Reflection on Assignment 1: Data Structures

Challenge 1: The Array Artifact

What I Learned:

This challenge really helped me appreciate the simplicity and power of arrays. I now understand how fixed-size memory allocation can sometimes be a limitation, but the speed and ease of accessing data by index is a major benefit. Implementing both linear and binary search showed me how much faster binary search can be, but only when the data is sorted.

Challenges and How I Overcame Them:

One of the tricky parts was keeping the array sorted for binary search, especially after removing an artifact. At first, my solution broke the sort order, but I fixed this by shifting elements when removing items to maintain the array's sorted state.

Ideas for Improvement:

A dynamic array could be a nice upgrade, allowing the array to grow when needed. I could also add a more efficient sorting algorithm to keep everything in order automatically.

Challenge 2: The Linked List Labyrinth

What I Learned:

This challenge made me realize how useful linked lists are when the size of your data isn't predictable. I found the loop detection problem especially interesting—it showed me why knowing how to detect cycles is so important in certain types of data structures.

Challenges and How I Overcame Them:

Initially, my approach to detecting loops was too simple and missed edge cases. After researching, I learned about Floyd's cycle-finding algorithm (also called the "tortoise and hare" algorithm), which made solving the problem much easier and more efficient.

Ideas for Improvement:

Adding features like a doubly linked list would allow us to traverse the list in both directions, which could be helpful for more complex operations. This would give the data structure more flexibility.

Challenge 3: The Stack of Ancient Texts

What I Learned:

Working with stacks reinforced the importance of the LIFO (Last In, First Out) principle. Stacks are great for situations where you need to reverse actions or handle temporary data. I enjoyed building this simple stack, as it's such a fundamental data structure used in a lot of different areas.

Challenges and How I Overcame Them:

Searching for a specific scroll in the stack was more difficult than I expected, since stacks don't allow random access like arrays do. I eventually created a recursive search function to scan the elements one by one, though it's not the most efficient method.

Ideas for Improvement:

I could implement an undo/redo system using two stacks, which would add practical functionality. Also, switching the stack's implementation from an array to a linked list could make it more dynamic and scalable.

Challenge 4: The Queue of Explorers

What I Learned:

Queues, especially circular ones, taught me how efficient they can be in managing tasks that need to be handled in a specific order. Circular queues, in particular, optimize space usage by reusing empty slots, something I hadn't fully appreciated before this challenge.

Challenges and How I Overcame Them:

The toughest part was figuring out how to manage the front and rear pointers in the circular queue. I ran into some bugs where the queue appeared full even though there was space, but after reviewing the logic, I was able to get the wrap-around behavior working correctly.

Ideas for Improvement:

A priority queue would be a great next step. It would allow explorers to be queued based on their importance or urgency, adding a layer of functionality that could be useful in more complex scenarios.

Challenge 5: The Binary Tree of Clues

What I Learned:

I found binary trees fascinating because they make storing and retrieving data so efficient. Learning the different traversal methods—like in-order, pre-order, and post-order—gave me a better understanding of how these trees can be used in different ways, depending on the task at hand.

Challenges and How I Overcame Them:

Getting the traversal algorithms right was a bit challenging, especially because it required careful management of recursive function calls. After a few missteps, I was able to correctly implement each traversal and get a better feel for how the recursion flows.

Ideas for Improvement:

I'm thinking about extending this with a self-balancing tree like an AVL or a red-black tree, which would keep the tree balanced and improve search times. I could also add deletion functionality to make it a more complete solution.