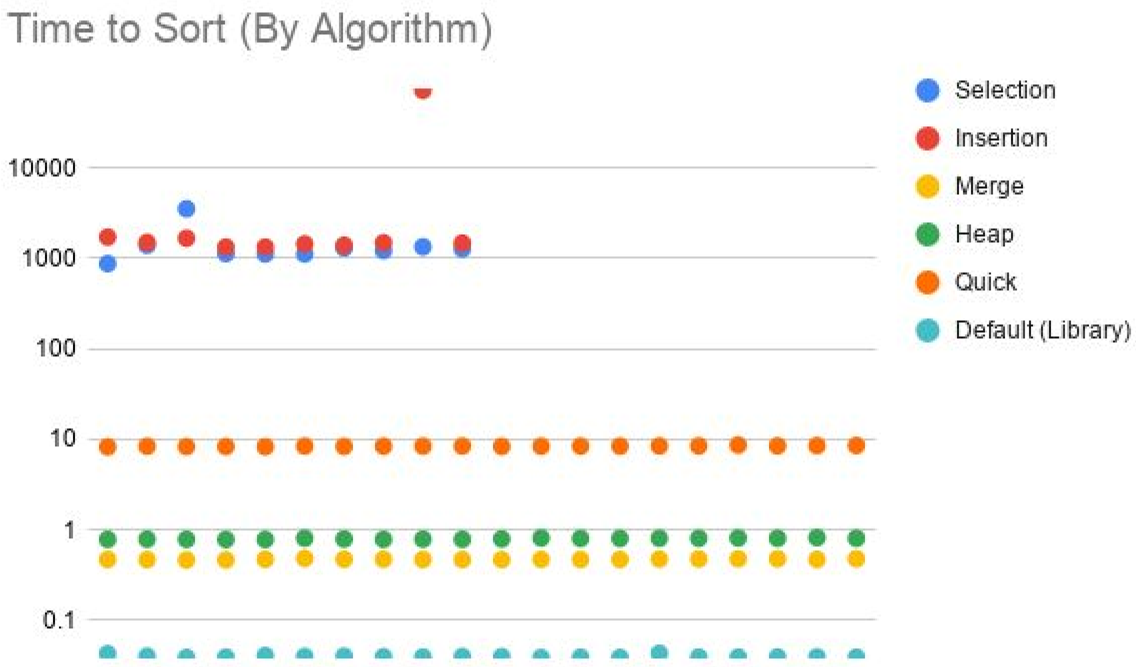
Chris Zhang

Exercise 2: Sorting Algorithms



The chart above shows the results of testing five sorting algorithms (selection sort, insertion sort, merge sort, heap sort, and quicksort) against the default library implementation of sort() in Python.

Insertion sort was the worst-performing, at an average of 8,536 seconds per run, with a standard deviation of 22,293. Since the time complexity for insertion sort is O(n^2), a long run is expected. It most likely performed significantly worse than selection sort because the input data was both mostly unsorted and extremely long, requiring a high number of swaps per step.

Selection sort performed the second-worst, with an average of 1,433 seconds per run and a standard deviation of 755. Like insertion sort, it has O(n^2) time complexity.

Quicksort required an average run time of 8.40 seconds, with a standard deviation of .09. The average time complexity of quicksort is O(nlogn), which is reflected in quicksort's much faster performance compared to insertion and selection sort. However, despite being expected to perform better than or equal to heap and merge sort, it likely performed worse in this situation because of the overhead of the iterative implementation rather than a recursive one, in order to avoid stack overflow.

Heap sort required an average run time of .80 seconds, with a standard deviation of .01. Like quicksort, heap sort has O (nlogn) complexity and performed relatively quickly despite the size of the input.

Merge sort performed the fastest, with an average run time of .47 seconds and a standard deviation of 0. Like heap sort and quicksort, it runs in O(nlogn) time.