Project 1: A Battle of Sorts

In this project, we compared the time it takes for insertion sort and selection sort to sort various arrays into ascending order. These arrays have cell counts that range from 1000 to 10000 cells, and are in ascending order, descending order, or a random order. Insertion sort looks at each cell one at a time and performs any necessary swaps so that its value is correctly sorted for all of the cells that have already been seen. A swap is essentially switching the values of two cells. This means that insertion sort only checks each cell in the array once, but it does not have a fixed number of swaps. Selection sort goes through the entire array and finds the smallest value, then performs only one swap to make it first. It repeats this for every value, finding the next smallest value and swapping it to second, and then third, until the array is ordered correctly. This means that regardless of the order, it must search through the entire part of the array that has not yet been checked n-1 times, with n being the length of the array. It is n-1 and not n, as the remaining value at the end must already be the largest value. In addition, searching accounts for most of the time that selection sort takes, with swaps making only a minor difference.

All of the combinations have timings that grow at increasing rates, except for increasing insertion. This is because with the other five combinations (increasing, decreasing, and random selection and decreasing and random insertion), for every cell added to the length, on average, there are more steps added. For example, adding one more cell to an array of length 10000 and then using selection sort would not have one more step, but instead, it would have 9999 more steps, as it would need to search that cell once for every value in the list, except for the last value. The reason why increasing insertion grows at a fixed rate is because, regardless of how many cells long an array is, adding another cell will only require insertion sort to perform one more step, as it is only looking through each cell once and not making any swaps.

For the reason listed above, increasing insertion sort performs much faster, in fact, hundreds or even thousands of times faster, than increasing selection sort. Neither sort needs to perform any swaps, but insertion sort only performs the check on each cell once. On the other hand, selection sort still has to search through the array n-1 times. Each search also requires more steps the longer the array is, so as n grows, it has an increasing number of steps added. This difference can be seen very clearly in the graph below.

A picture containing flying

Description automatically generated

With decreasing arrays, selection sort performs slightly faster. Now, insertion sort must perform the maximum number of swaps, as each value must be swapped from its cell to the first cell. This is because each value that insertion sort sees is the smallest value it has seen. Selection sort, despite needing to search through the array n-1 times, it only has to perform n-1 swaps, which is what accounts for it taking only slightly longer than increasing selection sort. Insertion sort must perform one more swap for every cell it sees, finally reaching n-1 swaps in the last step. This means that, although selection sort had to search through the array many more times, insertion sort had to swap many more values.

A screenshot of a cell phone

Description automatically generated

With random arrays, insertion sort seems to barely outperform selection sort. Again, selection sort takes almost the same amount of time as before, as it needs to search through one time for every cell in the list. With insertion sort, since the order is random, there will be a lot of variation in the number of swaps needed for each iteration. Although selection sort will require fewer swaps, random order means that insertion sort only performs, on average, about half the number of swaps as with a decreasing array. As seen in the graph, random insertion sort also takes about half as long as with decreasing insertion sort.

A screenshot of a cell phone

Description automatically generated

As we can see, the two sorting methods function at various speeds, depending on the initial array. Insertion sort performs much faster with arrays in ascending order, and slightly faster with arrays in random order. Selection sort performs slightly faster in descending order, but it also has much more precision with its timings. The fastest sort is increasing insertion sort with timings in the fractions of seconds, while the slowest is decreasing insertion sort with timings above 10 seconds.

