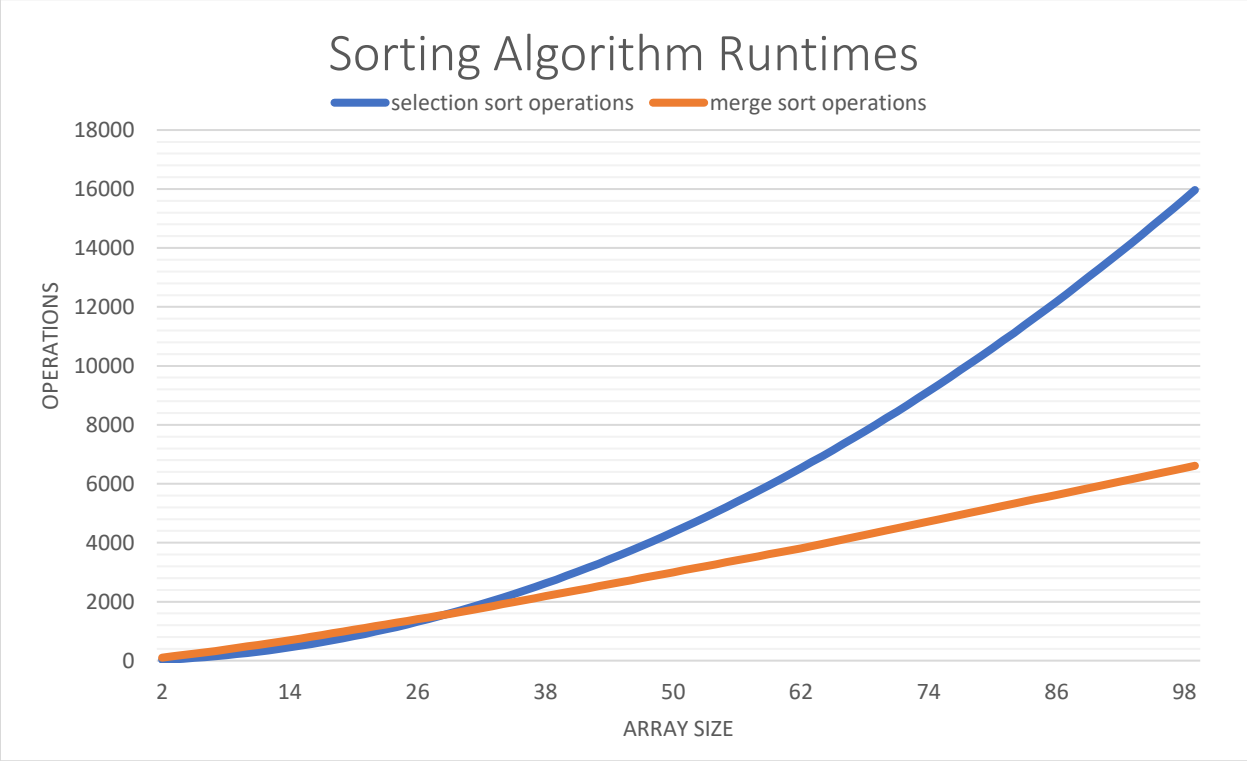
Dr. Don Evans
CSE 1342.001
19 March 2019

Selection Sort and Merge Sort



The growth patterns in the chart above match expectations for selection sort and merge sort. Merge sort requires additional memory to perform its operations, so for small data sets it is outperformed by the simpler selection sort. The divide-and-conquer approach of merge sort requires additional operations to allocate auxiliary memory and operate within it. However, selection sort iterates over the elements of the array and sorts them in place.

It is expected that an algorithm with $O(n^2)$ complexity would outperform another with $O(n \log n)$ complexity for smaller values of n. But the recursive approach of merge sort heavily outperforms the iterative selection sort for large data sets. In this specific implementation of these two algorithms, merge sort begins to outperform the quadratic runtime complexity of selection sort starting around an array size of 29.

Selection sort divides an array into a sorted and unsorted portion, where the entire array is considered unsorted to begin with. The algorithm iterates over the values of the unsorted portion to find the index of the smallest value, then switches it with the element at the lowest index of the unsorted portion. That index is then added to the sorted portion and the remaining unsorted portion is iterated over until the whole array has been sorted.

Merge sort recursively divides the array in half until what remains is a series of single-element arrays, which by definition are already sorted. The algorithm then merges these divided arrays one element at a time by comparing the largest elements of each. The merged arrays are then merged with other merged arrays until a single sorted array is achieved.