CS344 Project 1: Sorting Algorithms

Benjamin Lannon & Hunter Quant

March 10, 2016

1 Overview & Implementation

The project's objective was to implement three sorting algorithms we have discussed in class: insertion sort, a deterministic quicksort, and a randomized quicksort. To measure the running times we ran on a set of vectors, we implemented an Integer class which keeps track of every time we do a < comparison.

2 Comparison Table

The below shows results for running these sorting algorithms on a sorted vector of size n (which in our case, N = 100), a reverse sorted array, and the average of 10 vectors with N random elements.

	Insertion sort	Det. quicksort	Rand. quicksort
Sorted vector	5050	5050	5050
Reversed vector	5050	5050	5050
Random vectors (avg)	49680	50329	51338

3 Analysis

As seen in the table, running all 3 algorithms on a sorted vector and reverse sorted vector result in the same amount of $5050 = \Sigma_{i=1}^{100} i = \Sigma_{i=1}^{n} i$ comparisons. Our theoretical analsys suggests that insertion sort runs in $O(n^2)$ time, and the same amount of comparisons occur with quicksort as well, so it seems that they are also running in $O(n^2)$ time, which is expected, but it isn't as fast as the theoretical approach of $O(n \lg n)$ time for the two quicksorts. On the other hand, the average of 10 random vectors result in about 10x the amount of comparisons compared to the sorted and reversed vectors (Note: the calculation amount is not relative to the number of random vectors averaged, as when run on a single vector, it still sits around 50k comparisons). This seems logical since the first two types of arrays are either already sorted or almost sorted, so not many calculations need to be done, while the random vectors need more to become sorted.