CPE 349: Assignment 3 Counting Inversions

Consider the following problem that arises in analyzing rankings to find people with similar tastes. (The previous ranking of a group of people who have similar tastes can be used to make recommendations to other people in the group.) So given a ranking and a database of rankings from other people, how do you find people who have similar rankings?

We will simplify by assuming everyone is ranking the same n things – say books. A natural approach would be to label the books with integers and label them in the order of the first person's ranking. Then reorder these labels by the ordering of another customer's preference. Then count the number of books that are "out of order", in the second person's ordering relative to the first person's ordering(ranking).

This can be simplified to the problem, given n positive integers $a_1, \ldots, a_n$, we want a measure of how scrambled the numbers are. Think of these numbers as the rankings of the second person. We will do this by counting the number of inversions. **Let A be an array of comparable elements. Assume the elements of A are distinct. A pair A[i], A[j] is called an inversion if i<j and A[i]>A[j]**

Your assignment is to develop and implement an algorithm to count inversions whose worst case computational complexity measured by the number of comparisons is Θ(n log n). You may consult the textbook, the web, and other students for ideas. However all code most be your own.

0. Get an idea of how to solve the problem. **MergeSort is a good place to start**. Test it by hand on small problems. Finally write the pseudo code for your algorithm to count inversions.
1. **Implement your algorithm recursively in Java using Divide and Conquer**. The algorithm must be clear and as simple as possible. Its worst case complexity should be in O(n * log n)
2. Determine the recurrence relation that describes the number of comparisons your algorithm makes of entries of the array as a function of the length of the array.
3. Solve the recurrence relation by back substitution.

Deliverables:

1. Class **Inversions.java** must contain a method that returns a non- negative integer, **public static int invCounter (int [])**.
2. The method should return the number of inversions in the array. The numbers in the array will be a subset of {1, 2, …, 999}. The array will contain at least two elements.
3. The method invCounter and any helper methods **must** be designed to clearly indicate the divide step, the conquer step, and the combine step. **Your code must be easy to understand and contain good comments**. You may (but do not have to) have invCounter call other methods to carry out specific parts of your algorithm. Your comments should clearly explain how the algorithm works. Remember that comments are to help humans understand code. **Avoid too many or too few comments. Also avoid too much white space or too many small methods.**
4. Submit your Java code for the class **Inversions.java** to Canvas.
5. At the same time submit a pdf file, **CountAnalysis.pdf** , that contains
   a. The recurrence relation for the number of comparisons of array entries needed by the algorithm to count the inversions.
   b. Show the derivation (**using back substitution**) of **the worst case closed form** solution for the number of comparisons. You are **not** to derive the asymptotic class form (O, Θ, Ω) but **an actual function** that gives the number of comparisons in the worst case.

Input to the method:
```
6 4 3 1
```
Returned by the method:
```
6
```

Input to the method:
```
2 3 8 6 1
```
Returned by the method:
```
5
```