Nathan Johnson

SNHU

CS-499-13459-M01

4-2 Milestone Three: Enhancement Two: Algorithms and Data Structure

03/30/2025

Enhancement Two: Algorithms and Data Structure

Artifact Description

I chose a contact management system from CS 320. This program, built in Java, handles a database of contacts with core CRUD functionality. It stores personal information like names, phone numbers, and addresses. Users can add new contacts, update existing information, delete contacts, and retrieve specific records by ID. The original implementation used an ArrayList-based storage system that worked adequately for small datasets but would struggle with larger collections of contacts. The program successfully managed contact information, but its underlying data structure limited its performance potential.

Justify the Inclusion of the Artifact

The contact system worked, but performance was an issue. The ArrayList implementation meant every operation required scanning through the entire list, making searches slow as contact numbers grew. Updating or deleting contacts required finding the right index first, which wasn't efficient. The code worked fine with small numbers, but would struggle with larger datasets. That made it perfect for improvement. I saw a chance to take something functional but inefficient and transform it into something that could handle real-world scenarios with hundreds or thousands of contacts.

Improvements and Enhancements

I improved the system by replacing the ArrayList with a HashMap for contact storage. Using contact IDs as keys dramatically improved lookup speed. This change reduced search, update, and delete operations from O(n) time complexity to O(1), making the system much faster. While an ArrayList needs to check every contact until it finds the right one, a HashMap jumps straight to the correct record.

Through optimizing the update function, I made the system more efficient by only changing what's different. The old code replaced everything every time. Now it only updates what actually changed. This saves time and keeps things running smooth, especially when just changing one small detail. The Contact class already had validation in the constructor, but I improved how the service layer works with it to maintain data integrity throughout the system's operation.

I put in better error checks too. The system tests contact IDs and other fields before accepting changes. When something's wrong, it says exactly what the problem is instead of just failing. Users get real help fixing issues instead of guessing what went wrong. The whole system runs better now and doesn't break easily when bad data comes in.

Course Outcomes

I fixed this artifact and did everything I planned to do. I wanted to make the system faster by using better data structures and smarter algorithms. I got rid of the slow searching and put in direct lookups instead, which made things much quicker. The better error checking makes the system more stable and less likely to crash. I did all these things just like I intended to do.

I also made the system talk to users better. Error messages now actually tell you what's wrong and how to fix it. The system behaves more consistently and doesn't surprise users with weird responses. The improved code with its smart data choices shows strong algorithmic thinking and problem-solving skills that computer science is all about.

Data structures matter in algorithm design. A HashMap gives immediate access without searching the entire collection. This approach isn't just theoretical, it's practical. Our system can now handle larger contact databases without slowing down. The code also adapts to new

requirements more easily because of its more modular design and improved error handling system.

Reflection on Process

Performance matters in software development. When contacts are stored in a HashMap, operations stay high-speed even as the database grows. Searches don't get slower with more contacts, and updates happen immediately because the system knows exactly where to look. Looking back, I should have considered even more specialized data structures like a TreeMap for sorted output, though the HashMap provides the best balance of features for most use cases. Throughout the optimization, I deliberately eliminated inefficient operations and replaced them with direct lookups. The code just runs faster now, it's much more responsive, makes better use of resources, and you can add thousands of contacts without it slowing down.

Time complexity and space requirements don't always align. HashMaps need more memory than arrays, but they save tremendous time. Finding this balance was the main challenge. Testing with different dataset sizes confirmed the performance improvements. Functions needed to produce the same results, just with better efficiency. Over time, slow operations became instant lookups. The program handles larger datasets now and responds much faster. This project taught me how important data structure selection is, and how important a skill it is for any software engineer working with data-intensive applications.