Caleb Moore
3/18/21
Data Structures

Problem:
You won't believe this! Some scoundrels have mixed up and hidden all the implementations of the professors sorting algorithms! No need to worry; I think I can figure this out.

Here is what I am thinking:
I plan to run several different sized arrays through these mystery sorting algorithms. Not only will they be different sizes, but I will make a run of them in ascending (correct) order, random order, and descending (backwards) order. After running these, I will have them output their runtimes into the charts below. Finally, I will use what I know about the time complexity of these different sorting algorithms to analyze how they compare with each other to decipher their identities!

ascending order:

```
/Users/caleb/Documents/GitHub/21s-pa03-calebm00re/cmake-build-debug/PA03_sorting_mystery
elements       mystery01         mystery02         mystery03         mystery04         mystery05
10000        0.000004260       0.040481192       0.000432450       0.000023709       0.016143070
20000        0.000007085       0.140221872       0.000599952       0.000016490       0.061241714
30000        0.000010343       0.305667740       0.000778350       0.000023368       0.133964046
60000        0.000021280       1.225903185       0.001447732       0.000046334       0.550710445
100000       0.000037278       3.397664419       0.002366755       0.000091673       1.521260132

Process finished with exit code 0
```

random order:

```
/Users/caleb/Documents/GitHub/21s-pa03-calebm00re/cmake-build-debug/PA03_sorting_mystery
elements       mystery01         mystery02         mystery03         mystery04         mystery05
10000        0.090212168       0.000488444       0.000811520       0.014428891       0.017052977
20000        0.414559788       0.001098040       0.001583267       0.055412818       0.068259205
30000        0.978126576       0.001504241       0.002467382       0.114873555       0.134068527
60000        4.089767753       0.002962911       0.004888143       0.450776362       0.537094822
100000       11.212365870      0.005466208       0.008914889       1.390184961       1.700907924

Process finished with exit code 0
```

descending order:

```
/Users/caleb/Documents/GitHub/21s-pa03-calebm00re/cmake-build-debug/PA03_sorting_mystery
elements       mystery01         mystery02         mystery03         mystery04         mystery05
10000        0.052220548       0.029192975       0.000290414       0.028855643       0.018631286
20000        0.192119859       0.113173341       0.000521893       0.111778956       0.068341766
30000        0.410467933       0.239575086       0.001072003       0.261235672       0.152884872
60000        1.690868041       0.970912629       0.001604004       0.937600587       0.607780932
100000       4.400981735       2.622403740       0.002580690       2.587283090       1.722675940

Process finished with exit code 0
```

Here are the conclusions I came to:

mystery01---optimized bubble sort

mystery02---quicksort (last element as pivot)

mystery03---merge sort

mystery04---insertion sort

mystery05---selection sort

The first two I figured out were selection sort and merge sort. I decided on these two first as their big O's were the same across all the cases (worst, best, and average). This, I figured, meant that the order in which the elements were pre-arranged would not matter. As the charts show, both mystery sort three and five are relatively the same no matter the order of pre-arranged elements, therefore indicating the approximate same number of operations are happening. Between these two, it was just a matter of finding out which was n log n and which was n^2. This was not hard as mystery sort five was clearly always taking longer than mystery sort three, so it should be the selection sort and mystery three would be merge sort. Next, I decided I would try and locate the quick sort. I knew that with the pivot being at the end, the worst cases would be the arrays in order (ascending and descending), and that it would most likely work more quickly with a random number being in the last spot. To find this one, all I needed to do was locate a mystery sort where the randomly generated array was quicker than the two in order. This led to quick sort being quickly identified as mystery two. This only left the bubble and the insertion. These ones were the trickiest as they have the same O(n) for the one in ascending order and O(n^2) for the random and descending. What I had to look for here would be which took longer on the average case. The mystery sorting using insertion should be shorter with a random amount of elements because it could stop comparing when it located the element's place in the sorted list, while the bubble sort would have to continue comparing all the way through. Looking at my charts I made, I could tell that mystery four was much quicker on the average while mystery one had to take a little while longer. This led to me identifying mystery one as bubble sort and mystery four as insertion sort, and now they are all identified. Take that hackers! (hopefully)

used sources found at https://www.bigocheatsheet.com (mainly their charts for time complexity)