Project 3 Report

Team Name: Project 3 61

Team Members: Gerard Quinn (other two members dropped the course at the last minute)

GitHub URL: https://github.com/gquey/Project-3

Link to Video: n/a

Problem: This project aims to analyze the time complexity of sorting different data set sizes using Quick sort and Counting sort.

Motivation: Having sorted data is an important attribute and indicator of how functions will perform when accessing, analyzing, and manipulating data. Having sorted data before it is used will lead to faster data access and analysis and will, in turn, lead to a lower computational complexity for any function that utilizes it.

Features Implemented: An analysis of the speed by which Quick sort and Counting sort organizes data.

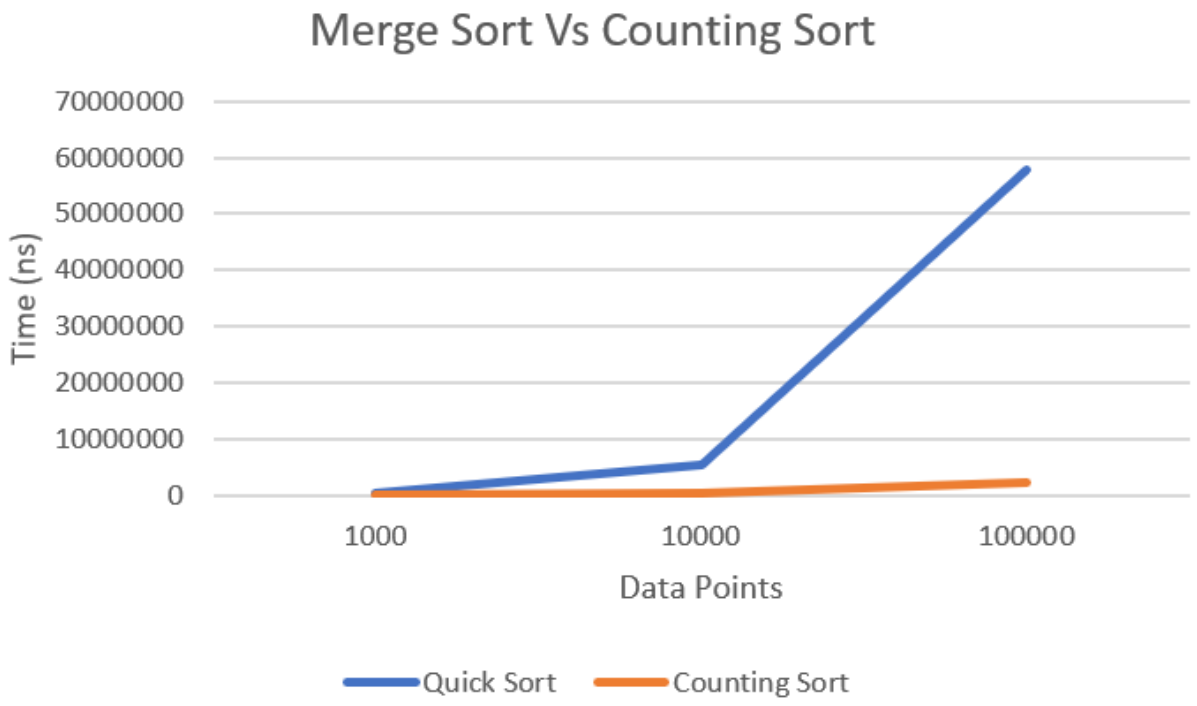
Description of Data: The data that is to be sorted is generated randomly using the rand() function.

Data Structures/Algorithms implemented: An array of randomly generated data was used to store the data that was to be sorted. This array is then passed into the Quick sort and Counting sort functions to sort the data from smallest to largest.

Output

- Quick Sort sorting time for 1000 data points: 460200
- Counting Sort sorting time for 1000 data points: 247600
- Quick Sort sorting time for 10000 data points: 5544800
- Counting Sort sorting time for 10000 data points: 570400
- Quick Sort sorting time for 100000 data points: 57872100
- Counting Sort sorting time for 100000 data points: 2284600

Analysis



- Theoretical average case complexity:
  - Quick Sort – O(n log n) where n is the number of data points to be sorted.
  - Counting Sort – O(n + k) where n is the number of data points to be sorted and k is the range of the values.
- Theoretical worst case complexity:
  - Quick Sort – O(n^2) where n is the number of data points to be sorted.
  - Counting Sort – O(n + k) where n is the number of data points to be sorted and k is the range of the values.
- As seen from the program output and the graph, Counting sort is vastly more efficient at sorting data sets.
- Quick sort works very well on small data sets. However, in comparison to Counting sort, it struggles to maintain that same efficiency when the data size is increased.
- Evidently, the main advantage of Counting sort is its linear complexity. However, the complexity of the algorithm is dependent on the size of the range of values – something that is independent of data size and in the world of data analysis, has a likely chance of being significant and effecting performance.

Reflection

Overall, the project was an interesting experience as it presented me with some of the difficulties that can be experienced when working with other people. Initially, one of my group members decided that he was going to pursue a solo project at the last minute - in the meeting that we were finalizing our project proposal. Furthermore, as the deadline for our submission was approaching, my remaining group member was conspicuously quiet about the progress he was making on his portion of the work. It was not until later that I learned that he decided to drop the course.

If I were to start this project again, I would create a function that calculates the time for the sorting to take place. There was a lot of repetition in my code that could have been avoided this way.

Through this project, I learned about the importance of clear and consistent communication with one's group members. In the future, when working with other people, I am going to maintain clearer and more concise lines of communication so that unexpected such as those I just described can be avoided.