Homework #1 – COMP 250, Fall 2015

Due date: Sept 23 2015, before 23:59.

Important format information:
You must submit through MyCourses two separate files (not zipped):

1. Your java source file (*not* the class file) with your solutions to Q1-4. Write your name and student ID at the top of the file.
2. A PDF document with your answer to Q5.

In this first assignment, you will implement in Java some of the algorithm we have seen in class to compute the size of the intersection between two unsorted list of students containing no duplicates. Start by downloading the code at http://www.cs.mcgill.ca/~blanchem/250/hw1/StudentList.java .

Question 1. (10 pts)

Write the code of the intersectionSizeNestedLoops method, implementing the algorithm seen in class (Lecture 2; Algorithm 1).

Question 2. (20 pts)

Write the code of the intersectionSizeBinarySearch method, implementing the algorithm seen in class (Lecture 2; Algorithm 2). Use the sort() method provided.

Question 3. (20 pts)

Write the code of the intersectionSizeSortAndParallelPointers method, implementing the algorithm seen in class (Lecture 2; Algorithm 3). Use the sort() method provided.

Question 4. (20 pts)

Write the code of the intersectionSizeMergeAndSort method, implementing the algorithm seen in class (Lecture 2; Algorithm 4). Use the sort() method provided.

Question 5. (30 pts)

So, which of the methods implemented in questions 1-4 is fastest? This is for you to discover! For each intersection algorithm implemented, you will measure the average running time needed to compute the intersection size of two lists of 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000, 512000, and 1024000 students each.
Notes:

- For these measurements, the best is to use lists of randomly generated student IDs, as done in the code already.
- For small list sizes, the execution will be too fast to be measured reliably, so you should measure the time to do, say, 1000 such intersections (each time on a

different pair of randomly generated lists – see code for an example), and then divide the total running time by 1000.

- You can use
  long currentTime = System.nanoTime();
  to get the time (in nanoseconds) before and after the execution of the 1000 repetitions.
- The time taken to generate the two random lists is not negligible. You should first measure the running time of a program that generates the list but doesn't call the relevant intersectionSize method, and then run the program again, this time generating the lists and calling the intersectionSize method. The difference between the two running times is the amount of time taken by your method.
- When running your experiments, make sure that your computer isn't running too many other jobs; this may affect your measurements.

Questions:

a) (10 points) Report a table with the average running time for a single list intersection for each method and list sizes.

b) (10 points) Based on your observations, try to determine how the running time of each method increases as a function of the size of the two lists. In general, what would be the average running time of each of the four methods for computing the intersection between two lists of *n* students? Give your answers as a function of *n* (e.g. $T(n) = 43\, n + 5 \log(n)$ milliseconds).

c) (10 points) Now, consider what happens if the two lists are of different sizes. For each of the four algorithms, report the running time for computing the intersection between a list L1 of 32000 students and a list L2 of 1024000 students, and the running time for computing the intersection between a list L1 of 1024000 students and a list of L2 of 32000 students. For each algorithm, explain (in 2-3 lines at most) why the running time changes or doesn't change.

Good luck!