BLG 335E - ANALYSIS OF ALGORITHMS

PROJECT-1 REPORT

Onat Şahin - 150150129

1)

Note-1: For this question, I calculated the runtimes using my own computer. This is because of ssh server's time limit which makes it impossible to calculate some of the runtimes. However, the code is tested on the ssh server and it compiles and runs without any problems. The compilation instructions are standard. Also, explanations about the functions and algorithms are provided in the code files with comments.

Note-2: In the files I downloaded from Ninova, lines ended with the Windows type newline character ("\n\r"). Therefore, I've read the input files and wrote the output files according to this character. The file reading portion of my code may not properly work without this character.

Note-3: Since the runtimes of insertion sort with 1M data set are too long, I used approximate values which are 100 times the runtime values of 100k set.

Note-4: This runtimes are the results of standard compiling command "g++ main.cpp cardmanager.cpp". It is possible to get shorter runtime results using "g++ main.cpp cardmanager.cpp -O2" command which optimizes the program.

FULL SORT TABLE

Algorithm / Set	10K	100K	1M
Insertion Sort	3.8897 sec	348.184 sec	~34818.4 sec
Merge Sort	0.067626 sec	1.03522 sec	13.9603 sec

FILTER SORT TABLE

Algorithm / Set	10K	100K	1M
Insertion Sort	1.70056 sec	167.751 sec	~16775.1 sec
Merge Sort	0.029517 sec	0.358041 sec	4.29822 sec

2)

When looked at the results of these 3 sets alone, we see that merge sort always outperforms the insertion sort. Also, the difference in runtime between merge sort and insertion sort increases as amount of elements to be sorted increases. This is because of the complexities of these two algorithms. The complexity of insertion sort is O(N^2) while the complexity of merge sort is O(NlogN). If the number of elements in the set becomes 10 times its original value, the amount of operations insertion sort does becomes 100 times the old amount, while the amount merge sort does becomes 10log10 times the old value. However, if the sorting is done on a very small set (like 10 elements), insertion sort performs better, because in small sets iteration is faster than recursion, which becomes a more dominant factor than the difference between complexities.

3)

Rarity and set parameters are similar to type but different from name in terms of the amount. While the number of different types, sets and rarities are small and close the each other, the number of different cards are very big. This could make a difference in the speed of merge sort because since there are less types, rarities and sets, the merge function makes more comparisons where the two strings are the same when sorting by type, rarity and set. This causes sorting by type, set and rarity to take more time than sorting by name.

Since the numbers different of types, sets and rarities are close to each other, I do not think using set or rarity in filter sort instead of type would make much of a difference in terms of runtime.

4)

If a sorting algorithm does not change the input order of elements with equal keys in the output, then this sorting algorithm is a stable sorting algorithm. Insertion sort and merge sort are stable algorithms. In my full sort algorithms, I used a fullCompare() function which alone compared elements according to class, cost and name in the right order. However, there was an other way: We could first call the sorting function to sort only by class, then call to sort only by cost, and finally call to sort only by name, which would give the same right order. This is because insertion sort and merge sort are stable algorithms. If used an unstable sorting algorithm, this approach would not work.