Final Project Report

Group 62
Team Name: AmanTech
Team Members: Alen Zacharia, Jared Mindlin, Robert Medina
Project Title: FoodSearch
Github: https://github.com/da-b0ss/P3-v2
Video: https://youtu.be/diCSyVFskd0

Project Proposal

Problem:

Many people struggle with meal planning and ensuring they get the necessary nutrients, especially those who are dieting, trying to bulk up, or losing weight. This can be particularly challenging for people living in food deserts or who lack the resources to access nutritious foods. We are trying to address this issue by providing an easy and convenient way for people to find foods based on their desired nutritional content in order to avoid any nutrient deficiencies.

Motivation:

Ensuring proper nutrition is critical for maintaining good health and well-being. Poor nutrition can lead to a range of health problems, including obesity, diabetes, and heart disease. Meal planning and food selection can be time-consuming and difficult for many people, leading to unhealthy eating habits and poor nutritional choices. Not everyone is aware of the nutrient contents of what they eat every day. This project will allow for efficient access to the USDA's food and nutrition data.

Features:

Our program, FoodSearch, will provide users with easy access to various recommended foods based on their desired nutritional content. Users should be able to search for foods using a scrollbar based on a range of criteria, including protein, carbohydrates, fats, and more. The foods highest in such desired nutrients would then be displayed first. The project is also able to display the speeds of both sorting algorithms.

Data:

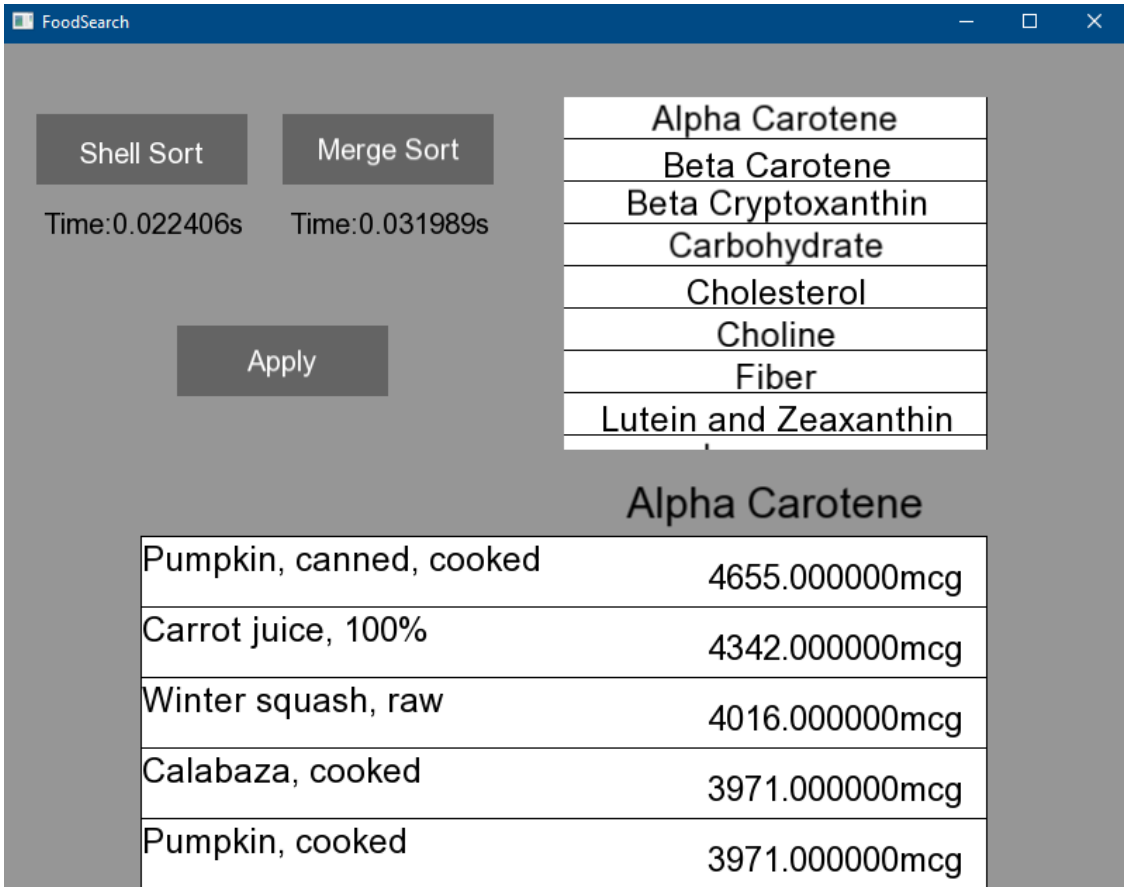
We will use the United States Department of Agriculture's Food Composition Database, sourced from the U.S. Department of Agriculture's 2019/20 data. This dataset is a comma-separated value (.csv) file that contains detailed information about the nutritional content of a wide range of foods, including macronutrients, micronutrients, and other relevant information. For each food item(row), there are 38 columns corresponding to different data points, 35 of those columns hold nutritional content, and there are 7079 rows of data (the top row holding titles and the rest of the 7078 rows being different foods).

Tools/Languages/APIs/Libraries:

- C++
    - SFML library used https://www.sfml-dev.org/documentation/2.5.1/

- Git

Visuals:



Algorithms implemented / Additional Data Structures/Algorithms used:

We implemented a dynamically allocated memory array to store the food and its nutrition content data. Each array index contained a food object that held strings with names and categories and an array of doubles to store the nutrient contents. We also used a merge-sort algorithm and a shell-sort algorithm to find foods high in specific nutrient content that the user selected by sorting the foods by nutrients in descending order. Then, these two sorting algorithms were compared by having timers that run for as long as the sorting algorithm is sorting. For example, the user selects vitamin A from a scrolling selection menu and then chooses between using a merge-sort or shell-sort algorithm to find foods that are highest in vitamin A, then clicks apply and this data is sorted and displayed as well as the time it took to sort.

Distribution of Responsibility and Roles:

Visualization/GUI: Robert, Jared, Alen
Data processing, containers, and storage: Jared
Merge-Sort algorithm: Robert
Shell Sort: Alen

Analysis

While designing the project architecture, we discovered that utilizing an array of ‘Food’ objects for storing data would be better than using a singly linked list. This implementation provided significant benefits in terms of access time complexity and improved compatibility with merge and shell sorting algorithms. Shell sort in particular would be incredibly difficult to implement with a singly linked list data structure. We implemented a ‘Food’ object which stores the category, description, and id number as member variables and the nutrient contents in an array where each index corresponds to a specific nutrient.

We decided against including a search function for nutrients due to how unlikely a user would be to remember all these long, technical names. Instead, we were able to accomplish the same functionality with a scrollbar. With regards to the design, we shifted the display of the foods and nutrients to the bottom of the window as food names in the dataset were longer than originally expected. With the new layout, we are able to more effectively utilize the same amount of space.

In our final design, we added a timer underneath the sorting algorithm buttons to help compare the speeds of merge sort and shell sort. This helps represent the purpose and differences between the two different sorting algorithms. From our testing, it appears as though merge sort and shell sort have roughly the same time complexities. However, they differ in two ways. Shell sort has less overhead, as there are no recursive function calls, so it is slightly faster. Also, merge sort performs roughly the same regardless of the order of the data, whereas shell sort has a bit more variability.

Time Complexities:

split - A helper function in main.cpp that assists in parsing data in the while loop that reads and stores data in our foodList with a time complexity of O(n) for n lines of data.
merge - A function in FoodList.cpp that assists mergeSort by merging two subarrays of size n with a time complexity of O(n).
mergeSort - A stable recursive merge sort algorithm in FoodList.cpp that sorts an array by a given nutrient index with a time complexity of O(nlog(n)) for the food array of size n. Calls the merge function to combine the separated, sorted arrays and this causes the log(n), since mergeSort function also uses merge, merge contributes to the n that is being multiplied to logn in O(n*logn).
shellSortFoodList - A shell sort algorithm in FoodList.cpp that sorts an array by a given nutrient index with a time complexity of O(nlog(n)) for the food array of size n.
timeStart - A function in FoodList.cpp that starts a timer with a time complexity of O(1).
timeStopAndDisplay - A function in FoodList.cpp that stops and displays a time with a time complexity of O(1).
run - A function in GUI.cpp that runs the other major GUI functions. This means it will have the time complexity of the worst function, which is update(). Thus, it has a complexity of O(n log(n)).
eventHandler - A function in GUI.cpp that handles input from user interactions. It only sets flags to perform actions later, so it only has a time complexity of O(1).
render - A function in GUI.cpp that renders all SFML objects of the program. This is fixed in the program so it has a time complexity of O(1).

update - A function in GUI.cpp that updates textboxes and calls the sorting algorithms. It therefore has the same complexity as the sorting algorithms, which is O(n log(n)), for the n foods.
updateFoods - A function in GUI.cpp that only prints the first five food items in the sorted food list. This has no dependence on the total number of foods so it has a time complexity of O(1).

Reflection

Completing the project was an overall highly positive and rewarding experience despite the difficult time constraints of impending finals and exams. We were able to apply many of the concepts and practices learned in class, previous knowledge from other classes, and new knowledge gained from working as a team.

In the beginning, we had a few challenges understanding how to use GitHub desktop and the git framework correctly. Pushing, fetching, and updating the correct branches with the right files was a little difficult; A couple of mishaps due to non-standardized formatting of code among teammates resulted in us needing to create a new repository. Though after getting the hang of it, it greatly improved our workflow and efficiency. There were also challenges in learning and implementing SFML to create a graphic interface. The library offers a broad range of functionalities that were difficult to navigate, and it required some refreshing of content from Programming 2 and significant experimentation and trial and error to get the desired results.

If we were to start again as a group it would be great to refine and better modularize our program and functions and to make the GUI look nicer. The only major change in the project from our initial plan was switching from using a linked list to a dynamically allocated memory array. The SFML visualizations took a large portion of the workload, it would have been helpful to properly plan and distribute that portion of the project now that we have a better understanding of the library. It also might have helped to use a library dedicated to making GUIs like Qt. SFML is versatile, but we had to implement some rather basic functionality from the ground up, like a scrollbar and clickable text boxes. Using a library with these features already done would have meant we could spend more time working on aesthetics and adding additional features.

Personal Reflections:

Comment on what each of the members learned through this process.

Alen: In doing this project, I became significantly better at using SFML and understanding how the library is structured. I also gained first-hand experience in seeing the benefits of how splitting the project into multiple files helps when working in a team. Initially, when we were putting most of our code in main, there was a lot of conflicts between changes we would make. After making our code more modular, it became more clear which functions each person should be working on and we had far less conflicts.

Jared: I do not have a large amount of experience using git with a team, so this project greatly helped my understanding of the framework. Using SFML in a project again was a little challenging but enjoyable despite the difficult time constraints of other coursework. Reflecting on our process helps us recognize areas we can improve in the future and has me looking forward to applying this to projects I work on in the future.

Robert: I had barely used git in the past, however, after this project my confidence in using github as drastically increased, utilizing github desktop after a bit of trial and error, I am now able to proficiently solve merging issues and code in a manner that avoids many other merging issues. I understand the importance of code format standardization when working in teams now. The teamwork experience was also invaluable, I have never worked with a team to program something with this level of complexity using github. Being able to efficiently manage a team and allocate tasks was an experience I enjoyed more than I thought I would. I have usually disliked teamwork but this experience was positive. Finally, it was also satisfying to use material from previous classes to complete this assignment.

References

“Documentation of SFML 2.5.1.” *Main Page (SFML / Learn / 2.5.1 Documentation)*, https://www.sfml-dev.org/documentation/2.5.1/.

“Food and Nutrition.” USDA, https://www.usda.gov/topics/food-and-nutrition.

Whitcomb, Ryan, et al. “Food CSV File , .” *CORGIS Datasets Project*, https://think.cs.vt.edu/corgis/csv/food/.