CS-499-11684-M01

Professor Fitzroy Nembhard

Fernanda Coghlan

29 May 2025

# CS 499 Module Four

# Milestone Three:

# Enhancement Two: Algorithms and Data Structure

**PART I**

**Artifact Narrative**

# Structure

This narrative documents the second enhancement for my CS 499 ePortfolio project, focusing on algorithms and data structure. The objective of this enhancement is to showcase my ability to design and implement an optimized algorithmic solution using data structure principles while improving a real-world Android application originally created in CS 360.

# Artifact Description

The artifact selected for this enhancement is the FLC Weight Tracker Android mobile application, originally developed in Spring 2024 as a final project for CS 360: Mobile Architecture and Programming. The application was built using Java, Android Studio. It enables users to register, log in, record daily weight entries, set a weight goal, and track their progress over time through a user-friendly interface.

Initially, the application featured a straightforward Activity-to-Activity architecture and used in-memory data storage, limiting its usefulness for long-term tracking or data analysis. While the original version fulfilled basic functionality requirements, it lacked any analytical or algorithmic components capable of deriving insights from user data. Specifically, there were no mechanisms to identify patterns, trends, or meaningful summaries based on weight entry history.

This enhancement targeted that gap by incorporating an algorithmic data structure approach to analyze weight trends weekly. By processing user entries, grouping them by week, and calculating average weekly weights, the application now provides a dynamic summary of whether a user has gained or lost weight over time. This implementation leverages sorting algorithms, map-based grouping, and average calculations across a TreeMap data structure, significantly improving the artifact’s complexity, usefulness, and alignment with real-world computing needs.

# Justification for Inclusion

This artifact was selected for enhancement under the Algorithms and Data Structure category due to its initial simplicity and absence of data analysis features. The original application allowed users to input and view weight data but lacked any logic to extract insights or identify patterns over time. To address this gap, I implemented a custom trend analysis algorithm that processes user weight entries to display weekly weight change trends, providing a more meaningful and engaging user experience.

The enhancement involved several algorithmic and structural improvements. I used a TreeMap to group weight entries by ISO calendar week while maintaining chronological order. The grouped entries were then processed to compute average weekly weights. Finally, the application compares the earliest and most recent weekly averages to determine whether the user has gained or lost weight, displaying this insight via a reactive UI component. This required a strong understanding of date manipulation, sorting, grouping, and numeric averaging across dynamic data sets.

The solution also uses LiveData to ensure that trend calculations are updated in real-time as new weight entries are added or removed, further reinforcing my knowledge of reactive programming and MVVM architecture in Android. By combining these algorithmic techniques with appropriate data structures, TreeMap for weekly grouping and ArrayList for averaging, the enhancement clearly showcases my ability to solve practical problems through structured, efficient, and maintainable code.

# Course Outcomes and Updates

#### Course Outcome Alignment

This enhancement aligns primarily with Course Outcome #3: “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 (data structures and algorithms).”

By implementing a custom algorithm that groups weight entries by calendar week, computes weekly averages, and analyzes trends using TreeMap and List data structures, I applied sound algorithmic principles to a real-world scenario. These enhancements reflect my ability to structure logic for effective computation, data organization, and user feedback.

Additionally, this enhancement contributes to Course Outcome #2 by presenting algorithmic results in a clear and informative way through real-time, user-facing UI components. This ensures that technical solutions are effectively communicated to users in a meaningful and accessible format.

#### Planned Outcomes

This enhancement fulfills the planned learning objectives from Module One. My goal was to move beyond simple data display and implement a meaningful analytical feature using custom logic and standard data structures. Hopefully I can met that goal by grouping weight entries by week, calculating trends across time, and delivering insights through reactive UI updates.

The artifact was improved in response to instructor feedback that the original enhancement plan (a basic weekly average) was too trivial. The revised enhancement incorporates complexity by organizing the data into weekly groupings and analyzing changes across time periods. The use of lifecycle-aware LiveData, structured computation, and best-practice coding conventions supports this improvement.

# Reflection on Enhancement Process

Enhancing the application provided hands-on experience in applying algorithmic thinking to real-world problems involving user-entered data. I learned how to design logic that extracts meaningful insights, such as weekly weight trends, from a flat, unsorted list of records. This required transforming raw input into structured groupings, calculating weekly averages, and analyzing the difference across time to communicate progress to the user.

A key learning moment was working with Java modern date and time API, particularly LocalDate and WeekFields, which enabled accurate calendar-based grouping of weight entries. I chose to use a TreeMap to maintain chronological order of grouped weeks and processed weekly averages using List<Float>. This ensured clarity, precision, and scalability in the algorithm's logic.

The most significant challenge was designing a trend detection algorithm that was both meaningful and efficient. Initial instructor feedback pushed me to go beyond a trivial average calculation and instead build a solution that included weekly grouping and analysis over time. I faced issues ensuring that dates were sorted correctly, and weeks were consistently calculated regardless of locale or gaps in user data.

Testing edge cases, such as sparse or inconsistent entries, required multiple iterations to refine the algorithm and improve reliability. I also had to ensure that LiveData updates were bound safely to the UI lifecycle, allowing real-time feedback without performance trade-offs or crashes.

This process strengthened my confidence in decomposing abstract algorithmic requirements into structured, maintainable, and user-friendly code. It also deepened my understanding of balancing data structure choices (TreeMap vs HashMap), efficiency, and user experience.

# Evidence of Enhancement

The following section includes code snippets, screenshots of the trend output on-screen, and logs demonstrating the grouping, averaging, and final calculation logic. This evidence illustrates the before-and-after impact of the enhancement.

**Screenshot of working app with weekly trend**

**A screenshot of a phone

AI-generated content may be incorrect.**

**Added Weekly Weight Trend to show gained or lost weight over the weeks.** Original app had no trend analysis.

**A screenshot of a phone

AI-generated content may be incorrect.**

**Added instruction to user on how to delete a data, and multiples weight data can be added in the same day.** Original app was unable to delete a data or hold multiple data in the same date.

**A screenshot of a weight loss program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**WeightViewModel.java**

// CS-499-11684-M01  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 29 May 2025  
  
package com.example.cs499trackerapp;  
  
import android.app.Application;  
import android.util.Log;  
  
import androidx.annotation.NonNull;  
import androidx.lifecycle.AndroidViewModel;  
import androidx.lifecycle.LifecycleOwner;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
  
import java.time.LocalDate;  
import java.time.ZoneId;  
import java.time.temporal.WeekFields;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
import java.util.Locale;  
import java.util.Map;  
import java.util.TreeMap;  
  
*/\*\*  
 \* ViewModel for managing weight entries and computing weekly trend.  
 \* Separates UI logic from data and handles lifecycle-aware data.  
 \*/*public class WeightViewModel extends AndroidViewModel {  
  
 private static final String *TAG* = "WeightViewModel";  
  
 // Repository pattern to access Room database  
 private final WeightRepository repository;  
  
 // LiveData list of all weight entries for the user  
 private final LiveData<List<WeightEntry>> allEntries;  
  
 // Computed trend message (e.g., "You gained 5 lbs over 3 weeks")  
 private final MutableLiveData<String> weeklyTrend = new MutableLiveData<>();  
  
 */\*\*  
 \* Constructor initializes repository and retrieves weight entries for a user.  
 \*/* public WeightViewModel(@NonNull Application application, String username) {  
 super(application);  
  
 Log.*d*(*TAG*, "Initializing repository with user: " + username);  
 repository = new WeightRepository(application);  
 allEntries = repository.getEntriesForUser(username);  
  
 Log.*d*(*TAG*, "Observing entries initialized.");  
 }  
  
 */\*\*  
 \* Observes the LiveData entries and calculates the weekly trend based on weight averages.  
 \* @param owner The LifecycleOwner (typically an Activity or Fragment)  
 \*/* public void bindTrendToOwner(LifecycleOwner owner) {  
 allEntries.observe(owner, entries -> {  
 if (entries != null && entries.size() >= 2) {  
 Log.*d*(*TAG*, "Weekly trend: processing " + entries.size() + " entries");  
  
 // Sort entries chronologically by date  
 List<WeightEntry> sorted = new ArrayList<>(entries);  
 sorted.sort(Comparator.*comparing*(WeightEntry::getDate));  
  
 // Group weights by week number using a TreeMap to preserve week order  
 Map<Integer, List<Float>> weeklyGroups = new TreeMap<>();  
 for (WeightEntry entry : sorted) {  
 LocalDate localDate = entry.getDate().toInstant().atZone(ZoneId.*systemDefault*()).toLocalDate();  
 int weekOfYear = localDate.get(WeekFields.*of*(Locale.*getDefault*()).weekOfWeekBasedYear());  
  
 // Create week list if missing, then add weight  
 weeklyGroups.putIfAbsent(weekOfYear, new ArrayList<>());  
 weeklyGroups.get(weekOfYear).add(entry.getWeight());  
 }  
  
 // Calculate weekly average weights  
 List<Float> weeklyAverages = new ArrayList<>();  
 for (List<Float> weekWeights : weeklyGroups.values()) {  
 float sum = 0;  
 for (float w : weekWeights) {  
 sum += w;  
 }  
 weeklyAverages.add(sum / weekWeights.size());  
 }  
  
 // Calculate weight change from oldest to latest week  
 float oldest = weeklyAverages.get(0);  
 float latest = weeklyAverages.get(weeklyAverages.size() - 1);  
 float change = latest - oldest;  
  
 // Build message for display  
 String result = "You " + (change < 0 ? "lost " : "gained ") +  
 Math.*abs*(change) + " lbs over " + weeklyAverages.size() + " weeks";  
  
 Log.*d*(*TAG*, "Trend result: " + result);  
 weeklyTrend.setValue(result);  
 } else {  
 // Not enough data to compute trend  
 weeklyTrend.setValue("Not enough data yet.");  
 }  
 });  
 }  
  
 */\*\*  
 \* Returns all weight entries for the user as LiveData.  
 \*/* public LiveData<List<WeightEntry>> getAllEntries() {  
 return allEntries;  
 }  
  
 */\*\*  
 \* Returns computed weekly trend message as LiveData.  
 \*/* public LiveData<String> getWeeklyTrend() {  
 return weeklyTrend;  
 }  
  
 */\*\*  
 \* Inserts a new weight entry into the database.  
 \*/* public void insertEntry(WeightEntry entry) {  
 repository.insert(entry);  
 }  
  
 */\*\*  
 \* Deletes a weight entry from the database.  
 \*/* public void deleteEntry(WeightEntry entry) {  
 repository.delete(entry);  
 }  
}

**WeightTrackerActivity.java**

// CS-499-11684-M01  
// Professor Fitzroy Nembhard  
// Fernanda Coghlan  
// 29 May 2025  
  
package com.example.cs499trackerapp;  
  
import android.app.AlertDialog;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.util.Log;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.TableLayout;  
import android.widget.TableRow;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.lifecycle.ViewModelProvider;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.List;  
import java.util.Locale;  
  
// Activity responsible for managing user interactions for weight tracking  
public class WeightTrackerActivity extends AppCompatActivity {  
  
 private static final String *TAG* = "WeightTrackerActivity";  
  
 // UI components  
 private TableLayout weightTable;  
 private TextView weightGoalValue;  
 private TextView weeklyTrendValue;  
 private WeightViewModel viewModel;  
 private String currentUser;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_weight\_tracker*);  
  
 Log.*d*(*TAG*, "onCreate started");  
  
 // Step 1: Retrieve username from Intent  
 currentUser = getIntent().getStringExtra("username");  
 Log.*d*(*TAG*, "Got from intent: " + currentUser);  
  
 // Step 2: Fallback to SharedPreferences if intent is null  
 if (currentUser == null || currentUser.isEmpty()) {  
 SharedPreferences prefs = getSharedPreferences("UserPrefs", Context.*MODE\_PRIVATE*);  
 currentUser = prefs.getString("loggedInUser", null);  
 Log.*d*(*TAG*, "Fallback to prefs: " + currentUser);  
 }  
  
 // Step 3: Redirect to login if username is still missing  
 if (currentUser == null || currentUser.isEmpty()) {  
 Log.*e*(*TAG*, "No username found. Redirecting.");  
 Toast.*makeText*(this, "No user found. Returning to login.", Toast.*LENGTH\_SHORT*).show();  
 startActivity(new Intent(this, LoginActivity.class));  
 finish();  
 return;  
 }  
  
 // Step 4: Initialize UI components  
 weightTable = findViewById(R.id.*weight\_table*);  
 weightGoalValue = findViewById(R.id.*weight\_goal\_value*);  
 weeklyTrendValue = findViewById(R.id.*weekly\_trend\_value*);  
 EditText inputDate = findViewById(R.id.*input\_date*);  
 EditText inputWeight = findViewById(R.id.*input\_weight*);  
 Button addWeightButton = findViewById(R.id.*add\_weight\_button*);  
 Button setGoalButton = findViewById(R.id.*set\_goal\_button*);  
  
 // Step 5: Initialize ViewModel with current user context  
 try {  
 WeightViewModelFactory factory = new WeightViewModelFactory(getApplication(), currentUser);  
 viewModel = new ViewModelProvider(this, factory).get(WeightViewModel.class);  
 Log.*d*(*TAG*, "ViewModel created for user: " + currentUser);  
  
 // Bind LiveData observer for computed trend logic  
 viewModel.bindTrendToOwner(this);  
 } catch (Exception e) {  
 Log.*e*(*TAG*, "ViewModel creation failed", e);  
 Toast.*makeText*(this, "Unable to load weight data.", Toast.*LENGTH\_LONG*).show();  
 finish();  
 return;  
 }  
  
 // Step 6: Observe LiveData for weight entries and trend updates  
 viewModel.getAllEntries().observe(this, this::updateTable);  
 viewModel.getWeeklyTrend().observe(this, trend -> weeklyTrendValue.setText(trend));  
  
 // Step 7: Add weight entry logic  
 addWeightButton.setOnClickListener(v -> {  
 String dateStr = inputDate.getText().toString().trim();  
 String weightStr = inputWeight.getText().toString().trim();  
  
 if (dateStr.isEmpty() || weightStr.isEmpty()) {  
 Toast.*makeText*(this, "Please fill both fields.", Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 try {  
 float weight = Float.*parseFloat*(weightStr);  
 SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy", Locale.*US*);  
 Date date = sdf.parse(dateStr);  
  
 if (date != null) {  
 // Step 7.1: Create a new weight entry  
 WeightEntry entry = new WeightEntry(weight, date, currentUser);  
 viewModel.insertEntry(entry);  
  
 // Step 7.2: Optionally check if goal is reached  
 String goalStr = weightGoalValue.getText().toString().replace(" lbs", "").trim();  
 if (!goalStr.isEmpty()) {  
 try {  
 float goal = Float.*parseFloat*(goalStr);  
 if (Float.*compare*(goal, weight) == 0) {  
 Toast.*makeText*(this, "🎉 You reached your goal!", Toast.*LENGTH\_LONG*).show();  
 }  
 } catch (NumberFormatException ignored) {  
 }  
 }  
  
 // Step 7.3: Reset input fields  
 inputDate.setText("");  
 inputWeight.setText("");  
 } else {  
 Toast.*makeText*(this, "Invalid date format.", Toast.*LENGTH\_SHORT*).show();  
 }  
  
 } catch (Exception e) {  
 Toast.*makeText*(this, "Error adding weight.", Toast.*LENGTH\_SHORT*).show();  
 Log.*e*(*TAG*, "Weight entry error", e);  
 }  
 });  
  
 // Step 8: Set weight goal logic  
 setGoalButton.setOnClickListener(v -> {  
 final EditText input = new EditText(this);  
 input.setHint("Enter weight goal (lbs)");  
  
 new AlertDialog.Builder(this)  
 .setTitle("Set Weight Goal")  
 .setView(input)  
 .setPositiveButton("Set", (dialog, which) -> {  
 String goalStr = input.getText().toString().trim();  
 if (!goalStr.isEmpty()) {  
 try {  
 float goal = Float.*parseFloat*(goalStr);  
 weightGoalValue.setText(String.*format*(Locale.*US*, "%.1f lbs", goal));  
 } catch (NumberFormatException e) {  
 Toast.*makeText*(this, "Invalid weight goal entered.", Toast.*LENGTH\_SHORT*).show();  
 }  
 } else {  
 Toast.*makeText*(this, "Weight goal cannot be empty.", Toast.*LENGTH\_SHORT*).show();  
 }  
 })  
 .setNegativeButton("Cancel", null)  
 .show();  
 });  
 }  
  
 */\*\*  
 \* Updates the weight entries table dynamically.  
 \* Clears existing rows (except headers) and repopulates with new data.  
 \*  
 \* @param entries List of weight entries to display  
 \*/* private void updateTable(List<WeightEntry> entries) {  
 int rowCount = weightTable.getChildCount();  
 if (rowCount > 1) {  
 weightTable.removeViews(1, rowCount - 1);  
 }  
  
 SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy", Locale.*US*);  
  
 for (WeightEntry entry : entries) {  
 TableRow row = new TableRow(this);  
 TextView dateView = new TextView(this);  
 TextView weightView = new TextView(this);  
  
 // Format and set values  
 dateView.setText(sdf.format(entry.getDate()));  
 weightView.setText(String.*format*(Locale.*US*, "%.1f", entry.getWeight()));  
  
 // Padding for better readability  
 dateView.setPadding(8, 8, 8, 8);  
 weightView.setPadding(8, 8, 8, 8);  
  
 row.addView(dateView);  
 row.addView(weightView);  
  
 // Long-press to delete an entry  
 row.setOnLongClickListener(v -> {  
 new AlertDialog.Builder(this)  
 .setTitle("Delete Entry")  
 .setMessage("Do you want to delete this entry?")  
 .setPositiveButton("Delete", (dialog, which) -> {  
 viewModel.deleteEntry(entry);  
 Toast.*makeText*(this, "Entry deleted", Toast.*LENGTH\_SHORT*).show();  
 })  
 .setNegativeButton("Cancel", null)  
 .show();  
 return true;  
 });  
  
 weightTable.addView(row);  
 }  
 }  
}

**PART II**

**Artifact Submission**

Technical artifact files were zipped and uploaded and I used the check list below to assure the enhancement proposals for Algorithms and Data Structure were implemented:

* 1. **Algorithmic Processing Implemented**

A weekly trend detection algorithm was added, calculating average weight per week from user entries.

* 1. **Data Grouping Strategy**

Weight entries are sorted by date and grouped by calendar weekusing java.time and TreeMap for order.

* 1. **Use of Data Structures**

Used List, Map, and TreeMap to store, group, and process weight data by week.

* 1. **Date-Based Calculations**

Used LocalDate, ZoneId, and WeekFields to normalize and group dates by ISO week number.

* 1. **Trend Output Calculation**

Algorithm computes weight difference between the first and last weekly averages to determine if the user lost or gained weight.

* 1. **Reactive UI Feedback**

Calculated trend is posted to a LiveData<String> observable and displayed automatically in the app.

* 1. **Lifecycle Awareness**

bindTrendToOwner() method ensures that observers are attached in a lifecycle-aware way using the Activity context.

* 1. **Instructor Feedback Addressed**

Enhancement was revised to move beyond trivial averaging, adding meaningful week-based temporal analysis.

* 1. **Input Validation Maintained**

The app ensures sufficient data is available before computing the trend; edge cases are handled gracefully.

* 1. **UI Behavior Improved**

A clear message is displayed to the user (e.g., "You gained/lost X lbs over Y weeks") based on computed data.