Vincent Lee

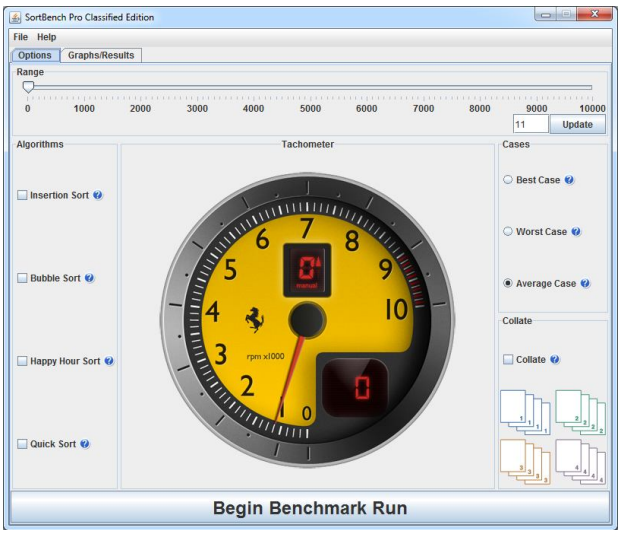
leevince@uga.edu

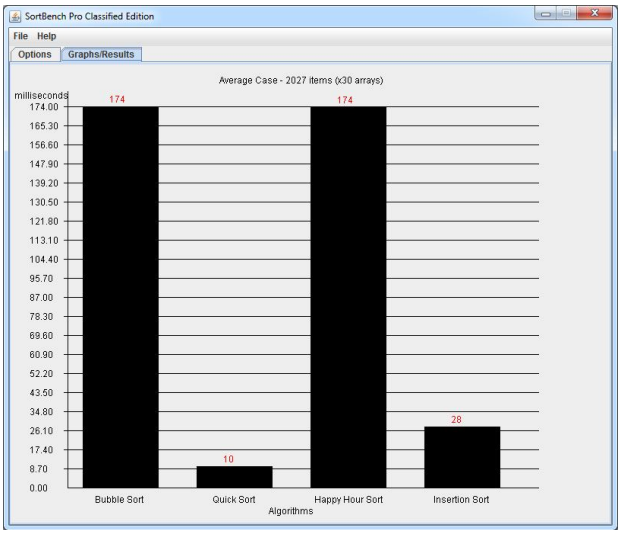
SortBench Pro Classified Edition v114.2

Experimental Design:

For SortBench Pro the aim of the software was for it to resemble and act like PC benchmarking software. For the design we utilized a border layout consisting of cardinal direction locations and center. Also there is a menu bar with file-close options, and help-about-getting starting options. For the main panel we have a tabbed layout in which the options resize and a graph panel for results and conclusions. Inside of the option panel, we have segregated panel borders for each group. For the range slider, the north panel was taken up due to the fact it is a slider, which makes sense to give it the most lateral real-estate possible. There is also a text field if you would like use a precise number for the range. For the algorithm choices, we went with check boxes, which are mutually inclusive, where one can be selected all the way up to all four algorithms. In the center we have a tachometer which is linked to how many check boxes are selected at a given time. In the tachometer, the engine is in 0 gear, neutral, and has a speed of 0 km/h. It also is idling at a little above 1k. This is to represent the program is running. When you select algorithms, the engine power increases, and the gear and speed change to represent this. Top speed is 315 km/h in 6th gear. On the right side we have our cases and the collate option. The cases represent different types of data. Best case would be already sorted data. Worst case would be reverse sorted data, and average case is randomly generated data. For the collate option, it is set to off by default. Selecting it will mean the algorithm will run each algorithm once before proceeding to the second array. This program also has help icons. These are clickable and give more information. And finally we have the benchmark run button located at the bottom. This is natural to have this as the last object, giving no more options to select.

Underneath the hood the program is very simple. There are two int arrays which are “30” arrays long. In each of these the length of the 30 arrays is set by the slider. The arrays are filled by which case is selected when benchmark is run. The benchmark button will run only through the algorithms selected at the time of running, and only report back those results. The time is logged through a 4 long variables, one for each of the 4 algorithms. The program also decides how to execute based on if collate is selected. If it is selected, the decision tree will go through that option. Once the benchmark has finished execution, the graph is created using bars, lines and strings. I utilized a numbers between 10 and 10001, because 10000 is a good number to see distinction between times, and on all of machines would not cause the recursion to generate a stack overflow exception.





Hypothesis:

Cocktail sort/Happy Hour
Worst case performance $O(n^2)$
Best case performance $O(n)$
Average case performance $O(n^2)$

Insertion sort
Worst case performance $O(n^2)$
Best case performance $O(n)$
Average case performance $O(n^2)$

Bubble sort
Worst case performance $O(n^2)$
Best case performance $O(n)$
Average case performance $O(n^2)$

Quicksort
Worst case performance $O(n^2)$
Best case performance $O(n \log n)$

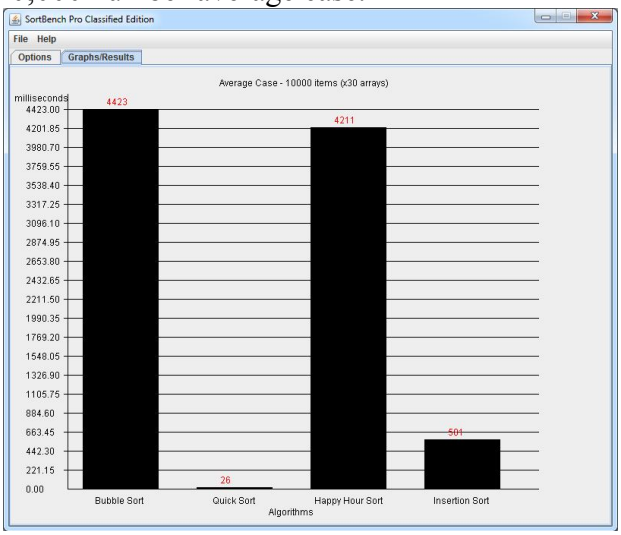
From the chart above, information gathered from wikipedia, the worst case performance for all four algorithms should be the same. Also for the happy hour sort, bubble sort, and insertion sort, the best case and average case performance should be the same. This is all theoretical and in real life I believe this will be different. From looking at the statistics on those three arrays I cannot conclude a winner. What I can hypothesize about is that quick sort's best case and average case performance should be a clear winner taking only a fraction of time since its time is O(n log n). For all algorithms besides quick sort, average case scenario time should exponentially increase with the amount of numbers being sorted. Quicksort seems most efficient through the average case performance through the numbers.

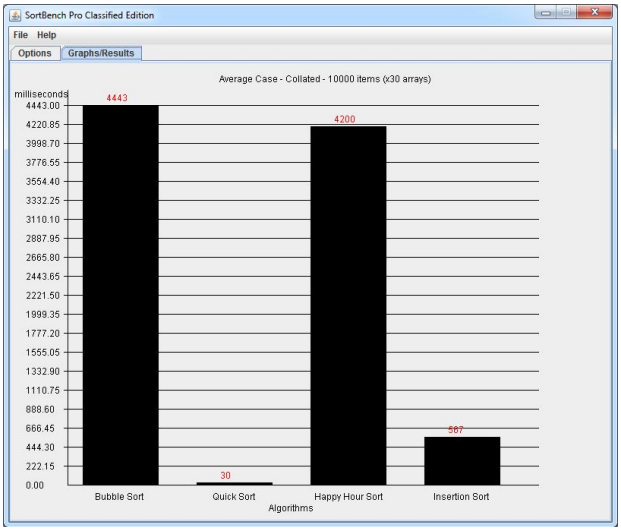
I believe that the performance will be different on different platforms, i.e. Nike the Mac's in 307, and on a few of my personal machines. Nike's performance, I believe, will be the most unpredictable. For one Nike is a multi-user system, and as Plaue also notes, handles all of the .cs.uga.edu emails. In a perfect world, we could run our sort algorithm, on Nike through a physical terminal connected to Nike, with all network connections off. That is not possible frankly. Some worst case scenarios would be for example, if a 1301 student was running an infinite loop on Nike, taking up all 4 of the Intel Xeon 2.4Ghz 10 core 20 thread processors. When running the sorting algorithm during this event would yield longer times than normal. Another problem that could occur is if a Backdoor Exploit was utilized on Nike to create a black hole email server, which would spam email accounts, and use up precious CPU time. Also since JAVA is ran on a virtual machine, students running a similar project in C++ are closer to the hardware in terms of execution, could use up more CPU time than ran on JAVA's virtual machine. The Mac's in the Boyd 307 should yield similar results if they are all running the latest version of OSX. The only problem is the Mac's do not have JAVA installed. On my personal computer there would be some variation, because my computer is not running server grade chips, and I have a lot of background process running on my Windows 7 Box. Performance, I would conclude would be quite close, as I am the only user on the machine at a particular time.

Results:

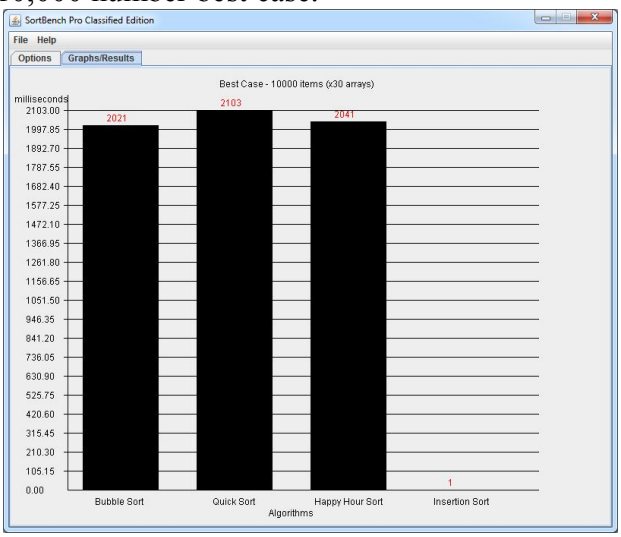
The program has 4 algorithms, 3 cases for the algorithms, and 2 options for the collate feature. This would conclude if we have all the algorithms on at all times, there are 6 different possibilities.

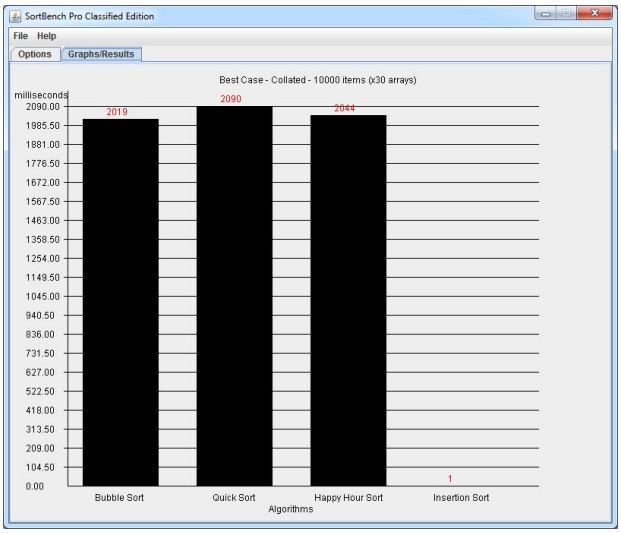
10,000 number average case.



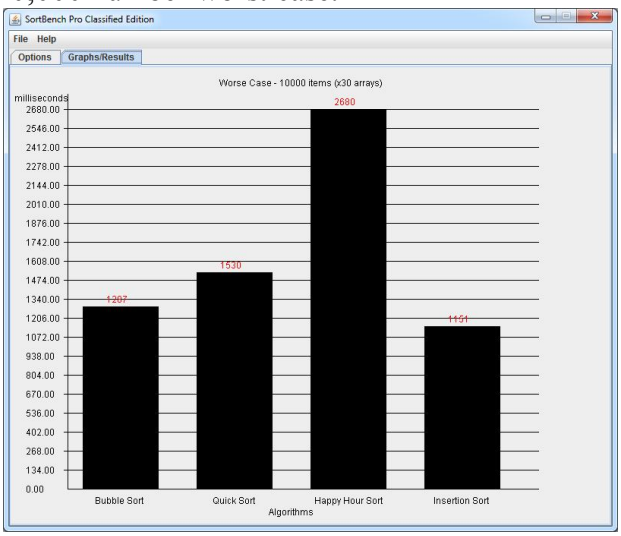


10,000 number best case.





10,000 number worst case.



Discussion:
Having all 6 possibilities we can see some general data trends. First, the collate option has almost no effect on the graph time. Sometimes the collated option brings up results that are slightly higher than the control, and sometimes they are lower. I would say it is nice to implement this method, but useless in real life.

Going back to the original hypothesis, there was not a correct statement made in that early phase judging by their theoretical times. All of the 4 sorting algorithms had the same time complexity for average, best, and worst case, except for quick sort**.** In theory all three of these should be exactly the same length in terms of time. This is frankly not the case and we can clearly see a difference.

For example, if you were sorting data sets contained random numbers, utilizing a quick would give you the fasted performance by far of any of the other 3 contenders. If you wanted to check a data set that was already in order, utilizing insertion sort would take the least of time. For worst case numbers, there is not really a clear winner but choosing anything but happy hour sort, would be best advised.

Clearly the cost/benefit of the algorithms would be the amount of cpu time it takes to compute the data sets into order, which relates to actually power costs of running up the cpu for processing. In theory, the quicker an algorithm, the less cost associated with the algorithm, and lowering the cost required, thus increasing the benefit to cost ratio. There is no clear winner, but based on my program, one can have a general idea of what type of data he has, and using my graph can select the most efficient algorithm to complete the task at hand.

All testing was done on a personal machine, because I found the results on nike too unpredictable to cite as quantifiable data. I ran into problems where non-collated, and collated runs were completely different because there were many people on the CS server, and decided to do all of my testing on a personal machine. This machine was restarted and eclipse was running to start the benchmark software. Also all network connections were disabled during the test.

Computer Specs
Quad Core 3.0Ghz 65nm
8GB 1000mhz DDR2
9800 GTX+ SSC
32SSD's RAID 0