CS 340 Project I

Sorting and Heaps

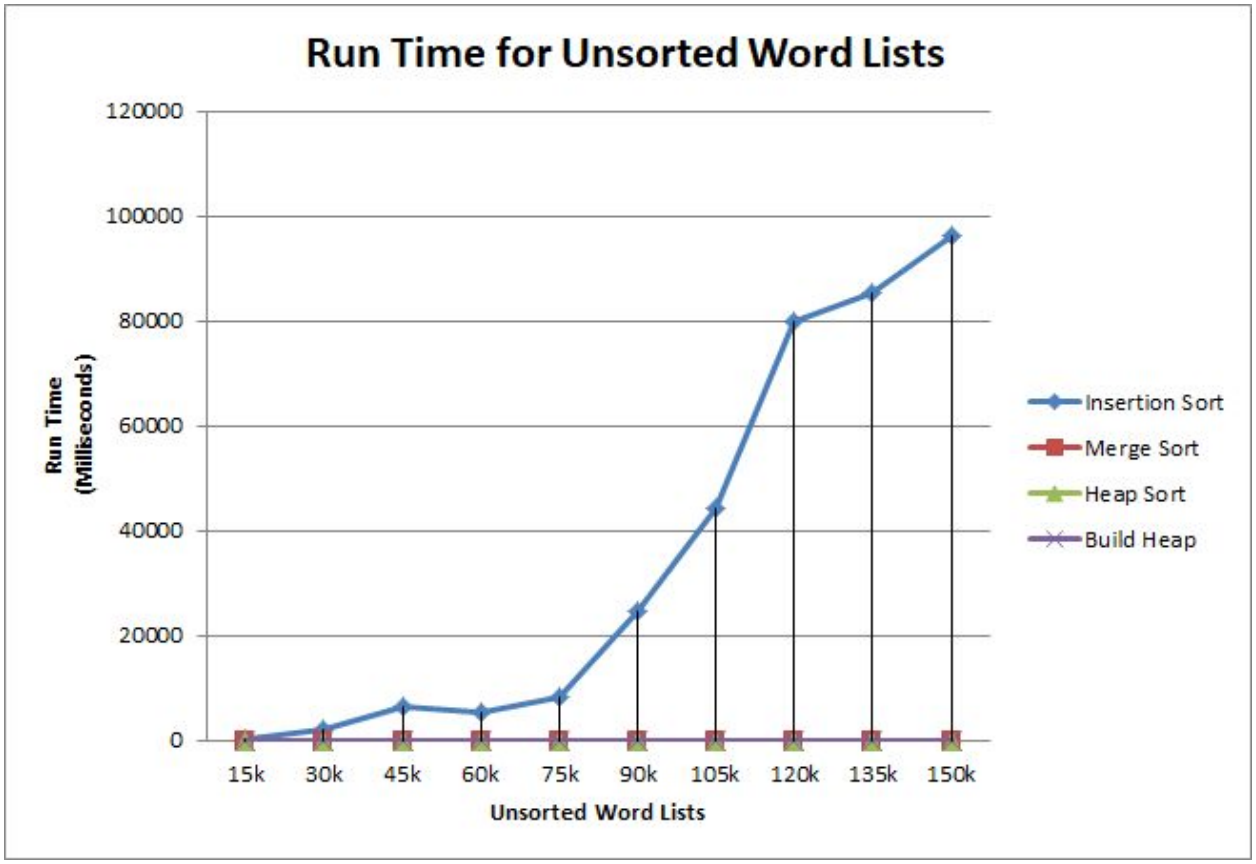
Jordan Kramer

Algorithms and Data Structures 340

John Matta

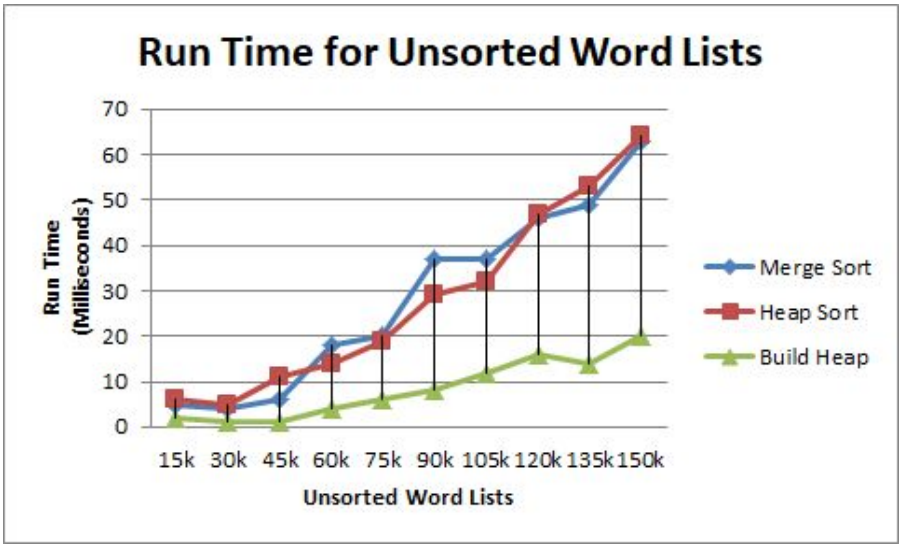
September 4th, 2018

Looking into the first graph of the unsorted word lists, one will find that insertion sort rises highly above all of the other sorting algorithms. This is due to the fact that insertion sort's expected average case time complexity falls into the case of Θ(n^2). This turns out to be the longest of all the run times because, looking at the graph, it ascends at a rate of n^2.



Merge sort's expected average case is Θ(n log(n)). In the graph above, this is actually a lot faster than insertion sort. The graph below shows an enlarged view of merge sort, heap sort and build heap.

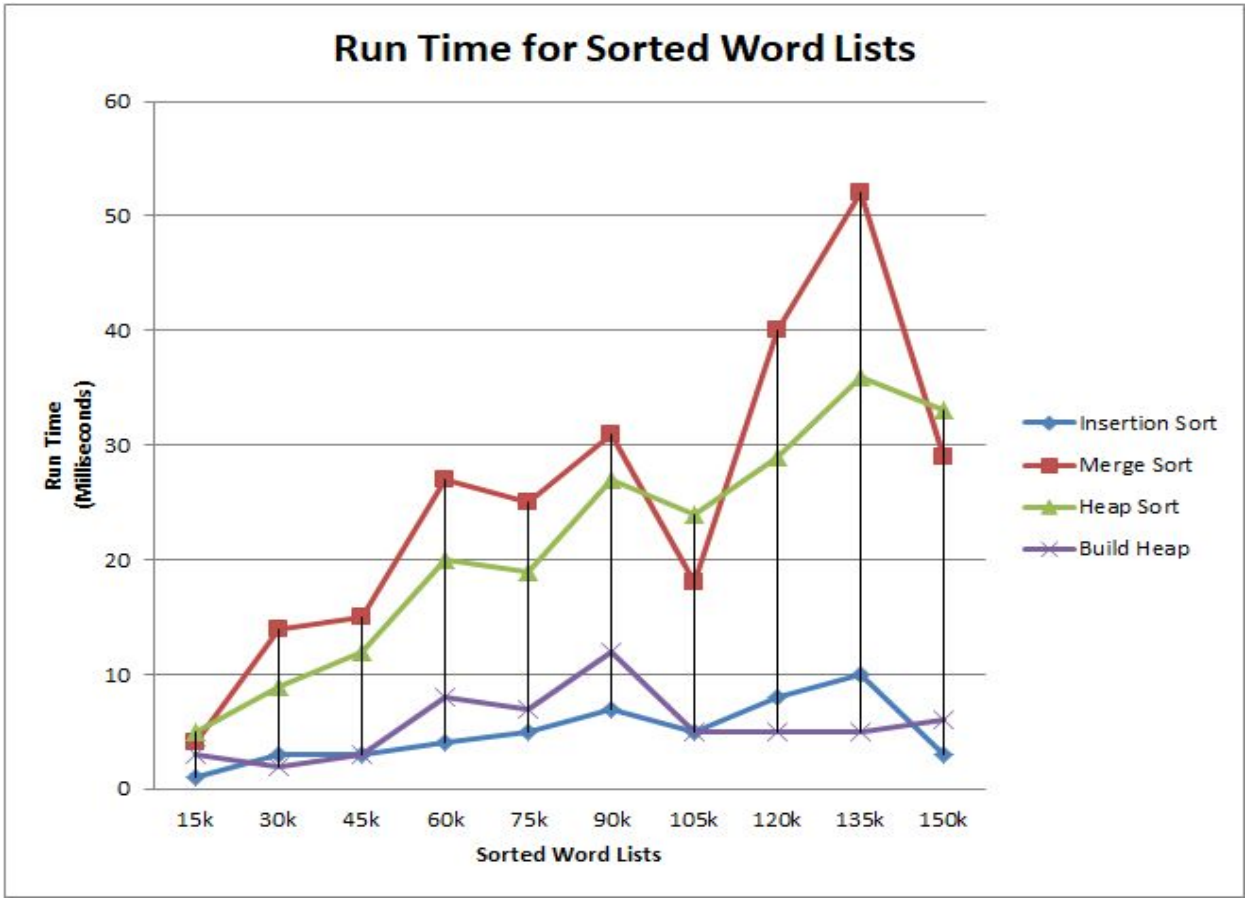
In the graph, merge sort's path actually closely resembles that of heap sort. Both of these two sorts follow the n log n path.



Since the plots show that merge sort and heap sort have similar run times, it must be correct. This is because heapsort's expected average case time complexity is Θ(n log(n)). Then at the bottom one can see how build heap is the fastest of all four sorting algorithms. Build heap's expected average time complexity is Θ(n). The graph clearly shows the correct run time for build heap because build heap is shown running faster and that should be assumed because Θ(n) is faster than Θ(n log(n)). The data for my build heap also follows the expected behavior because build heap's line raises at a linear rate.

Venturing into the sorted word lists we should expect to see insertion sort change drastically. Since all of the words are in sorted order, this is the best case for insertion sort. In this case, insertion sort's expected time complexity should be Θ(n).

From what is seen above, build heap's expected time complexity is also Θ(n). This does not change the time complexity because sorting the data doesn't effect build heap. In the graph below, insertion sort and build heap's run times are similar for each different word list and they both raise at a linear rate.



Merge sort and heap sort turn out to be slightly similar to the unsorted graphs. Merge sort's best case time complexity is the same as its average which is Θ(n log (n)). This is the same case for heap sort. The graph shows that merge sort and heap sort have nearly identical run times and they they follow a n log n path for each word list which is to be expected. This is correct because when looking at the data, heap sort and merge

sort both take longer than insertion sort and build heap. That should hold true because Θ(n log (n)) will take more time than Θ(n).

In conclusion, when looking at the results, insertion sort's best case time complexity is Θ(n) and its worst case is Θ(n^2). Merge and heap sort's best and worst case both turn out to be Θ(n log (n)). Finally, build heap continues to be Θ(n) for its best and worst case.