Project Title: Efficient Sorting of Athlete Performance Data

Team Name: Team 116

Team Members: Jai Patel, Alexander Razo, Jonathan Li

Github Names: jaipatel34, razo-alexander, ljonathan0719

Github Link: https://github.com/razo-alexander/Project-3.git

Video Link: https://youtu.be/bEODQz8_Mjs

Proposal

Problem:

When it comes to sports analytics, accurate and efficient sorting of performance data is necessary for awarding championships, tracking competition performance, and evaluating athlete's development. Modern sports datasets often contain detailed records of names, sports, events, and various performance metrics all of which must be analyzed and ranked in order to determine winners, track trends, and evaluate the overall performance of the various athletes.

Motivation:

The ability to process larger datasets quickly and accurately is crucial, especially for specific events like marathons, triathlons, or the Olympics, where a vast number of athletes compete across many events. Efficient sorting is vital for post-event analysis, real-time results, and ranking during these larger, more involved sporting events, where using poor algorithms can cause delayed rankings and hinder real-time analysis because of the scale of modern sports datasets. We wish to ensure these athletes can get the recognition and credit they deserve.

Features Implemented and Description of Data:

Our main features are our merge and quick sort implementations in C++, we also take measurements of the execution time of both algorithms to compare the two and the amount of time it takes for the two to produce an accurate result. We also have an interactive graphical interface built with SFML that allows users to input specific sports and events, view rankings, and compare the sorting results. The dataset used is a simulated CSV file containing an Olympic Games dataset that includes the following fields: ID, Name, Sport, Event, Performance Time, and Country.

Tools Used and Algorithms Implemented:

We made our project in C++ using CLion as our IDE. We used custom algorithms for our merge and quick sort implementations, as well as SFML for our UI visualization. We made use of a custom CSV parser for reading, processing, and handling input data, along with the Chrono library for the timing of our algorithm execution. The Olympic dataset is stored as a vector of structs which helps contain the fields the CSV file contains, lastly, we had a function that extracted specific athletes based on user input for sports and events.

Distribution of Responsibility and Roles:

Alex Razo: Focus on implementing Merge Sort and handling dataset input.

Jai Patel: Implement Quick Sort and test performance comparison.

Jonathan Li: Design the command-line interface and handle data visualization.

Analysis

We did not make any major changes to our project after the proposal as our plan was mostly set in stone. The only change made was made to our UI using SFML where the resulting athlete's name is printed with some "pizzazz".

mergeSort - O(nlogn) where n is the number of elements in the input vector. Merge sort algorithm divides the array into two halves recursively, meaning log n levels of recursion, and then merges them back together in O(n) time at each level. Therefore, the total complexity is O(nlogn).

quickSort - $O(n^2)$ where n is the number of elements in the input vector. The pivot chosen is always the smallest or largest element in the worst case, resulting in largely unbalanced partitions. This means that there will be n levels of recursion, each requiring O(n) comparisons, which is why the worst case complexity is $O(n^2)$.

partition - O(n) where n is the size of the subarray being partitioned. Used in quickSort, partitioning scans the array once, comparing each element with the pivot. This takes a linear amount of time.

GetAthleteCSVData - O(n). This function reads each line of the CSV file once, parsing and storing data for each athlete, resulting in linear complexity relative to the numbers of lines in the file.

merge - O(n) where n is the total number of elements being merged. The function iterates through all elements of the two subarrays, copying them into the merged array. Each element is compared and written into the result once, meaning that it has linear time complexity for merging.

checkEandS - O(n) where n is the total number of athletes. This function iterates through all the athletes to check if their event and sport match the user input.

Reflection

As a group, the overall experience of working on this project was challenging but rewarding. We gained valuable insights and coding experience from this project, especially working as a team. Integrating sorting algorithms and an interactive UI using SFML allowed us to practice our implementation of the concepts we learned in class. It was rewarding to see our project progress from concept to a fully functional program.

We encountered several challenges during the project like data handling, algorithm implementation, SFML integration, and team coordination. Parsing and managing the athlete dataset required us to do a lot of debugging in order to ensure the data was properly being extracted and processed. We had a small hiccup when it came to algorithm implementation as we had to make sure that both merge sort and quick sort work correctly in general and when handling edge cases like duplicate performance times. Creating a graphical interface took a while because we had to refresh on how to integrate SFML, especially when it came to managing user inputs and producing outputs that had some sort of visual appeal.

If we were to start this project over, there would be a few changes made to the project and workflow. We would definitely start this project earlier. We really only started making progress on this project during Thanksgiving break, something we all did not find enjoyable because we wanted to enjoy our holiday. If we spent more time working on this project beforehand, we could have lessened the workload and maybe even gotten the project done earlier. Also, planning out the UI more would be nice as it would make the program more intuitive and visually appealing. Overall, there are only a few things that we would change if we restarted the project, but they are key components that would've affected our project greatly.

There are various things that each member learned throughout this project development process. All of us learned how to improve our communication and teamwork skills, contributing to debugging and overall coordination. This improved our skills in problem solving and team management. Jonathan focused on SFML integration and further developed his skills in creating user interfaces. He also learned to handle user inputs effectively. Alex and Jai learned the importance of data parsing and how to implement merge sort and quick sort. They also gained a better understanding of how to handle large datasets and edge cases. Jai worked on time measurement and performance analysis, giving him a better understanding on how to profile algorithms and interpret their run time behavior depending on the scenario.

References

"Tutorials for SFML 2.6." *2.6 Tutorials (SFML / Learn)*, www.sfml-dev.org/tutorials/2.6/. Accessed 25 Nov. 2024.

"Quick Sort." *GeeksforGeeks*, 16 Nov. 2024, www.geeksforgeeks.org/quick-sort-algorithm/. Accessed 23 Nov. 2024

"Merge Sort - Data Structure and Algorithms Tutorials." *GeeksforGeeks*, 16 Nov. 2024, www.geeksforgeeks.org/merge-sort/.