Blake Rusteberg

February 2, 2019

Project 1 Sorting

There are 4 plot graphs listed below that show a detailed representation for runtimes of the sorting algorithms implemented on permuted and sorted inputs. The runtimes were tracked in milliseconds and then converted to seconds for a cleaner output. The sorting algorithms used were Insertion Sort, Merge Sort, Heap Sort, and Build Heap. Graph 1 displayed Insertion Sort and Graph 2 displayed Merge Sort, Heap Sort, and Build Heap. Graph 1 and 2 display the outputs from the given permuted inputs.

Insertion Sort at its best is linear time O(n), base on the size that is being checked. Its worst case is . To achieve the worst-case scenario for insertion sort, the algorithm must run through its loop the maximum number of times, thus resulting into a quadratic time base scenario. It is clear in Graph 1 that the graph grew exponentially as inputs grew larger in size. In Graph 3 the graph grew in linear time achieving the best scenario for this sort. It did this because all the words have already been sorted in the text file.

Merge sort is expected to run at O(n log n) time. It takes O(1) time to run though the array and split it in half. Each subarray gets sorted recursively and “n” of the tree height gets divided by the leaf level. Once all the levels are divided and sorted, they are put back in a similar way they were taken apart taking O(n) time. This gives us the equation T(n) = 2T(n/2) + O(n). Merge sort should demonstrate an even to steady rise as the list size grows larger. Graph 2 displays this growth.

Heap Sort is expected to run at O(n log n) at its worse. The same complexity as Merge Sort. The best case for Heap sort would be seen if it was implemented on a fully sorted list. The algorithm can be split into 2 parts, whenever that heap is built O(n) and when the heap is sorting O(log n) times. Graph 2 represents this behavior accurately, although compared to merge sort in an unsorted list performs a little slower. When the list is already sorted they sort like each other.

Build heap is performed at O(n) time and only occurs when building the heap. Its worst case is also O(n) time, because it performs the same throughout.

Graph 3 and 4 are graphed from the given sorted inputs. All the sorts performed at O(n) time when ran on the sorted files.

Insertion Sort performed best for the first 4 text files than gradually fell to perform the same or worse like Heap and Merge Sort. It does this because it only must run through the list once.

Merge Sort’s behavior changed very little when given a list of sorted inputs. This is because the algorithm must take time to split and reassemble the list, even if it is already in order.

Heap Sort acted like Merge sort when given a sorted list. The reason it is a little slower than insertion is because it must build the heap first then perform the sort.

Build Heap was affected very minimally. It was still faster than build permutated inputs but begins to rise gradually the greater the input size.