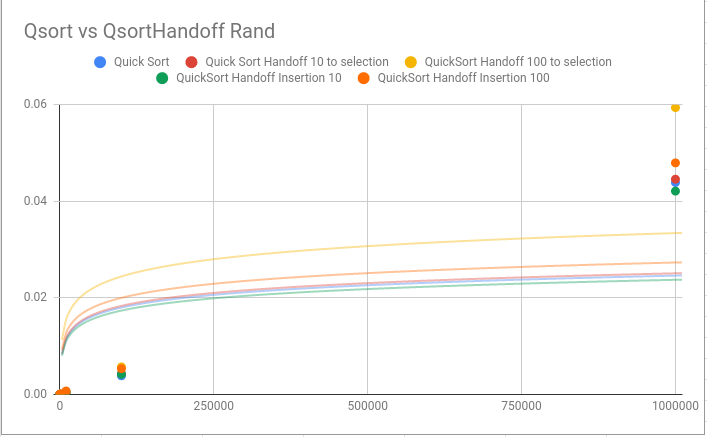
Christian Gould

CS 3353

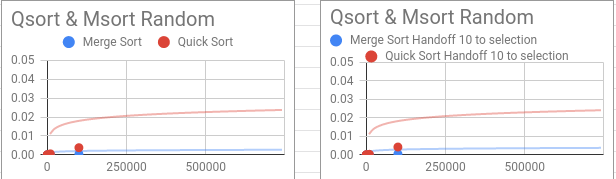
Sorting Findings

In the sorting of all these objects, the themes of all of it tends to stay the same. Algorithms which have O(n^2) tend to be very significantly worse than that of algorithms in the O(nlg(n)) category. So much so, that while making this project, I actually had to remove the functionality of running 1,000,000 sized arrays on: insertion sort, cycle sort, and selection sort, because otherwise running the program would take way too long. This alone is evidence enough to show that we as computer scientists need to have a lot of attention to detail, and make sure that we use the correct tools for the job. However, in this project, I discovered that certain tools combine well together. One finding which surprised me a lot was how well insertion sort synergized with both quicksort and merge sort. I was very pleased to find that it worked better together, because after testing a lot with selection sort, and having no such luck, it was very discouraging. The one which shows this very plainly is the one which displays quicksort, with the y-axis being seconds, and the x axis being number of items sorted:



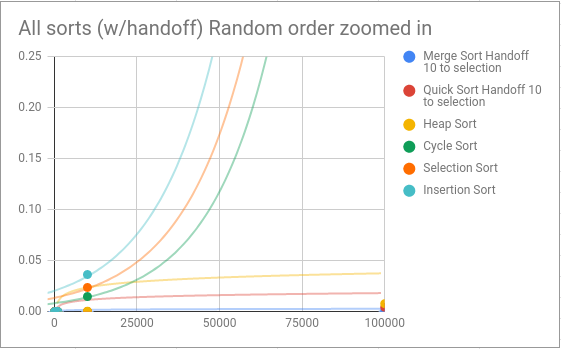
Here, this chart shows that there is actually a difference when using multiple tools together. Insertion sort, which works well with small datasets, helps quicksort, which is good at working with large datasets.

Quicksort and mergesort sorted in a very similar manner as regards how they responded to using a handoff with selection sort. Here is the graph for that relationship with the y-axis being seconds, and the x axis being number of items sorted:



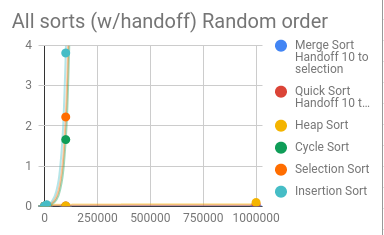
In this chart, it is clear that there is virtually no difference with selection sort and quicksort when handing off to 10 elements with a random dataset.

Now, the program obviously does not only cover quicksort and mergesort. As regards the whole family of sorts, these were the results for execution time with the y-axis being seconds, and the x axis being number of items sorted:



Now, this is zoomed in a bit to show detail, but it gives a clear picture as to what is going on. Merge sort, quick sort, and heap sort, all O(nlg(n)) sorts, do not gain so much in execution time as the dataset size gets large. However, cycle sort, selection sort, and insertion sort do. Selection sort and cycle sort actually start out below heap sort with a very small dataset, and cycle sort comes close to touching quicksort near the beginning. This is evidence that there is at least some salt in the idea that using a handoff could be useful.

Once we zoom out of that picture, it becomes much more clear that for large randomized datasets, we simply should not use cycle sort, selection sort, or insertion sort. They take far too much time compared to heap, merge, and quick sorting methods with the y-axis being seconds, and the x axis being number of items sorted:



The three O(n^2) algorithms nearly shoot straight up after we zoom out of the graph. These sorts should only be used for very specific types of data, and for times when we know that datasets are in order, or pseudo ordered, so that they do not have to make as many comparisons.

Overall, there is a lot to be learned from simple guess and check research. Overall, it is important to understand that no algorithm presented here is worthless, and each has its merits, it is simply up to us as computer scientists to decide which tool is right for the job.