The goal of this assignment is to become comfortable using the basic Array and ArrayList data structures and to gain some exposure to sparse matrices. A sparse matrix is a data structure which does not require any memory to represent a 0.0 entry. We’ll run a simple algorithm on the two matrix types to determine relative execution times.

1. [10] Open Eclipse and create a new project (File -> New -> Project -> Java -> Java Project) named “Assignment3” (keep all other default settings). When the program starts, prompt the user for an integer matrix dimension (we’ll be using square matrices), a double-precision matrix density, and an integer iteration count (we’ll run the algorithm on many different matrices to determine an average execution time).

2. [80] For each of the above iterations (i.e., specified by the iteration count), do the following:

a. (20) Create two data structures to represent matrices. The first structure should be a 2-dimensional array of type double to act as a dense matrix. This 2-dimensional array will be square and its dimensions based on the above matrix dimension (i.e., dimension x dimension). The second structure should be based on ArrayLists, and should also be of type double. This structure will be used to represent all of the non-zero entries in the dense matrix. You can organize this second structure any way you’d like, provided that will contains no 0.0 entries.

b. (10) Populate the dense matrix by looping through the data structure and first determining whether each cell should contain a nonzero entry. To do this, generate a random number on the range [0.0, 1.0) using Math.random() and then compare that number against the desired density (from part 1). If Math.random() <= density, then populate the cell with the result of Math.random() + Double.MIN_VALUE (the epsilon prevents 0.0 values); if not, populate the cell with 0.0.

c. (10) Populate the sparse matrix such that it contains the same values at the dense matrix above expect that there are no entries in the data structure for the 0.0 entries. You may either accomplish this while populating the dense matrix or by first generating the dense matrix and then copying the values into the sparse matrix. Keep track of the actual number of non-zero cells.

d. (20) Write an algorithm to determine which row in the dense matrix contains the largest row sum (i.e., the sum of the values in the row). Also, record the execution time taken for this operation (see System.currentTimeMillis()). We may do something more interesting with matrices later in the term but this will suffice for the purposes of this assignment.
e. (20) Write an algorithm to determine which row in the sparse matrix contains the largest row sum. Also, record the execution time taken for this operation. You do not need to print these values per iteration (that will be a lot of print statements!), but you’ll want to verify that your row sums are correct and consistent between the matrix types.

3. [10] Output the total time taken for the dense and sparse operations along with the average number of entries in the sparse matrices. Observe the differences in execution times as you vary the matrix dimension and number of iterations. List and analyze your observations in the comments of your code.

4. [+20] (extra credit) In addition to finding the row sum of the matrices, also calculate the product of each matrix multiplied by itself (i.e., implement matrix multiplication on dense and sparse matrices). Hint: implementing matrix multiplication will require you to store the column associated with each sparse row entry (which is not required to calculate a row sum).

Zip the Assignment3 folder in your Eclipse workspace directory and upload the .zip file to Blackboard (see Assignment 3 assignment in the Course Documents area).