My name is Chris Gryboski, and I am a second year Computer Science and Finance major at SMU. Hackers broke into my professors’ sorting algorithms and scrambled them, leaving the algorithms with the names mystery01 to mystery05. My task is to unscramble the mystery sorts by running arrays in different orders through the mystery sorts to uncover each mystery. When I ran the different arrays through the mystery sorts, I recorded the time it took for each function to sort the array and then reset the array to its unsorted form after each call. To ensure accuracy, I ran the tests 20 times to get an average time for each sorting algorithm and used an array of size 75,000 to get noticeable differences in sort times. The first array I used was an already sorted array, the second one was in reverse order, and the third array was randomly sorted before each test. Following these tests, I have concluded that mystery01 is merge sort, mystery02 is optimized bubble sort, mystery03 is insertion sort, mystery04 is quick sort, and mystery05 is selection sort. I decided upon merge sort as mystery01 because all in all cases, merge sort is O(n log n), and mystery01 has low times despite the large size, and the time remains relatively unchanged no matter how the data is sorted, ranging only from 0.002418769 to .015320153. After deciding on merge sort, I moved on to mystery02: bubble sort. I chose optimized bubble sort because mystery02 had the worst time for reverse order while ranking first in the already sorted array. Bubble sort’s best case, when it’s already sorted, clocks in at O(n), but in reverse order it turns into O(n^2) which explains the sudden change in mystery02. A similar change occurred with mystery03, going from 0.000082127 to 1.700732119 seconds when the data was switched from sorted order to reverse order. Mystery03 performed better in comparison to mystery02 when the data was in random order though (1.765465052 compared to 16.20964065), so even though insertion sort’s worst case is also O(n^2), it is mystery03 instead of mystery02 because one property of insertion sort is that it deals with large random data sets better than bubble sort (See chart below).

Chart, line chart

Description automatically generated

After finalizing mystery03 as insertion sort, I decided mystery04 was quick sort. Quick sort performs worst when the largest or smallest element in the array is chosen as the pivot, and because our quick sort had the last element as the pivot, the worst cases would be the sorted and reverse order array, but mystery04 was significantly faster for the randomized array, which coincides with the properties of a quick sort with the last pivot. The final sort, mystery05, had to be paired up with selection sort after all the others had been chosen, but selection sort matches with mystery05 because selection sort has a best case of O(n^2) and a worst case of O(n^2), and mystery05 had the lowest variability of the mystery sorts with O(n^2) complexity.

Mystery Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sort** | Mystery01 | Mystery02 | Mystery03 | Mystery04 | Mystery05 |
| **Answer** | Merge Sort | Bubble Sort | Insertion Sort | Quick Sort | Selection Sort |

Data below:

Table

Description automatically generatedTable

Description automatically generated

Table

Description automatically generatedA picture containing table

Description automatically generated

Sort Graph:

(mystery02 off page at 16 for random order)