Assignment 3: Experimental Verification of Asymptotic Running Time

Maximum Points: 100
Due date: November 22, 2010 at 9:59 p.m.
Two persons may work together and submit one solution.
This assignment is comprised of two parts.

1 Part 1: Running Time Experiments

In this part you will experiment with insertion sort, bubble sort, and quick sort algorithms on different types of input. You will be provided with the C++ code implementing the sorting algorithms and a C++ timer class to record the running time of sorting algorithms. You are required to write an appropriate main program and to undertake necessary modification of the supplied code and perform the following experiments.

1. **[Bubble Sort and Insertion Sort.]** Modify the code to record the number of comparisons and the number of assignments in each of the algorithms. Now run Bubble sort and Insertion sort algorithms on the following input data and record in a table, in each case, the time needed and the number of comparison and the number of assignments executed by the sorting algorithms.

   (a) Randomly generated input of size 10000, 20000, 50000, 100000 elements.

   (b) Almost sorted input (90% and 80% of the elements are already in their correct location in the sorted order).

   (c) Reversely sorted input.

2. **[Quick Sort.]** Modify the code to record

   (a) the number of comparisons and the number of assignments

   (b) the running time

   in Quick Sort where the pivot is (a) the leftmost element, (b) the median of the first, middle, and the last element (c) randomly selected element on the following input data.

   (a) Randomly generated input of size 100000, 200000, 500000, 1000000 elements.

   (b) Almost sorted input (90% and 80% of the elements are already in their correct location in the sorted order).

   (c) Reversely sorted input.

Answer the following questions.

1. [35] For each sorting algorithm determine the worst-case input (input that requires the longest time or executes the largest number of fundamental operations) and explain why this particular input constitutes the worst-case input.

2. [35] This question concerns comparing the actual running time for a sorting algorithm with its Big-oh running time estimate. Although this comparison can be done rigorously using a mathematical technique called *least squares* (which is beyond the scope of this course) we will take a much simpler approach here. Suppose we have an algorithm with asymptotic complexity of $O(n^2)$. We record the actual CPU time of the algorithms for k different input sizes say $n_1, n_2, \ldots, n_k$. To even out some computer specific anomalies in recording running time, for each input size we run the algorithm 20 times and record the average of these 20 CPU times. Next, we divide the average running time with the corresponding input size. For example, if the CPU running time of the algorithm averaged over 20 readings is l seconds for input size n_j then we record the value $c_j = \frac{l}{n_j^2}$. Then we will have k values $c_1, c_2, \ldots, c_k$ for the input sizes $n_1, n_2, \ldots, n_k$. Plotting these values along y-axis against the input sizes along x-axis and connecting the points on the graph paper we would get a curve. If our Big-Oh analysis was "tight" (i.e. we haven't overestimated the running time), then this curve would roughly be a straight line parallel to the x-axis.

   Use $O(n^2)$ for bubble and insertion sort, and $O(n \log_2 n$ for quick sort and find these c_is. Use only randomly generated input data for this question. In your answer, mention the computer system (the CPU name and its processor speed, the operating system, the compiler etc.) that you have used. Explain the shape of the curve that you got for each of bubble, insertion, and quick sort algorithms.

Submit the following.

1. The main program and the sorting routines (modified).

2. The running time graphs and tables together with your answers to the questions.

2 Implementation of Recursive Functions

1. (a) [20] Reimplement the Insertion Sort as a recursive function. Now run both version of insertion sort function on different input data as in Part I. Compare the running time and explain any differences in running time.

   (b) [10] Write a recursive function that finds the smallest element in a vector of integers.

   Submit the code for the functions and a main program that tests the functions for different input sizes.