ASSIGNMENT 6 -SORTING

Jesse Ruhl

In this assignment, students were told to create a program that sorts a given file with a data set of numbers and sort it using various sorting methods. These sorting methods included Quick Sort, Merge Sort, Insertion Sort, Selection Sort, and Bubble Sort. From my experience, I found that the easiest to implement were Bubble Sort and Insertion Sort. The most difficult sort to implement was the Merge Sort. This is because I had trouble with figuring out which values (left, right, and middle) needed to be in what place. In addition to that, one has to write a merge function that partitions the data and a mergesort function that recursively divides the given list of data into two halves. This was the most time-consuming sort methods to implement the code for. However, Merge Sort took the quickest to sort my large set of numbers. This is because it is most efficient in cases of large array sizes or datasets, and that can be seen with my results of the time it took to sort 40,000 numbers with the time being .001 ms. When looking at the time differences, I found that the differences were like what I expected to be with the exception of Merge Sort. For some reason I thought that would take longer than Quick Sort. But upon further research I found that Quick Sort is most beneficial in cases of smaller array sizes, and since I have a large data set it would make more sense that Merge Sort is faster. The main tradeoffs that I found when choosing one algorithm over another was the implementation process. As stated earlier, I had a difficult time with Merge Sort. However, even though one of the easier methods to implement was Bubble Sort, it has a worse time when compared to Merge Sort. Therefore, easier code does not always mean that it is efficient to process. Another tradeoff that I found was the runtime complexity of each code. When running my program I found that I had to wait the longest time for the Selection Sort to get done sorting. Some of the shortcomings of this empirical analysis were that it was not very cost efficient and that the algorithms needed to be implemented to see the time differences between each sorting method. Another shortcoming is that the runtimes are dependent of the compilers used to implement the algorithm. The language and compiler I wrote in may implement one sorting algorithm well but implementing it on another language or compiler could result in slower times. Therefore, this project may only be rough estimate to see what sort methods are superior to others. However, overall I found that this program was very interesting and provided lots of insight of how important time complexity and runtimes are when it comes to programming.