Sorting a list of numbers is a serious problem in Computer Science. While it may seem straightforward at first thought, the fact is, when working with large lists of numbers, it takes a long time to sort them by standard means. This was clearly exhibited by the results of this lab. This lab also showed that by using more creative tactics, the power of recursion can be leveraged such that what takes hours to do the “normal way” takes only seconds.

In this lab, four different sorting algorithms were tested on random arrays of sizes from 100 to 10,000,000 in multiples of ten. These algorithms were as follows: Quicksort, Mergesort, Insertion Sort, and Bubble Sort. The first two have best and average cases time-complexities of O(n log(n)), with Quicksort’s worst case being O(n^2) whereas Mergesort’s stays O(n log(n)). The other two have best cases of O(n), but only on a list that’s already in order. Average and worst cases are O(n^2). As this lab used arrays of random numbers, the best and worst case are unlikely, and therefore the average is the most relevant. This means we are most concerned with the first two averaging O(n log(n)) and the last two averaging O(n^2). Therefore, the first two should be considerably faster than the second two.   
  
The results (in seconds) are as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 100 | 1,000 | 10,000 | 100,000 | 1,000,000 | 10,000,000 |
| Quicksort | 0.002 | 0.007 | 0.021 | 0.090 | 1.394 | 35.245 |
| Mergesort | 0.003 | 0.006 | 0.017 | 0.151 | 0.649 | 8.372 |
| Insertion Sort | 0.002 | 0.018 | 0.097 | 8.913 | >30 min | N/A |
| Bubble Sort | 0.004 | 0.030 | 0.403 | 39.294 | >30 min | N/A |

This translates to a graph as follows:

This isn’t a great visualization, but it does show how the Quicksort and Mergesort scale a lot better than Insertion Sort and Bubble Sort. The most surprising thing about these results is that Quicksort seems to stop being so effective for the higher numbers. In fact, in general use, Quicksort is usually slightly better than Mergesort. Perhaps the algorithm truly intends to use the median of all elements as the pivot, whereas this implementation used the median of the first, middle, and last elements. This seems unlikely, as the calculation of the median of all elements is at least O(n), which would make the algorithm as a whole less efficient. Upon further experimentation, attempting to use the first element as the pivot decreased performance.

In attempting to find why this Quicksort was so inefficient, other tests were done. For example, the same code was run multi-threaded, but it didn’t fare much better, at ~32 seconds to the single-threaded’s ~35 on 10,000,000 elements. This shall have to remain a mystery.

Below is a clearer visualization of the sub-second times.

As can be seen, under 1,000 elements it doesn’t much matter which is chosen. However, as things scale, it is quite obvious that some sorts are better than others.  
  
In the end, one thing is clear: to make code that won’t take forever to run, one must be clever about how they perform their task.