Tom Fenyak

CPSC 3273 Programming Assignment Analysis 05

First I would like to say that this program works and prints out the run times of each MergeSort, QuickSort, and InsertSort (in that order).

To compile this program, I first had to get an array created with random integers (for me I chose between 1 – 100,000 and I will explain why later) with a very large size. This Array is named ‘G’. Then we had to make a second array that would copy **n** amount of integers from the original array and put them in the second array ‘A’.

Next we sort this new array ‘A’ and track how long it takes to sort in milliseconds using System.currentTimeMillis(). We then increment the size of the array ‘A’ by 1,000. We repeat this process until we have reached the full length of our array ‘G’, which in this case was 1,000,000. For InsertSort I capped the array size at 100,010 because due to its lack of scalability, it would have taken an extremely long time to complete the program. We sort the Array ‘A’ using three different methods, MergeSort, QuickSort, and InsertSort. We track the times of each so we can graph and compare the run times.

In our graph below we compare the run times of the three algorithms. Right away one of them sticks out as it is not very scalable in comparison to the other two. That algorithm is InsertSort and this makes sense because on average its time complexity is O(n^2). MergeSort and QuickSort on average run in O(n log(n)). This graph proves that the MergeSort and QuickSort algorithms are much more scalable than InsertSort. What I would expect to see if our size of array was to increase is that eventually QuickSort’s run time would be far more than MergeSort as we approached infinity because in worst case it has a run time of O(n^2) while MergeSort’s worst is O(n log(n)).