Fall ’14 CIS 212 Assignment 2 – 100/100 points possible – Due Monday, 10-13, 11:59 PM

The goal of this assignment is to become comfortable using the basic Array and ArrayList data structures and to gain some exposure to sparse arrays. A sparse array is a data structure that does not require any memory to represent an entry with the value 0, which is useful for low-density data. We’ll run a simple algorithm on the two array types as a quantitative experiment to compare the execution times of the two implementations.

For example, a dense array might look like:

[1, 10, 15, 0, 0, 2, 0, 7, 43, 0, 12, 0, 0]

The corresponding sparse array, containing all data necessary to expand back to the sparse representation, night look like:

[[0, 1], [1, 10], [2, 15], [5, 2], [7, 7], [8, 43], [10, 12]]

1. [10] Write a Java program that first prompts the user for an integer array length and a double-precision array density. You will use these values to create the arrays for the experiment (see parts 2 and 3 below). For example, if the user enters an array length of 100 and a density of 0.1, your program will create arrays of length 100 with on average of 10 random non-zero entries. Catch any exceptions due to the user entering unparsable input and prompt them to reenter the value(s).

2. [20] Write a function which takes an integer length and an array density of type double as arguments and returns a new array of type int representing a dense array. For each entry in the array, compare the density with a random number on the range [0.0, 1.0) (i.e., 0.0 up to, but not including, 1.0) to determine whether or not the entry should be 0 (hint: see the java.util.Random class). If the entry should be 0, simply populate the entry as such. If the entry should be non-zero, populate it with a random integer on the range [1, 1000000] (i.e., 1 through 1 million).

3. [20] Write a function which takes an integer length and an array density of type double as arguments and returns a new ArrayList representing a sparse array. As above, use the density to determine whether or not each entry should be 0. If the entry should be 0, simply do nothing (i.e., don’t add it to the ArrayList). If the entry should be non-zero, store its index and value (also on the range [1, 1000000]) in the ArrayList (hint: you may want to use an ArrayList of type int[] – note that this is not the most elegant solution but we have not yet covered objects in Java). In this way, the returned ArrayList will only require enough memory to hold the non-zero elements and could be expanded back to dense form if necessary.

4. [20] Write a function which takes an int array as an argument and prints the max value in the array, along with the index of that value in the array.

5. [20] Write a function which takes an ArrayList as an argument and prints the max value in the array, along with the original index of that value in the array (i.e., the index that you stored in part 3).

6. [10] Use the System.nanoTime() method to record the amount of time taken to run each of the above functions (i.e., steps 2-5 above). Print your timing results. Spend some time trying different combinations of inputs and write your findings in the comments of your code. Which implementation is faster for various cases? Your output should look something like:

```
Please array length:

1000000

Enter density:

0.01

createDenseArray() length: 1000000 time: 39323000

createSparseArray() length: 10143 time: 28992000

findMax (array): 999990 at: 677720

dense findMax() time: 4083000

findMax (list): 999908 at: 494143

sparse findMax() time: 1793000
```

Zip the Assignment2 folder in your Eclipse workspace directory and upload the .zip file to Blackboard (see Assignment 2 assignment in Assignments area).