Computability Report

# Introduction

<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf>

# Homework Exercise

## The Program

I chose to write all three sorting algorithms (Bubble, Insertion, and Merge) and compare them. The application uses an extremely simple CLI and was written in plain C using the Visual Studio 2022 IDE. The program can run in two modes depending on the input provided:

* If the word gen and no other input is provided the program will generate 20 files of increasing size up to the maximum containing just over 1 million random signed 64-bit values encoded as ASCII as to be readable.
* If a filename is given, the program will load this file and try to interpret it as a list of randomly arranged, signed, 64-bit integers delimited by newline characters ignoring any other characters. The program will then make 3 copies of this list and use one algorithm to sort each of these lists respectively, it will then make a second pass of the algorithms on the already sorted lists, and then make a final pass re-sorting the lists in reverse order. Finally, the program will display statistics gathered from these operations (Figure 1 shows a typical output).

A screenshot of a computer

Description automatically generated

## Results

The program was run on all 20 generated data files 3 times with an average time in seconds taken from these 3 runs and recorded in a spreadsheet as can be seen in Table 1. Additionally, several graphs were created showing:

* a comparison between the average case for each algorithm (Figure 2).
* a comparison of the different passes for each algorithm respectively (Figures 3, 4, and 5).

A screenshot of a spreadsheet

Description automatically generated

## Analysis

As we can clearly see from Figure 2 bubble sort is the slowest algorithm with insertion sort being in the middle and merge sort being the fastest. Bubble sort is the slowest algorithm in all cases but by looking at the spreadsheet we can see that insertion sort beats merge sort on small datasets up to 64 integers in size, most likely due to the overhead of merge sort.

### Bubble sort

The time complexity of all cases is as expected with the random order and reverse order scaling quadratically, while the sorted order scales linearly. What is interesting however is the reverse order being consistently faster than the average (random order) case. It is likely that this is because the branch predictor is incorrectly guessing the direction of our branches less often when compared to a random order and thus reducing the number of pipeline stalls within the CPU.

### Insertion sort

The time complexity of all cases is as expected with the random order and reverse order scaling quadratically, while the sorted order scales linearly.

### Merge sort

The time complexity is linear in all cases as expected. While all cases are expected to take the same amount of time random order is the slowest and this is most likely due to the branch prediction of the CPU again. The difference between reverse order and sorted order is likely an implementation detail and could be due to reverse order staying within the inner loop more often and causing fewer scope changes.

## Conclusion

The time complexities of all algorithms were largely as expected with the differences being largely attributable to hardware optimizations. This is quite likely a lot more noticeable in a language like C where we are close to the hardware as compared to higher level languages where significant work is done between the instructions requested by the programmer.

# Extension Exercise