CS325 programming assignment 1

Printed report Due on Feb 4th in class
Electronic submission of Code due at 3pm the same day

January 23, 2013

Hand in Instructions Code should be submitted through the TEACH website. Report should be printed and submitted in class.

Counting inversions In this assignment, you will implement and test two algorithms for the inversion counting problem discussed in class. 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 $[1, 4, 2, 5, 3]$ has a total of 3 inversions $(4, 2)$, $(4, 3)$, and $(5, 3)$.

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 subarrrays 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 each pair (runs in quadratic time).

3. Merge and Count. This algorithm builds on merge sort. It also divides the given array A into two equal-sized subarrrays A_l and A_r, and each subarray is then recursively sorted and counted for inversion. When merging the two sorted sub-arrays, it counts the number of inversions between A_l and A_r in linear time. See the reading material on inversion counting for full detail of the algorithm.

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 have also provided a test file that contains 10 inputs, your report should list your answers to these 10 inputs so that the TA can verify 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 $1, \ldots, n$ 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:

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