Tom Fenyak

Programming Assignment Module 03 Analysis

For this assignment we were to track the time that it took to complete a merge sort of an array with a rapidly increasing sample size. This program does correctly implement the merge-sort algorithm and track the time to complete each for loop.

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 100,000 (explained later). Same process as the last program.

This time however, we performed a merge-sort which proves to be much more efficient than selection sort. The time for my program to run and produce 10 times the amount of output was so much quicker than the previous sort. For merge-sort we use the divide and conquer technique. We split our large array of random ints into two smaller arrays. We then use the merge-sort method to sort the two smaller arrays with mergeSort(). Finally we merge the two together with the merge() class.

Our graph down below takes T(n) / nlog(n) where T(n) is the time it takes to complete the for loop. This allows for us to compare the function directly to nlog(n). The run times remain almost constant throughout and takes a fraction of the time as a normal selection sort does.