REFLECTION PDF

1. Artifact Vault

What I Learned:
From the Artifact Vault challenge, I learned how to manage a collection of objects with dynamic behaviours like adding, removing, and searching artifacts. The challenge helped me understand the importance of implementing both linear and binary search algorithms to efficiently find items. Binary search, in particular, taught me about sorting data and the efficiency it brings, reducing the search time from linear to logarithmic complexity.

Difficulties Encountered & Solutions:
The main difficulty I faced was ensuring that the binary search worked correctly after sorting the artifacts. Initially, I did not handle the sorting step properly, which led to incorrect results. I resolved this by sorting the artifacts by age before performing the binary search. I also had to manage the vault’s capacity, which required me to handle edge cases like when the vault is full.

Ideas for Improvement:
In future versions of the Artifact Vault, I could implement a dynamic resizing mechanism to avoid hitting the vault's capacity. I could also optimize the search algorithms by allowing the user to search based on multiple criteria (e.g., name, age, or other attributes).

---

2. Clue Tree

What I Learned:
The Clue Tree challenge gave me valuable insight into binary search trees (BSTs). I learned how BSTs maintain sorted order and how they allow efficient insertions, deletions, and searches. The three types of traversals—pre-order, in-order, and post-order—taught me how to access the tree’s data in different ways, depending on the use case.

Difficulties Encountered & Solutions:
One challenge I encountered was correctly implementing the recursive traversal methods, especially for post-order. I initially struggled with the order in which the nodes should be accessed. I overcame this by closely following the recursive structure and ensuring that the traversal methods called the appropriate sub-methods in the correct order.

Ideas for Improvement:
A potential improvement for the Clue Tree would be to implement balancing techniques like AVL or Red-Black trees to keep the tree height low, ensuring that search operations remain efficient even with a large number of clues.

---

3. Scroll Stack

What I Learned:
This challenge deepened my understanding of stack data structures, specifically the Last-In-First-Out (LIFO) principle. I learned how stacks can be used to manage elements in such a way that the most recently added element is the first to be removed. I also gained hands-on experience with handling dynamic stack resizing.

Difficulties Encountered & Solutions:
One difficulty was ensuring that the stack could grow dynamically to accommodate more elements than its initial size. Initially, the stack would throw an error when exceeding its capacity. I resolved this by implementing dynamic resizing, where the stack doubles in size whenever it reaches capacity.

Ideas for Improvement:
A more efficient solution could involve implementing a stack with a fixed, but optimal, capacity to avoid the overhead of resizing. Additionally, I could introduce a maximum size for the stack to avoid infinite growth.

---

4. Explorer Queue

What I Learned:
Through this challenge, I learned the fundamentals of queue data structures and their use in managing a collection of elements in a First-In-First-Out (FIFO) order. Implementing a circular queue helped me optimize the memory usage, as it ensures that the queue always uses the available space efficiently.

Difficulties Encountered & Solutions:
Initially, I faced issues with correctly implementing the circular queue behavior. I struggled with the logic to correctly wrap around when the queue's rear index reached the end of the array. This was fixed by using modulo arithmetic to wrap around the indices properly.

Ideas for Improvement:
One improvement would be to allow the queue to resize dynamically if it reaches full capacity. Additionally, I could introduce priority queues to manage the explorers based on specific attributes, such as urgency or importance.

---

5. Labyrinth Path

What I Learned:
The Labyrinth Path challenge taught me how to approach maze-solving problems using search algorithms. I implemented a depth-first search (DFS) to explore paths, which helped me understand how to systematically traverse a grid and backtrack when necessary. This challenge also reinforced the idea of recursion in solving problems.

Difficulties Encountered & Solutions:
The main difficulty was ensuring that the DFS algorithm correctly handled backtracking. Initially, I did not keep track of the visited cells properly, leading to infinite loops. I solved this by maintaining a separate visited array to track cells that had already been explored.

Ideas for Improvement:
For more complex mazes, I could improve the solution by implementing more efficient pathfinding algorithms like breadth-first search (BFS) or A*, which guarantee the shortest path in most cases. I could also extend the solution to handle 3D mazes or different types of obstacles.

Conclusion

In completing these challenges, I gained practical experience in implementing and understanding core data structures like stacks, queues, trees, and arrays. Each challenge introduced me to a new data structure and the corresponding algorithms required to manipulate and interact with the data efficiently. These hands-on exercises helped solidify my understanding of both the theory and application of data structures and algorithms in real-world scenarios.

The difficulties I encountered throughout these challenges taught me valuable lessons in problem-solving, debugging, and optimizing code. By implementing these structures and algorithms from scratch, I now feel more confident in my ability to use them in future projects and real-world applications.