Project Description

In this project is to develop a program that implements arithmetic with large integers, of arbitrary size.

Objective: To implement the class Num that stores and performs arithmetic operations on arbitrarily large integers. Following data structures must be used for representing Num: Array of long integers, where the digits are in the chosen base. In particular, strings should not be used to represent the numbers. Each entry of the list stores exactly one long integer. The base is defined to be 10 in the starter code, but you may modify it. In the discussions below, we will use base = 10. For base = 10, the number 4028 is represented by the list: {8,2,0,4}.

Methods to implement:

- Num(String s): Constructor for Num class; takes a string s as parameter, with a number in decimal, and creates the Num object representing that number in the chosen base. **Note that, the string s is in base 10, even if the chosen base is not 10.** The string s can have arbitrary length.
- Num(long x): Constructor for Num class.
- String toString(): convert the Num class object into its equivalent string (in decimal).
  There should be no leading zeros in the string.
- Num add(Num a, Num b): sum of two numbers a+b stored as Num.
- Num subtract(Num a, Num b): a-b
- Num product(Num a, Num b): product of two numbers a*b.
- Num power(Num x, long n): given an Num x, and n, returns the Num corresponding to x^n (x to the power n). Assume that n is a nonnegative number.

Use divide-and-conquer to implement power using O(log n) calls to product and add.

- printList(): Print the base + ":" + elements of the list, separated by spaces.
- Num divide(Num a, Num b): Integer division a/b. Use divide-and-conquer or division
  algorithm. Return null if b=0.
- Num mod(Num a, Num b): remainder you get when a is divided by b (a%b). Assume that
  a is non-negative, and b > 0. Return null if b=0.
- Num squareRoot(Num a): return the square root of a (truncated). Use binary search.
  Assume that a is non-negative. Return null if b < 0.
- Num evaluatePostfix(String[] expr): evaluate the expression in postfix and return the
  resulting number.
- Num evaluateExp(String expr): parse/evaluate the given expression and return the
  resulting number. The operands and operators in the input string may or may not be separated by empty space. For example, both “(3+4) * 5” and “(3 + 4) *5” are valid inputs. Use StringBuilder to tokenize the input string, as strings are immutable in Java.