CptS 223 Micro Assignment #5 - Instrumented Sorting

For this micro assignment, you must implement the two following sorting algorithms:

Insertion Sort
Mergesort

Extra Credit: Quicksort

The sorts you implement each reside in their own files: instrumentedInsertionSort.h, instrumentedMergeSort.h. and instrumentedQuickSort.h. You'll note that the interface provided takes both the vector to be sorted and a pointer to a stats structure. Please look at the bubblesort implementation carefully to see how the stats structure is used. The reason this is an instrumented sort assignment is the primary focus centers around counting how many times the different sorting algorithms compare values in the vectors and how many moves/swaps happen during the sort.

In the BS algorithm, I put a ++stats.compares before the actual comparison in the for loop. This will count the number of times we do comparisons. If you postincrement in that code you'll get bugs in your algorithms, mostly due to off by one errors and possible div/0 instances. Preincrement is the right way to go here.

Then, since BS does swaps, I count the number of swaps. You could argue that I should count each stage of the swap (a->tmp, b->a, tmp->b), but a single count for the whole swap is reasonable. The other algorithms should all be counting moves, but BS still comes out way slow no matter which way you do it.

I have tested this code on the EECS SIG servers. I used the command line:

```
g++ -g -Wall -std=c++11 main.cpp
```

You can run this with 'make', 'make test', and 'make run' as per usual

There's an additional test 'make bigtest' that'll do arrays of 30,000 elements. It takes about a minute on my desktop machine and 47 seconds on SIG4 to run and gives some nice big numbers to mull over. 30,000 elements isn't really that many in a data set, but have a look at the quantity of compares BS does vs. the other sorts! I've put a screenshot of the results from my implementation at the end of this assignment description so you can see the rough numbers you'll get. Depending upon your implementations, these numbers might vary, so don't assume you'll get exactly the same values.

It currently builds and runs properly, but Insertion, Merge, and Quick sort all fail their tests because they're stubbed in code. I would encourage you to see how I'm generating the testVectors. They're classes that inherit from vector. The shuffled ones allow you to set the random seed so you can get the same results each time, or get truly shuffled sets if you don't set the seed when you instantiate the object. I also use a templated *function* in main for the testing. You can use templates to define the variable types within a function as well as a class. I use it to pick the type of sorted data I want to run the tests on. Lastly, each sorting algorithm has code to time how long the actual sorting takes to complete. Your sort should go between these two clock() calls, be it a function or just the full sorting code for the given algorithm.

Grading

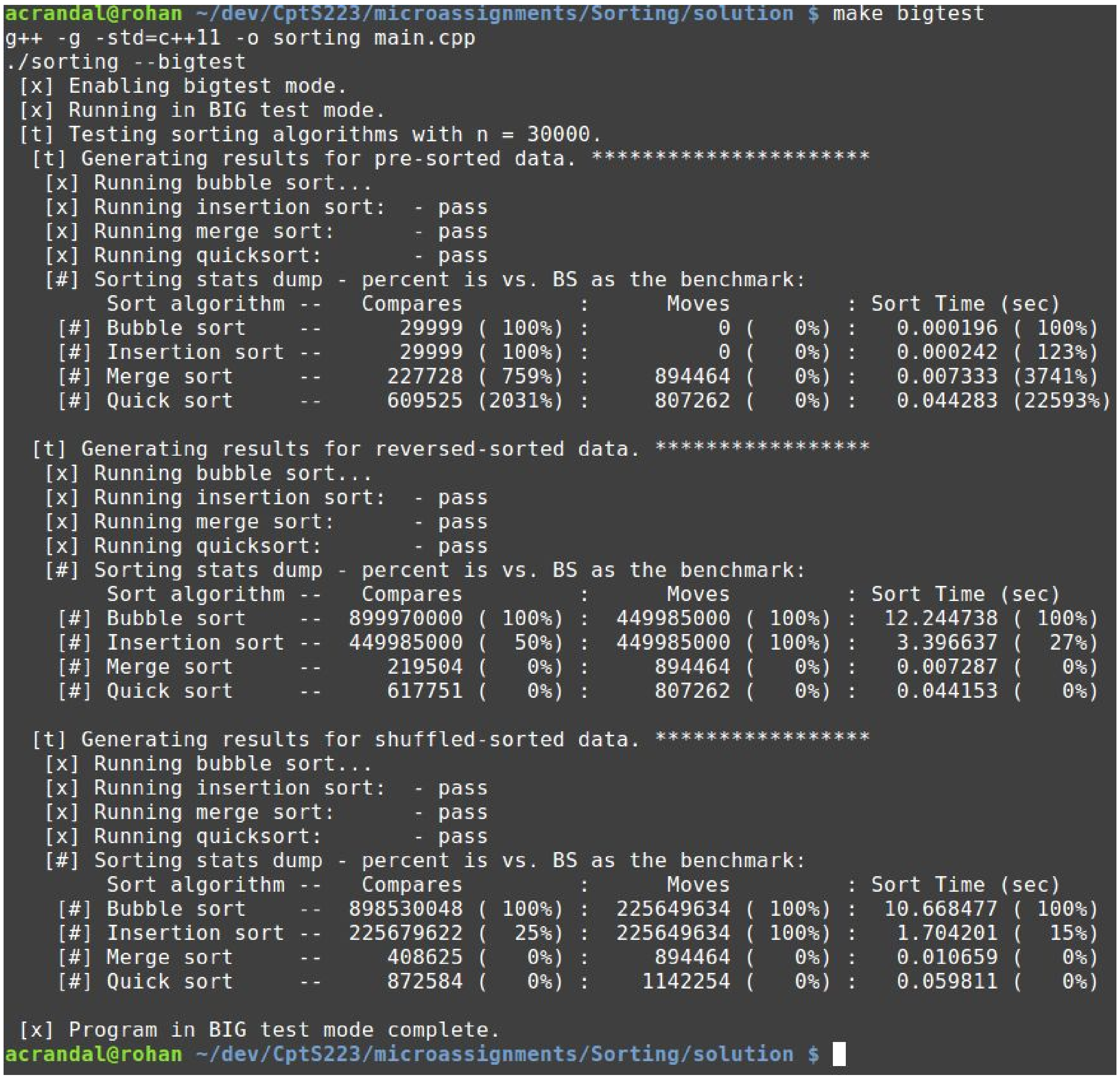
Your submission will be graded based on the following:

1. [10] Your solution builds, does not cause any runtime issues, and passes all test cases
2. [4] Everything works for (1) plus you've also done quicksort

Final Submission

As per normal, the assignment is to be checked into the project branch in your git repo. The only thing turned into Blackboard is a file with your name, WSU ID, EECS username, and git commit hash from your final version of the code.

Screenshot from my implementation of the algorithms. I wanted you to see how it should generally look when you're filling in the stats during your sorting algorithms. Note that bubblesort actually beats insertion sort in my implementation for pre-sorted data. Also see how mergesort and quicksort have different profiles in how they compare and move the data, but that my quicksort algorithm is much less efficient than my mergesort algorithm.



```
acrandal@rohan ~/dev/CptS223/microassignments/Sorting/solution $ make bigtest
g++ -g -std=c++11 -o sorting main.cpp
./sorting --bigtest
 [x] Enabling bigtest mode.
 [x] Running in BIG test mode.
 [t] Testing sorting algorithms with n = 30000.
  [t] Generating results for pre-sorted data. **********************
   [x] Running bubble sort...
   [x] Running insertion sort:  - pass
   [x] Running merge sort:      - pass
   [x] Running quicksort:       - pass
   [#] Sorting stats dump - percent is vs. BS as the benchmark:
       Sort algorithm --   Compares        :      Moves         : Sort Time (sec)
    [#] Bubble sort    --      29999 ( 100%) :          0 (   0%) :   0.000196 ( 100%)
    [#] Insertion sort --      29999 ( 100%) :          0 (   0%) :   0.000242 ( 123%)
    [#] Merge sort     --     227728 ( 759%) :     894464 (   0%) :   0.007333 (3741%)
    [#] Quick sort     --     609525 (2031%) :     807262 (   0%) :   0.044283 (22593%)

  [t] Generating results for reversed-sorted data. *****************
   [x] Running bubble sort...
   [x] Running insertion sort:  - pass
   [x] Running merge sort:      - pass
   [x] Running quicksort:       - pass
   [#] Sorting stats dump - percent is vs. BS as the benchmark:
       Sort algorithm --   Compares        :      Moves         : Sort Time (sec)
    [#] Bubble sort    --  899970000 ( 100%) :  449985000 ( 100%) :  12.244738 ( 100%)
    [#] Insertion sort --  449985000 (  50%) :  449985000 ( 100%) :   3.396637 (  27%)
    [#] Merge sort     --     219504 (   0%) :     894464 (   0%) :   0.007287 (   0%)
    [#] Quick sort     --     617751 (   0%) :     807262 (   0%) :   0.044153 (   0%)

  [t] Generating results for shuffled-sorted data. *****************
   [x] Running bubble sort...
   [x] Running insertion sort:  - pass
   [x] Running merge sort:      - pass
   [x] Running quicksort:       - pass
   [#] Sorting stats dump - percent is vs. BS as the benchmark:
       Sort algorithm --   Compares        :      Moves         : Sort Time (sec)
    [#] Bubble sort    --  898530048 ( 100%) :  225649634 ( 100%) :  10.668477 ( 100%)
    [#] Insertion sort --  225679622 (  25%) :  225649634 ( 100%) :   1.704201 (  15%)
    [#] Merge sort     --     408625 (   0%) :     894464 (   0%) :   0.010659 (   0%)
    [#] Quick sort     --     872584 (   0%) :    1142254 (   0%) :   0.059811 (   0%)

 [x] Program in BIG test mode complete.
acrandal@rohan ~/dev/CptS223/microassignments/Sorting/solution $
```