Inversion Counting  In this assignment, you will implement and test two algorithms for the inversion counting problem. Inversion counting is a commonly encountered problem that is very useful for comparing two rankings. For completeness, here is a brief definition of the problem. Given an input array of size $n$, which contains all the integers between 1 and $n$ in a random order. Count the number of inversions in the array. An inversion happens when a pair of elements in the array are ordered in reverse. For example, the array $[4, 2, 5, 3, 1, 8]$ has a total of 7 inversions $(4, 2), (4, 3), (4, 1), (2, 1), (5, 3), (5, 1)$, and $(3, 1)$.

Please implement three different algorithms for this problem.

1. Brute-force. This algorithm works simply by checking all possible pairs of elements to count the total inversions.

2. Naive Divide and Conquer. This algorithm will count the inversion by dividing the given array $A$ into two equal-sized subarrays $A_l$ and $A_r$. To come up with the total inversions, recursively count the inversions in each subarray, and then count how many inversions happened between $A_l$ and $A_r$ by considering all cross-pairs (one element in $A_l$, one in $A_r$). This algorithm will also run in quadratic time.

3. Merge and Count. This algorithm builds on the intuition that when counting the inversions between the two subarrays, if both subarray are already sorted, then we don’t have to compare all cross-pairs. For example, consider two sorted subarrays of the above example $A_l = [2, 4, 5]$ and $A_r = [1, 3, 8]$. Since the first element of $A_r$ (1) is less than 2, which is the first and smallest element of $A_l$, we can tell that it must be smaller than all the elements of $A_l$. Thus, without comparing 1 with every elements of $A_l$, we already know that it creates 3 inversions, where 3 is the size of $A_l$. This suggests an algorithm that is similar in spirit with Merge Sort, but when we merge two sorted arrays, we need the added functionality of counting the inversions between the two subarrays. This algorithm should run in $O(n \log n)$ time.
Empirically testing the correctness of your algorithm. You can use the sample inputs/outputs (http://classes.engr.oregonstate.edu/eecs/winter2013/cs325/hws/verify.txt, each row is an input, the last number is the right output) to test the correctness of your algorithm. I will also provide a test file that contains a limited number of inputs, your report should list your answers to these inputs to help TA to get a quick assessment of the correctness of your algorithm.

Empirical analysis of run time. Please run your algorithms on input arrays of size 1k, 2k, 3k, 4k, 5k, and 10k, 20k, 30k, 40k, 50k respectively. To do this, you should generate your own random inputs use a random number generator (random permutation of numbers 1, . . . , n where n is the input size) provided by your programming language. For each size, you should generate 10 random inputs and run each algorithm on it and measure the running time of each algorithm. In your report, please plot the running time as a function of the input size. Include an additional plot of the running time in a log-log plot. See http://en.wikipedia.org/wiki/Log-log_graph for an explanation. Note that if the slope of a line in a log-log plot is m, then the line is of the form $O(x^m)$ on a linear plot.

Report. In your report, you should include the following:

- **Pseudo-code.** Please provide the Pseudo-code for each of the three algorithms.

- **Correctness proof.** Please construct a proof for each of your divide and conquer algorithms to show that it is correct.

- **Asymptotic Analysis of run time.** Please provide the runtime analysis for the three algorithms. In particular, please provide the recursive relation of the runtime for algorithm 2 and 3 and solve them.

- **Testing** Please test your algorithm on the testing inputs provided on the class website, and report the outputs for each input.

- **Extrapolation and interpretation** Use the data from the experimental analysis to answer the following questions. What is the largest input size your algorithm could solve in one hour? Determine the slope of the lines in your log-log plot. Discuss any discrepancy between the experimental runtime and the asymptotic runtime.