1. The Array Artifact

In this task, I built the ArtifactVault class, which stores ancient artifacts using an array. It includes features to add artifacts to the first empty spot, remove artifacts by name, and search for artifacts either using linear search or binary search (assuming the array is sorted by age). One key challenge was ensuring the array remained sorted for binary search to work efficiently, and I had to handle null entries carefully to prevent issues during searches.

What I Learned: This exercise deepened my understanding of how to manage arrays in a more practical way. The biggest challenge was making sure the array remained organized for binary search while also dealing with null values. It reinforced the importance of keeping arrays tidy and in the right order for optimal search results. A future improvement could be using a dynamic array that adjusts in size as artifacts are added, removing the restriction of a fixed-size array.

2. The Linked List Labyrinth

For this challenge, I designed the LabyrinthPath class using a singly linked list to represent a path through a labyrinth. The class allows adding new locations, removing the most recently visited location, and checking for loops or traps using a well-known algorithm called Floyd's Cycle Detection (or the tortoise and hare method). It also has a feature to print the entire path.

What I Learned: This task helped me see the benefits of linked lists, especially when it comes to working with data that grows dynamically. The loop detection using Floyd's algorithm was interesting to implement and provided an efficient way to check for traps. An idea for improvement could be adding functionality to find the size of the loop if a trap is detected or switching to a doubly linked list for better navigation through the labyrinth.

3. The Stack of Ancient Texts

I created the ScrollStack class to manage scrolls using a stack, where the last scroll added is the first to be removed (Last-In, First-Out). The class allows adding, removing, and viewing scrolls, as well as searching for specific scrolls by title. The stack grows dynamically to accommodate more scrolls as needed.

What I Learned: This exercise reinforced my understanding of how stacks work and how useful they can be for managing ordered collections of data. One challenge was implementing the search functionality, which required iterating through the entire stack. A future improvement could be adding a caching system to make searches faster, especially for frequently accessed scrolls.

4. The Queue of Explorers

In this task, I implemented the ExplorerQueue class using a circular queue to manage explorers waiting to enter a temple. Circular queues efficiently use space by wrapping around

when they reach the end of the array. The class includes methods to add explorers, remove them from the front, view the next explorer in line, and check if the queue is full or empty.

What I Learned: This task gave me valuable insights into the benefits of circular queues, particularly when it comes to saving space and improving efficiency. Managing the wrap-around behavior of the queue required careful index management. A possible future enhancement would be adding a priority system where higher-priority explorers could enter the queue ahead of others.

5. The Binary Tree of Clues

For this problem, I created the ClueTree class, which organizes clues in a binary search tree (BST). The class allows inserting clues, performing different types of traversals (in-order, pre-order, and post-order), searching for specific clues, and counting the total number of clues in the tree.

What I Learned: This task was more complex due to the recursive nature of binary trees. I learned a lot about how binary search trees allow for efficient insertion and searching, but also how important it is to keep the tree balanced to avoid worst-case performance. An improvement to the solution would be to implement a self-balancing tree, like an AVL or Red-Black Tree, to ensure that operations remain efficient even as the tree grows.