Algorithms Project 1 Results

For this project I recorded the best, worst, and average times for 9 different sorting algorithms sorting lists of 100,000 elements whose values ranged between 0 and 10,000, 10 and 1,000,000, and 0 and 100,000,000 over 10 different runs. It was then repeated with 100 runs, and then with 10 runs and only 10,000 elements. (See results.xlsx for more details).

A number of general statements can be made on the data. In every case insertion sort was by far the slowest sorting algorithm. Its best time for any off the range sizes was 2930 milliseconds – the next highest was 35 milliseconds for Arrays.sort() on the small range. Looking at the graphs it is clear that this is the only O(N2) algorithm in our tests.

Of the different quicksort variations that were implemented, version 3 seemed to perform the best, but only marginally. Version 3 used the median of three data points for its pivot and also checked if a subarray was sorted before sorting it. It also had the strongest performance of any of the algorithms with the 10,000 range – possibly because this made the median more efficient or because subarrays were more likely to be sorted. While version three was slightly better, all of the quicksort implementations actually performed quite close to each other. If we look at the average runtimes (see results.xlsx for charts) there are slight discrepancies at the 10,000 range, but for the larger ranges all 5 versions were fairly consistent at 8 milliseconds. Heapsort was also fairly consistent, but slower than any of the other algorithms. Mergesort was also consistent and slower than any others besides heapsort.

Standard quicksort, with the pivot simply the element at the middle of the array, was actually the most consistent algorithm. It had the smallest difference between best and worst times, which isn’t what I would have expected.

Arrays.sort() had some very interesting behaviors. On the smallest range, 10,000, it had a shockingly bad worst case performance, almost twice the next worst (excluding insertion sort). However, once we got to the million and one hundred million range its worst was the smallest of any of the other algorithms. On those ranges, it had the lowest best, worst, and average times.

In summary, if the dataset in question has a very large range the built in Arrays.sort() seems to be the best plan, but if the range is smaller, using quicksort variation 3 seems to be the best.