Nathan Durrant
017121078
nathan.durrant@sjsu.edu
Programming Assignment 1

Problem 1

Measurements

Array size n=10
Trial 1: Insertion sort: 6001ns. Merge sort: 15900ns
Trial 2: Insertion sort: 3801ns. Merge sort: 10301ns
Trial 3: Insertion sort: 2800ns. Merge sort: 9500ns

n=50
Trial 1: Insertion sort: 27600ns. Merge sort: 43360ns
Trial 2: Insertion sort 28000ns. Merge sort: 38601ns
Trial 3: Insertion sort: 26600ns. Merge sort: 33800s

n=100
Trial 1: Insertion sort: 117400ns. Merge sort: 89300ns
Trial 2: Insertion sort 95801ns. Merge sort: 70801ns
Trial 3: Insertion sort: 89800ns. Merge sort: 70299ns

n=1000
Trial 1: Insertion sort: 2289200ns. Merge sort: 994899ns
Trial 2: Insertion sort 2198899ns. Merge sort: 736700ns
Trial 3: Insertion sort: 2732700ns. Merge sort: 1018800ns

Input array size	Insertion Sort average (ms)	Merge Sort average (ns)
10	0.0042	0.0119
50	0.0274	0.0385
100	0.1010	0.0768
1000	2.406	0.916

The theoretical analysis is confirmed by my results. Insertion sort is in order of O(n^2) while merge sort is in the order of O(nlogn). This means that for larger inputs, the runtime of sorting will grow at a slower rate for merge sort than insertion sort, as it has a lower order of growth. In the table above, this is confirmed when comparing input sizes 50 and 100. For the array size of 50, insertion sort is faster than merge sort on average, however by array size 100, merge sort is faster than insertion sort on average.

Insertion sort begins with a faster runtime, partly because it is an in place sorting algorithm. Without needing to allocate space for arrays, it starts with the lower run time, but for any array size 100 or more, merge sort is faster than insertion sort because it has a lower order of growth. This is also supported when the input array is a bigger size, as shown in the table at 1000. Merge sort is over a millisecond faster than insertion sort in every trial, as well as on average.

Problem 2

Results
After a lot of debugging, I got the result of all 3 test cases passing.
The algorithms “sort by last name” as well as “sort by last name then date of birth” take 0-1ms, while sorting by last name takes 10ms or more. This makes sense, considering that we are working with a small array in the test cases, so insertion and quick sort will be much faster than merge sort. Sorting by last name then date of birth does have to make more comparisons than sorting by just last name, but it does not make a considerable difference in run time.

Besides this, I don't really know what to touch on when it comes to my “results”, so I will elaborate a little more on how this problem went for me in order to have a thorough lab report.
I ran into a fair share of issues when it came to this assignment and a lot of debugging was required. I kept running into stack overflow errors which would usually be fixed by changing a comparison sign. Implementing the sorting algorithms from ints to objects was not too bad, a lot of the logic remained the same, but it just required using compareTo() methods to compare Date and Strings.