Holly Renfrew

Due Date: 11/23/25

CS 499 – Milestone 3, Enhancement 2

Algorithms and Data Structure

Artifact Description

The artifact I selected for this milestone is the Airgead Banking Investment Calculator, originally created in CS-210. The initial version of the program was a C++ console application that calculated yearly investment growth based on user inputs and displayed results using simple formatted output. The logic performed only annual compound-interest calculations and did not take advantage of modern data structures or algorithmic optimizations. This project served as one of my first exposures to applied financial algorithms and provided an ideal foundation for demonstrating my growth in algorithmic thinking and structured data manipulation.

For the purposes of Milestone Three, I significantly expanded the scope of the artifact. In addition to transforming the underlying algorithms and data structures, I redesigned the system as a full graphical desktop application, complete with:

- A Windows GUI built in C++
- A light/dark theme toggle
- A splash screen
- A custom application icon
- A live-updating results table for all yearly calculations

These additions modernize the usability and presentation of the calculator, reflecting professional-quality software engineering practices alongside the algorithmic enhancements.

Justification for Inclusion in the ePortfolio

I selected this artifact for my ePortfolio because it demonstrates both my foundational skills and my advanced abilities in algorithms, data structures, and software engineering.

The project evolved from a simple console tool into a structured, optimized, and user-friendly investment-calculation system.

The following enhancements showcase my abilities in the Algorithms and Data Structures category:

1. Monthly Compound-Interest Algorithm (vs. annual-only)

   I redesigned the core financial computation algorithm to run monthly compounding loops with yearly aggregation. This introduced:

   - Nested iteration
   - Local accumulation structures
   - Increased algorithmic complexity
   - More accurate real-world results

   This redesign demonstrates algorithmic refinement and performance-aware implementation.

2. Use of a Dynamic Data Structure (std::vector<YearRecord>)

   Instead of printing output directly to the console, I store each year's results in a vector of custom structs. Enhancements include:

   - Using vector::reserve() to proactively allocate capacity (Big-O efficiency improvement)
   - Encapsulating yearly data in the YearRecord struct
   - Passing structured results into the GUI table

   This highlights my ability to design and use appropriate data structures for scalable data handling.

3. Sorting Algorithm (O(n log n))

   Yearly results are sorted using:

   ```
   std::sort(results.begin(), results.end(), comparator);
   ```

   Applying sorting demonstrates my understanding of STL algorithm usage, comparator design, and time-complexity trade-offs.

4. Binary Search on Year Values

I implemented a binary search function:

```
std::binary_search(sorted.begin(), sorted.end(), key, comparator);
```

This required transforming the data into a sorted-by-year sequence and demonstrates my ability to:

- Prepare data for search operations
- Use logarithmic-time search algorithms
- Justify data-structure and efficiency decisions

5. Stack and Queue Logging (LIFO & FIFO)

I introduced two additional data structures:

- queue<double> transactionHistory (FIFO)
- stack<double> reverseHistory (LIFO)

These structure monthly deposit history in two traversal patterns, demonstrating:

- Practical memory-structure choice
- Understanding of different retrieval orders
- Ability to augment an algorithm with meaningful auxiliary data structures

6. GUI Integration of Structured Data

The GUI displays all calculated results from vector<YearRecord>.
This demonstrates:

- Separation of concerns
- Layered architecture
- Passing structured algorithmic output into UI components

7. Complete User Interface Upgrade

Although not part of the algorithmic classification alone, the GUI upgrade demonstrates polished, industry-aligned software practices:

- Light mode ↔ Dark mode toggle

- Modern input controls
- Splash screen on startup
- Custom .ico branding
- Dynamic table output bound to vector results

This shows my ability to take algorithmically complex code and make it accessible to end users, which is required by the program outcomes.

Alignment with Course Outcomes

This enhancement directly aligns with the following Computer Science program outcomes:

Outcome 3 — Algorithms and Data Structures

"Design and evaluate 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."

My enhancements demonstrate this through:

- A redesigned monthly compounding algorithm
- Decisions about using vectors vs. linked structures
- Sorting (O(n log n)) and binary searching (O(log n))
- Reserving memory to improve vector performance
- Designing custom data-handling structures to fit program needs

Outcome 4 — Software Engineering / Design

"Design and evaluate 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."

My enhancements demonstrate this through:

- A full UI redesign in C++ rather than console printing
- A theme system with mode toggling

- A splash screen and user-friendly interface
- Integration of structured algorithmic output into GUI controls
- Clean modularity between data, algorithms, and UI layers

Outcome 2 — Professional Communication

"Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts."

This is demonstrated through:

- A clear, sortable table
- Readable labels
- Consistent formatting
- The addition of a professional-looking interface

This shows that I can adapt technical communication to a non-technical audience.

Outcome 5 — Security Mindset (Minor Application)

"Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources."

While not a security-focused project, the GUI input validation and data-type constraints reflect:

- Safe handling of user-supplied data
- Prevention of invalid numerical states
- Guarding against buffer or type errors

Reflection on the Enhancement Process

Enhancing this artifact was a comprehensive learning experience that reinforced algorithmic reasoning, modern C++ design, and the importance of structuring data effectively. I gained deeper insight into how STL containers operate internally, how algorithmic complexity impacts runtime behavior, and how data structures influence the architecture of a program.

What I Learned

- How to convert an annual-only calculation into a high-resolution monthly algorithm.
- How to measure and improve performance using vector::reserve().
- How sorting and binary searching interact with structured data.
- Practical benefits of FIFO vs. LIFO history structures.
- The challenge of integrating core logic with a Windows GUI. Especially ensuring a clean separation between logic and presentation.

Challenges Faced

- Balancing algorithmic accuracy with GUI responsiveness.
- Ensuring the vector, stack, and queue remained synchronized across monthly loops.
- Troubleshooting incorrect values caused by rounding differences between UI and algorithmic output.
- Making the theme-toggle system update all UI elements consistently.
- Adapting console-based code to event-driven GUI software, which requires different architectural patterns.

Despite these challenges, the final product is significantly more efficient, more maintainable, and far more user-friendly than the original. This enhancement meaningfully demonstrates my mastery of algorithms and data structures and my growth as a software developer.