Daniel Gregory

Assignment 1 Write Up

|  |  |  |  |
| --- | --- | --- | --- |
| The main function of this program first creates a vector of all the data sets so the program can use a for loop to run through them. For this to work all the data sets need to be unzipped and placed into the same folder as the code is so it can find the files easily. Next the program runs through a for loop, where it opens an excel file, or outputs an error message if it cannot open the file, and then saves the Pokémon name and stats as “Pokémon” and adds this to the vector “pokemons”. A temporary vector is also created to make sure the original order of the vector is never lost. The code then outputs the original order, and then runs the “pokemons” vector through each sorting method, sorting each by the stats of each Pokémon, and outputs them in their sorted order and the number of comparisons done during the method. Each of the sorting methods were originally made for arrays but I had ChatGPT take the code I wrote and change any of the parts that used arrays, to use vectors instead to make the program run easier. It did this perfectly as well I did not have to change anything because the code was already written I just needed it to rewrite a few parts. ChatGPT was also very useful when it came to making these sorting methods sort just by the stats of the Pokémon, I asked it to help sort each method by the stats that were saved from each Pokémon, and it knew exactly where to implement that in each of the sorting methods by putting “.stats” where it was required to sort by the stats. For the issue where there are ties in the stats, the program just leaves whichever came first in the original order of the data set. This chart below shows the number of comparisons for each method performed on each data set. |  |  |  |
| A screenshot of a computer  Description automatically generated |  |  |  |

Insertion Sorts best case is O(n) which is when the list is already sorted, and its worst-case runtime complexity is O(n^2) when the elements are in reverse order. Merge Sort has a consistent best- and worst-case scenario time complexity of O(nlogn). Quick Sort has a best-case time complexity of O(nlogn) when it can divide the elements into two equal parts and has balanced partitions at each step and a worst-case time complexity of O(n^2) when the elements are already sorted or nearly sorted.

The best- and worst-case runtime complexity for each sorting method lines up correctly with what we know for each method. As insertion had a really good runtime and number of comparisons when the elements were already sorted, and a really bad runtime and number of comparisons when they were reverse sorted. Merge Sort had a runtime that stayed the same and had a similar number of comparisons throughout the data sets. While Quick Sort handled the random data sets really well and the sorted data sets very poorly, while the reverse sorted sets it performed better on than the sorted sets, but worse than the random data sets.