In this assignment, I tested six different sorting methods on several sizes of arrays. These arrays were sorted in one of three ways: random, sorted low to high, or sorted high to low (reversed). I found that the organization of the data had a profound effect on the run time of the sorting algorithms. Across all data configurations, the simple quick sort algorithm had the worst performance, quickly rising to run times of over 20 seconds for array sizes of just 100k. The median of 3 quick sort algorithms were also less than ideal in run time. They also performed differently based on the data configuration, that is, different base cases seemed to work better for different data configurations. Specifically, larger base cases worked more efficiently with sorted arrays. This pattern was seen to a lesser extent with reverse-sorted arrays. The random pivot algorithm seemed to have the best overall times. In a random array, the random pivot was by far the most efficient algorithm, and merge sort was the worst (excluding simple quick sort). This is true for reversed arrays as well, but all times were much closer in this case. It is worth noting that the runtimes for merge sort seemed to grow more quickly as the array size grew. In the case of a sorted array, merge sort was actually the most efficient, suggesting that the merge sort algorithm deals with sorted arrays more efficiently than the others. Given that this is the only case that the random pivot algorithm was not the fastest run time, I can conclude that the random pivot algorithm is the best of the six for general purpose use.