Lab 5 -- Big Numbers

The goal of this lab is to practice the implementation of a linked-sequential structure through the application of union data types.

Big Numbers

When we represent a number on a computer we must allocate a fixed number of bits to it. The number of bits dictates the range of numbers that can be stored. This data limitation applies to all data types, and prevents us from storing large numbers.

One way to represent a number is in terms of its digits, similar to how int is stored in terms of bits, or how a String represents a sequence of characters. In this assignment you will represent large integral numbers in a linked-list representation using a *union data type*. Instead of storing bits, we store individual digits (0-9) in a single node of such a list. We can store them in one of two orders: big-endian (most significant digit first) or little-endian (least significant digit first). For example, the number 32411 can be stored as 3 -> 2 -> 4 -> 1 -> 1 or 1 -> 1-> 4 -> 2 -> 3.

Given an integral number, we can shift its digits to the left or right. For example 32411 can be *left-shifted* to get the number 324110. Thus left-shifting by one position is equivalent to multiplying the number by 10 (similar to how left-shifting by one bit in a binary representation multiplies the number by 2). Similarly, 32411 can be *right-shifted* to get the number 3241, which is the integer-division by 10.

We can support basic addition of a single digit to a number. For example 324115 + 7 = 324122. Shifting and adding single digits can allow us to create arbitrarily large numbers, one digit at a time. For example 32411 can be created by:

1. Start with 0.
2. Left-shift by 1 position, and add 3
3. Left-shift by 1 position (to get 30) and add 2
4. Left-shift by 1 position (to get 320) and add 4
5. Left-shift by 1 position (to get 3240) and add 1
6. Left-shift by 1 position (to get 32410) and add 1

Numbers can also be added by using simple arithmetic: start from the right-most digits and add them. Record the sum and carry, and add the carry to the next pair of digits, and so on. Note that the numbers may be of different lengths.

What to do

**Package:** bignumber

Start by downloading [this BigNumber interface](https://northeastern.instructure.com/courses/63372/files/7938767?wrap=1)[download](https://northeastern.instructure.com/courses/63372/files/7938767/download?download_frd=1)that defines method signatures for the operations described above. Your task is to implement this interface in a class called  BigNumberImpl. This implementation represents non-negative numbers of arbitrary lengths. Beyond implementing the BigNumber interface, this implementation should have the following features/obey these constraints:

* You are not allowed to use any of Java's existing list implementations, interfaces, **arrays**, or otherwise any collection classes or maps to implement big numbers. You must create your own list implementation using a *recursive union*. The implementation may be customized for this application.
* You are not allowed to use the BigInteger or any similar existing classes from JDK in your implementation; however you may use BigInteger in your tests.
* This class should have a constructor with no parameters that creates the number 0.
* This class should have another constructor that takes a number as a string and represents it as a BigNumber. This constructor should throw an IllegalArgumentException if the string passed to it does not represent a valid number.
* This representation should not contain any unnecessary digits. That is, if representing a 5-digit number, there should be exactly 5 digits represented in this object.
* This class must provide a way to compare two numbers.
* This class should include a toString method that returns a string representation of this number, as simply the number itself.

We recommend that you select an operation and implement it end-to-end before starting the next operation:

1. Add an empty implementation
2. Write test cases
3. Complete the implementation until all of the tests cases pass