. For example, a set of people might be sorted by their last names. Radix sort is a sorting technique that instead of comparing items on their sort keys, takes advantage of the structure of the data to be sorted. It uses a separate queue for each possible value of each digit or character of the sort key. The number of queues, or the number of possible values, is called the radix. For example, if we were sorting strings made up of lowercase alphabetic characters, the radix will be 26. We would use 26 separate queues, one for each possible character. If we were sorting decimal numbers, the radix would be ten, one for each digit 0 through 9.

Let's look at an example that uses radix sort to put ten three-digit numbers in order. To keep things manageable, we will restrict the digits of these numbers to 0 through 5, which means we will need only six queues.

Each three-digit number to be sorted has a 1s position (right digit), 10s position (middle digit), and a 100s position (left digit). The radix sort will make three passes through the values, one for each digit position. On the first pass, each number is put in the queue corresponding to its 1s digit. On the second pass, each number is put in the queue corresponding to its 10s digit. And finally, on the third pass, each number is put in the queue corresponding to its 100s digit.

Originally, the numbers are loaded into the queues from the input list. On the second pass, the numbers are taken from the queues in a particular order. They are taken from the digit 0 queue first, and then from the digit 1 queue, etc. For each queue, the numbers are processed in the order in which they come off the queue. This processing order is crucial to the operation of radix sort. Likewise, on the third pass, the numbers are again taken from the queues in the same way. When the numbers are pulled off of the queues after the third pass, they will be completely sorted.

The diagram bellow shows the processing of a radix sort for ten three-digit numbers. The number 442 is taken from the original list and put onto the queue corresponding to digit 2. Then, 503 is taken from the original list and put onto the queue corresponding to digit 3. Then, 312 is put onto the queue corresponding to digit 2 (following 442). This continues for all values, resulting in the set of queues for the 1s position.

Original list : 442 503 312 145 250 341 325 102 420 143

Digit	1s Position	10s Position	100s Position
0	420 250	503 102	
1	341	312	145 143 102
2	102 312 442	325 420	250
3	143 503		341 325 312
4		145 143 442 341	442 420
5	325 145	250	503

Assume, as we begin the second pass, that we have a fresh set of six empty digit queues. Actually, the queues can be used again if processed carefully (for example, by copying the elements back to the original list starting by the queue corresponding to digit 0 and ending by the queue corresponding to digit 5). To begin the second pass, the numbers are taken from the 0 digit queue first. The number 250 is put onto the queue corresponding to digit 5, and then 420 is put onto the queue corresponding to digit 2. Then, we can move to the next queue, taking 341 and putting it onto the queue for digit 4. This continues until all numbers have been taken off of the 1s position queues, resulting in the set of queues for the 10s position.

For the third pass, the process is repeated. First, 102 is put onto the queue for digit 1, then 503 is put onto the queue for digit 5, then 312 is put onto the queue for digit 3. This continues until we have the final set of digit queues for the 100s position. These numbers are now in sorted order if taken of each queue in turn.

Write a Python program that implements the above algorithm for radix sort. The input consists of three-digit positive integers (the digits are between 0 and 9). **The program must read the input from a text file called radix.in and must store the output in a text output file called radix.out**. The input file contains the numbers for each input list on a separate line (numbers are separated by single spaces). The output file contains the corresponding output lists on separate lines (numbers are separated by single spaces). Sample input and output files are as follows:

radix.in	**radix.out**
467 119 635 231 234 858	119 231 234 467 635 858
786 463 715 745 729 574 856 806 339 106 487	106 339 463 487 574 715 729 745 786 806 856
798 791 392 916 177 115 948 871 525	115 177 392 525 791 798 871 916 948

The input files that will be used to test your code can contain any number of lines. It will be called radix.in and will be placed in the same folder as your program.

Give your program the name **RadixSort.py**. Make sure to upload an electronic copy of the source file on your csc131 TRACE upload folder in a folder called HW\hw6. **Test your code after you upload it to TRACE**. Also make sure to turn in a stapled hardcopy of your code on the due date.