Carmen Mosquera

CS 499 – 11429-M01

10 /16 /2024

Narrative : Artifact Two

Artifact Description

Originally, this artifact was a basic C++ application that read and printed data from a text file using a binary tree. The data contained in the text file was a list of university courses, each with a course number, course name, and a sub list of prerequisites. The core purpose of the application was to demonstrate the implementation of basic data structures and algorithms in a static, text-based environment.

After enhancements, the artifact has evolved into a robust Java-based algorithm testing application with a dynamic, interactive interface using JavaFX. The updated version allows users to test and visualize multiple algorithms such as Binary Search, Quick Sort, Insertion Sort, and others in real-time, providing a clear understanding of their performance and behavior with different data structures.

I created the original version in 2022 for the CS 300 (Data structure and algorithms) course and developed the enhanced version this year for my CS 499 computer science capstone course.

Inclusion Justification

I chose to include this artifact in my ePortfolio because I want to improve and showcase my understanding of both fundamental computer science concepts and my ability to implement them in a meaningful, interactive application. This artifact will help me demonstrates my skills in algorithms and data structures, which are important for software engineering roles. In particular, the following components highlight my strengths:

- Algorithm Design and Implementation: The artifact features a variety of sorting and searching algorithms, both recursive and non-recursive. It shows my ability to not only understand these algorithms but also to implement them in an efficient, scalable way.
- User Interface and Real-Time Visualization: I implemented a JavaFX UI that provides real-time feedback to the user by visually representing the algorithm's execution. The animations, which represent sorting algorithms via moving and color- changing rectangles, demonstrate my ability to connect algorithmic concepts with concrete user interactions.
- Performance Analysis: I integrated metrics such as runtime and memory usage to enhance the educational value of the application.

Performed Enhancements List

1. Ported the program from C++ to Java
2. Created a graphical user interface using JavaFX

3. Implemented multiple sorting algorithms such as Insertion sort, quicksort, selection sort, and merge sort.
4. Integrated recursive and non-recursive binary search algorithms.
5. Added a new TreeMap search algorithm to showcase diverse searching techniques.
6. Developed real-time visualization features using JavaFX for sorting and searching algorithms.
7. Represented elements as graphical rectangles that dynamically change color while being manipulated by the algorithms
8. Added functionality to measure the performance of algorithms in terms of runtime and memory usage.
9. Displayed performance metrics for each algorithm, giving users insights into their time complexity and space efficiency.
10. Introduced interactive features allowing users to test different algorithms on various data structures such as arrays, linked lists, and binary trees.
11. Added a text area next to each algorithm where users can observe and read explanations about the runtime and memory usage
12. Provided detailed descriptions of how each algorithm works and how its performance is calculated, giving users a deeper understanding of algorithmic principles.
13. Refactored and modularized the code into distinct classes for each algorithm, data structure, and utility function.

Course Outcomes

The development and enhancement of the algorithm testing application helped me reach the following course outcomes:

Course Outcome 2: I designed, developed, and delivered professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts by completing the following enhancements:

- **real-time algorithm visualization**. I implemented visual aids where each element of a data structure is represented by a rectangle. The rectangles dynamically change colors to visually communicate the search process: starting with black, turning blue when visited, and highlighted in green when the target element is found. For sorting algorithms, I introduced animations that show the movement of each element as they are sorted, with rectangles shifting from right to left and turning blue once sorted. In the **QuickSort algorithm**, I highlighted the pivot element in red on each iteration. This visual representation significantly enhances the user experience by clearly demonstrating how each algorithm operates.

- Below each algorithm, I added **Vertical Boxes** to display execution results such as runtime and memory usage, along with clear explanations of the algorithm's behavior and time complexity. This visual and written communication ensures the application is technically sound and easy for users to understand.

- I included extensive comments throughout the code to ensure that collaborators and future developers could easily navigate, understand, and extend the artifact

Course Outcome 3: I designed and evaluated computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices by completing the following enhancements:

- Ported From C++ to Java: Porting this artifact from C++ to Java with JavaFX involved important design choices and adaptations. I faced the challenge of translating and restructuring the code to fit Java's syntax and object-oriented structure while ensuring the application maintained its original functionality. This transition not only improved the user interface with JavaFX but also enhanced the performance of the algorithms.

- I implemented multiple sorting and searching algorithms to test them across different data structures, allowing for a versatile execution environment. For example, sorting algorithms were applied to course objects alphabetically by course name, while searching algorithms targeted course objects by course number.

- To compare performance, I added both recursive and non-recursive searching algorithms, allowing users to test and evaluate their efficiency in real time. This enhancement demonstrates my ability to design computing solutions that address specific problems while managing trade-offs in performance and complexity.

Course Outcome 4: I used well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals by completing the following enhancements:

- I refactored and modularized the code into distinct classes, separating the algorithms from the data structures, following industry-standard practices. This modular design makes the application scalable, maintainable, and easier to debug.

- I created a **GlobalUI class** that implements the main components of the user interface, such as algorithm boxes, buttons, rectangles, containers, and labels. This class is reusable and adaptable, meaning future developers can easily modify the UI by working within this class without affecting the core functionality of the program. By centralizing UI management in the **GlobalUI class**, the artifact becomes more flexible and resilient to changes, enabling easy updates or redesigns to the interface. This modular approach, along with the integration of performance metrics (runtime and memory usage), ensures that the application delivers tangible value while adhering to best practices in software development. It also provides a solid foundation for further enhancements, such as integrating more advanced algorithms or refining the visual experience.