Helen Yang

CS 315 – 001

Due: February 14,2023

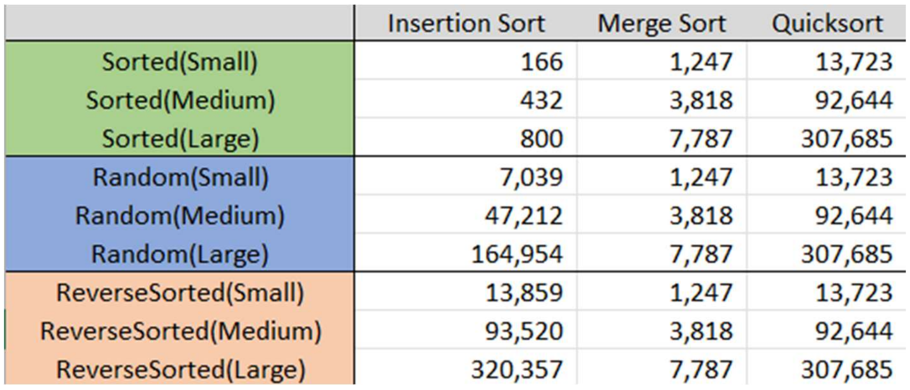
Sort Pokémon!

This program takes in your data file with data on Pokémon and their total stats. Using merge sort, quicksort, or insertion sort, you will result a list of the Pokémon and their stats in non-decreasing order. To run this program, download the zip file provided and unzip the file. In the file, you'll find a README file, sample data files, an executable file (sort), and the source code (sort_main-1.cpp). Then, to run this program, type in the command line in your terminal where this file is located:

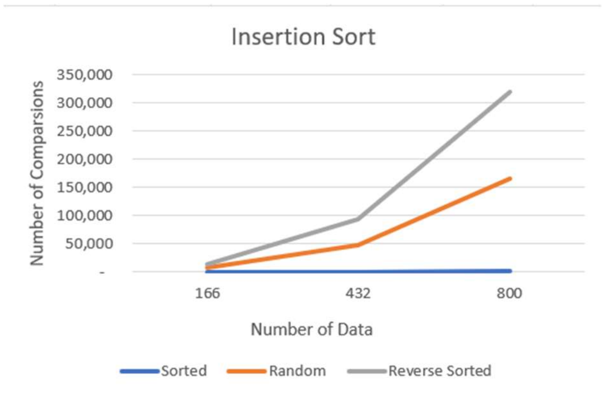
```
./sort <csv_filename>
```

The csv file could be one that's provided in the zip file or one of your own. This will start the program. The program allows you to choose which algorithm you would like to use to sort the data. It will show the data before its sorts, and the data order after its sorted. To quit the program, enter in "q" or "Q" when asked for that algorithm you would like to use.

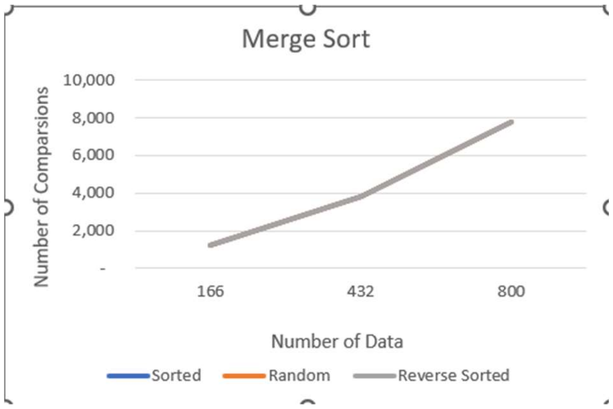
The purpose of the difference types of algorithm choices provided is the runtime or speed in which the algorithm could get the data sorted. The table and graphs below show the number of comparisons done in each algorithm while running the sample data. (Note: Small = 166 data sets, Medium = 432 data sets, Large = 800 data sets)



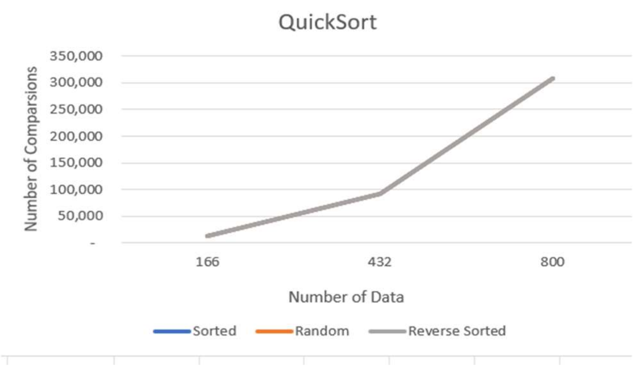
	Insertion Sort	Merge Sort	Quicksort
Sorted(Small)	166	1,247	13,723
Sorted(Medium)	432	3,818	92,644
Sorted(Large)	800	7,787	307,685
Random(Small)	7,039	1,247	13,723
Random(Medium)	47,212	3,818	92,644
Random(Large)	164,954	7,787	307,685
ReverseSorted(Small)	13,859	1,247	13,723
ReverseSorted(Medium)	93,520	3,818	92,644
ReverseSorted(Large)	320,357	7,787	307,685



The insertion sort data shows that the best-case scenario where that data is already sorted is O(n). Then if its randomly sorted before the program, the runtime wouldn't be the worst-case scenario, but still slower that a set of that that's already sorted. Looking at the graph to the left, we could see a parabola starting to form with the random and reverse sorted data set. This would allow us to predict that the runtime of an average random and reverse sorted data set is $c_1 * n^2$ and $c_2 * n^2$ where c_1, $c_2 > 0$. Therefore, the runtime of insertion sort is O(n^2).



Next, looking that the data and graph of the merge sort, we could see that whether the data is already sorted, randomly sorted, or reverse sorted, the number of comparisons would be the same. This tells us that the best-case runtime equals the worst-case runtime. Looking at the data collected, the runtime for merge sort is O(n*log(n)).



Finally, the number of comparisons shows that there are an equal number of comparisons made whether its sorted, random, or reverse sorted. The data shows a runtime of $O(n^2)$. Although this behavior does seems normal given if it's an average or worst-case scenario, the data collected does not behave as expected for the best-case scenario where the data is already sorted. In the best-case scenario, the runtime for quicksort should be O(n*logn).