Jeffrey Allen

Michael Beaver

CS 355

November 20, 2012

Assignment 11, Group 3

Testing Plan

To test the system we have created, the only thing we would need to do is insert data, in which the driver file gives the option to input manually or import from a file. If we did input the information manually, say a data set of size 10, we would enter in the numbers of our choice and use all the different types of sorting algorithms to sort the data. In doing so, the program would then run each method by instantiating a sorting object that is respectively named after its sorting method and sort the data we have input to acquire the different types of information produced by each sort.

The information gathered about each sorting algorithm is collected as the algorithm performs the process of sorting the data. This iteration information includes the number of swaps, loops (inner and outer), merges, recursions, and divisions executed by each sorting algorithm. After each sort is executed on the data set that is given, if the option is chosen to display the information in the program, the iteration information of each sort is output to a table that can be easily viewed and compared to one another. There is also the option of saving the results to file of the users choice prompted by the driver file if the user would like to compare algorithms elsewhere.

In general, any randomly ordered data set should have a runtime relatively similar to the established runtimes of each sorting algorithm. For example, a randomly ordered data set of size 100 should expect to have a runtime approximately 10000 for the N2 sorting algorithms, approximately 200 for the NlogN sorting algorithms, and approximately 100 for the linear sorting algorithms. Ideally, each randomly ordered data set, no matter the size, should have a runtime that corresponds to either O(N2), O(NlogN), or O(N), with respect to each sorting algorithm.

Sorted data should have a runtime of O(N) with BubbleSort. SelectionSort and InsertionSort may still approach O(N2) because these sorts do not have the “early out” system BubbleSort employs. With MergeSort, the runtime should still be O(NlogN), because the data is divided and merged, despite being in order. Likewise, QuickSort should approach O(NlogN). However, the runtime may approach O(N2) based on how the pivot is chosen. Finally, RadixSort always has a runtime of O(N) because it always touches every data item at least once.

Reversed order data will definitely approach a runtime of O(N2) with BubbleSort. SelectionSort may perform slightly better because it performs pairwise comparisons based on finding the minimum in the list. Ultimately, it should approach O(N2), though it might not be exact. MergeSort should have a runtime of O(NlogN) regardless of the ordering because it always divides the data and merges it back together. QuickSort could have a runtime of O(NlogN), or it could approach O(N2) if the pivot is poorly chosen. RadixSort will have a runtime of O(N) regardless of ordering.

While the collected statistical data may not exactly match the established Big-Oh runtimes, but the data should approach those runtimes. If not, the code may be incorrect.