EE2331 homework 3

Out: Oct. 24
Due: 11:59PM Nov. 2nd (Thursday)

This homework contains two parts. The first part includes one written problem. The second part involves programming. You should submit two files. The first file contains the solutions to Problems 1 and the running time report (Problem 3). The second file is your codes for us to test/run. This homework's total credit is 50.

There is one extra-credit problem. If you choose to do it, your maximum credit can be 60/50. Submit your answer to Problem 4 in the first file.

Problem 1. (10 pts) Following the format in the file note4/supplementary-note.pdf (Canvas), show all the mergesort functions (and their input parameters) in their calling order for input array <9 0 1 2 8 10>. Note that you must **use indentation** to show which mergesort functions are called by the same parent function (see the example in the supplementary file).

Problem 2. (10 pts) This is a programming problem. Modify the given mergesort algorithm in class to output a reversed sorted array. For example, input array is <1 200 234 21 35 20 99 30>. The output array is <234 200 99 35 30 21 20 1>.
Reversely sorted array means an array ordered in decreasing order or non-increasing order (if there are duplicated items). Examples: <100 98 32 1 -1>, <100 100 34 34 0 -1>.
Note: you are not allowed to just reverse a sorted array. Instead, you should directly modify the code to ensure that every sorted sub-array is reversely sorted.
A framework of generating some random numbers is given (hw3-Q2.cpp). It reads a number N and generates N random numbers. It uses some functions from C. You can modify it into C++ if you feel more comfortable with C++ (although it is not necessary).
Grading: correct implementation will be evaluated based on your code and the outputs on two tests. Every test is 5 pts.

Problem 3. (30 pts) This is a programming problem. Implement quicksort, mergesort, and insertion-sort in C/C++ and compare their running time for inputs with different sizes. The specific requirements are:

1. Implement all sorting algorithms in one cpp file. Name it as comparison-yourname.cpp. The framework of your code should follow the given code: compare-sort.cpp
2. You can re-use the insertion/merge/quick sort programs you implemented earlier.
3. You need to figure out **which timing function you need to use in order to demonstrate the running time differences**. Resolution at second level is not good enough. Please use the timing function to get microseconds for all sorting algorithms.
4. For all sorting algorithms, you must use STL vector (like an array, taught in class) to save the random numbers and the sorted numbers. No pointers/array should be used.

5. After testing the correctness of your sorting algorithms, compare and **report their running time** on inputs with following sizes:
   20, 100, 200, 400, 800, 1600, 3200, 6400

The running time comparison should be included in the same file as problem 1.

Correct implementation of all sorting algorithms using STL vector: 20 points, running time report: 10 points.

Extra-credit problem. This will be counted towards your total homework credit. For example, suppose you got 45 out of the first three problems. But you got 10 in problem 4. Your total will be 55/50 for this homework. The extra 5 can be used to cover deducted points in other homework/tutorials.

Problem 4. (10 pts) Watch the **Quick-sort with Hungarian (Küküllőmenti legényes) folk dance**
 (https://www.youtube.com/watch?v=ywWBy6J5gz8). **First**, use English to describe their sorting algorithm and highlight the difference between it and our in-class quicksort algorithm. **Then**, write the pseudocode for the sorting algorithm presented in this dance. Use the similar notation as the in-class pseudocode. Make sure that your pseudocode contains all the key operations. Add comments for us to understand your pseudocode. You can only get partial credit your logic is not clearly expressed by your pseudocode.