Were the time differences more drastic than you expected?
I have done this project 3 times now for multiple different classes and everytime the difference between the recursive sorts and the normal sorts makes me take a step back. I actually checked to make sure my timer was working even though I have seen these results before. I think it is so cool how a simple trick like recursion makes the program so much faster.
What trade-offs are involved in picking one algorithm over another?
Bubble Sort: This is your quick and dirty algorithm. It gets the job done if you need a sort and don't have access to the internet but it is worse than any other sorting method.
Selection Sort: This is the best algorithm (from the 5) that isn't recursive. You want to use this algorithm if no recursion is allowed and the list is completely unsorted.
Insertion Sort: This algorithm is special because if a list is partially sorted it is by far the best algorithm with a big O runtime of O(n). So if any list is partially sorted you should use insertion sort.
Merge Sort: Is a recursive sort. Whenever it is allowed it is better than the three non recursive sorts (other than the edge case with insertion sort). It is better than quick sort when there are a lot of numbers to sort and there is not a lot of extra memory for the sort.
Quick Sort: Is also a recursive sort. It is better than merge sort when there is limited memory for the sort. It can also be better if there are less numbers to be sorted overall.
How did your choice of programming language affect the results?
One thing that is special about c++ that Java doesn't do is it guesses what the next value is using branch prediction. This makes it much easier for the program so sort already sorted lists. A good explanation of it is by describing it as a railroad. I don't want to butcher the explanation so here is a really good stack overflow definition of why c++ is better in this situation.
https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array
This also means that c++ is a bit slower than java for an overall unsorted list.
What are some shortcomings of this empirical analysis?
The problem with empirical analysis is that it doesn't cover all the differences with sorts. I feel the best example of this is with swaps. Although we are able to get a good look at the speed of the methods we may also want to limit the amount of swaps we have in our program. With only the speed we are unable to see which algorithm uses more swaps. We also are unable to see which algorithm uses the most memory. Although we can figure out that merge sort uses more memory than quicksort if we look at the code we are unable to tell this from just looking at the speed differences.