Assignment 6 Write-Up

Logan Gantner

December 14, 2018

For this assignment I tested the speed of four different sorting algorithms: bubble sort, insertion sort, quick sort, and cocktail sort (a sort of improvement on the basic bubble sort algorithm). For the purposes of testing I generated a variable number of pseudo-random doubles into an array and recorded the speed of sorting for each algorithm on the data sets. As expected, the difference in speed between the methods depended largely on the size of the array to sort. Results (time to sort by method and size) are recorded below:

	Array Size			
	100	**1000**	**10000**	**100000**
Bubble Sort	0.000048	0.0055	0.51	66.5
Insertion Sort	0.000020	0.0038	0.13	13.1
Quick Sort	0.000015	0.0005	0.0016	0.02
Cocktail Sort	0.000045	0.0089	0.38	39.8

For a relatively small array, such as one of length 100, all of the algorithms perform on the same order of magnitude. As the array increases in size, the methods begin to diverge. Bubble sort is inferior in all cases, with both cocktail and insertion sorts typically a multiplicative factor faster. This is unsurprising, as these are all $O(n^2)$ algorithms. However, quick sort is much more robust to size than its competitors. With an array of one hundred thousand values, even as the other three methods take a minimum of 10 seconds to complete the sort, quick sort performs the same task in less than a fraction of a second. I was expecting quick sort to outperform the others by orders of magnitude since it is a $O(n \log n)$ method, but its speed went beyond my expectations.

There are obvious trade-offs to using quick sort. Of the four methods, it was the most challenging to execute. In addition, it has the arbitrary criterion of which index to choose as the pivot location at each split. A change in this decision could affect the sorting speed of different data sets in unpredictable ways. Insertion sort, on the other hand, should perform roughly the same no matter who implements it. Finally, there are issues with the analysis itself. While it is a fair and unbiased test to draw data randomly from a uniform distribution, it is not applicable to many real-world situations. In many cases input data may be partly or mostly sorted. We have not addressed how this will affect relative performances. We may expect insertion sort to do much better in these incidences.