CSE 331 - Project 1
Quick Sort vs. Insertion Sort

TA: Sorrachai Yingchareonthawornchai
yingchar@msu.edu
Due Date: 11:59 pm Sep 12, 2014

1 Project Description

In this project, you will complete implementation of the quicksort and insertion sort algorithms. You are given skeleton code (i.e., main.cpp) that performs reading inputs and printing outputs. Your job is to complete the implementation of the followings:

1. InsertionSort (in file insertionSort.h). Return number of comparisons the algorithm made for given input.
2. QuickSort (in file quickSort.h). Return number of comparisons the algorithm made for given input.
3. VerifySort (in file verifySort.h). Return true if the input vector is sorted. Return false otherwise.

You should count number of comparisons made by elements in the array. For quick sort, there is no need to count one-by-one. You can add $m - 1$ for each recursive call on the subarray where m is size of subarry. This is because partition makes exactly $m - 1$ comparisons. Different pivot scheme results in different number of comparisons. You **must** partition using 1st element as a pivot. That is, you always pick the pivot element to be the key from the **first element** of the subarray.
Make no other changes to the files other than completing the assigned methods. You must provide your own code; you may not copy another's code. You may not make any other explicit function calls from the assigned methods, except to functions you define yourself. You may not use any of the tools provided in the STL other than those in STL vector (i.e. you may not call STL sort, or any other STL algorithm, but you may use vector::size() and vector::resize()). Remember to read the project guidelines before you start coding.

2 Input Test Cases

There are two sets of input: randomly generated integers(i.e., r20k, r40k, ..., r100k) and almost sorted integers (i.e., as20k, as40k, ... , as100k). Input is a file containing integers line-by-line. Your task is to execute your implemented sort functions and report results (see Written Questions section) according to the following format:

```
input_file num_comp_qs time_qs num_comp_is time_is
Example:
r20k 318188 0 100436650 2
r40k 702190 0 398373506 6
..
```

Where num_comp_qs is number of comparisons made in quicksort algorithm. time_qs is actual running time for quicksort algorithm in second. num_comp_is is number of comparisons made in insertion sort algorithm. time_is is actual running time for insertion sort algorithm in second.
Note: use CSE black server as a running machine.
Larger test cases will be executed after due date for grading.
Your implementation of sorting may vary in number of comparisons by a small constant factor. As long as it has same order of magnitude (that is, order of $n\log n$, n, or n^2) it is fine.

3 Project Deliverables

The following files must be submitted via Handin no later than 11:59 pm Sep 12, 2014.

1. QuickSort.h - contains your implementation of Quicksort algorithm
2. InsertionSort.h - contains your implementation of Insertion Sort algorithm
3. VerifySort.h - contains your implementation of Verify Sort algorithm
4. project1.pdf - file containing your answers to written questions

4 Written Questions

1. Report your execution results from given input test cases for both quicksort and insertion sort algorithms.
2. According to your output in test case, which algorithm performs faster (1) for randomly generated data, (2) for almost sorted data. Brieftly explain why for (1) and (2). What conclusion you can suggest from this experiment?

3. Answer question (1) and (2), but now use random pivot scheme for quick-sort algorithm. (no need to submit code for this)

4. What is the running time of Quick Sort when all elements of array have the same value?