Jarrod Helmers

CS-300

Prof. Webb

17 March, 2024

The aim of this code is to showcase the contrasting efficiency of two sorting algorithms - selection sort and quicksort - when applied to a dataset of bids that have been sourced from a CSV file. The development process involved comprehending the specified requirements, incorporating the sorting algorithms into the existing program structure, and ensuring their proper invocation. A challenge that arose during this process was guaranteeing the accurate collection of timing results, which required a thorough review of the provided code and a detailed study of the sorting algorithms. Through trial and error, I was able to test various approaches until reaching the desired functionality.

Pseudocode:

* Read bids from the CSV file.
* Implement the selection sort algorithm on the bid titles.
* Measure and report the time taken for sorting.
* Implement the quicksort algorithm on the bid titles.
* Measure and report the time taken for sorting.
* Display the sorted bids.

Code Quality Assessment:

The source code has implemented both selection sort and quicksort algorithms on the bid titles, meeting its specifications. Although certain sections of the code have "FIXME" comments indicating incomplete areas, it executes without errors and aligns with most of the provided specifications. However, further testing and refinement may be necessary to ensure complete correctness and compliance with all specifications.

Code Annotation and Documentation:

The code has sufficient comments that explain each section's purpose and approach. Comments clarify the sorting algorithms' implementation, file input/output operations, and timing measurements. Although some parts lack comments, the annotations overall promote code readability and facilitate understanding for other programmers.

Modularity and Reusability:

The code demonstrates modularity and reusability by isolating distinct functionalities into separate methods and classes. However, there is an opportunity to improve modularity by organizing code into more cohesive and reusable components. For example, the selection sort and quicksort algorithms could be encapsulated within their respective classes, promoting better code organization and maintainability.

Readability:

The code emphasizes readability by using consistent whitespace usage, indentation, and clear variable naming conventions. However, some sections could benefit from additional comments or refactoring to enhance readability further. Overall, the code's readability supports ease of comprehension and future code maintenance.