**Assignment 6 Report: Part 2 - Elementary Data Structures**

**1. Objective**

The goal of this part is to build from scratch and test four fundamental data structures: Array, Stack, Queue, and Singly Linked List, understanding their operational logic and performance implications.

**2. Implementation Summary**

**Files:**

* array.py: Encapsulates array insert, delete, and access operations
* stack.py: Implements LIFO stack
* queue.py: Implements FIFO queue
* linked\_list.py: Implements insert, delete, and traversal
* demo\_data\_structures.py: Demonstrates and tests each structure

Each data structure was designed using Python's object-oriented programming principles with encapsulated logic and clear method naming.

**3. Sample Execution Results**

**Stack**

Top: 20

Top after pop: 10

**Queue**

Front: 1

Front after dequeue: 2

**Linked List**

List: [300, 200, 100]

After deletion: [300, 100]

**Array**

Access index 1: 10

After delete index 0: [10]

**4. Complexity Analysis**

| **Operation** | **Array** | **Stack** | **Queue** | **Linked List** |
| --- | --- | --- | --- | --- |
| Insert (end/head) | O(1) | O(1) | O(1) | O(1) |
| Delete (specific idx) | O(n) | O(1) | O(1) | O(n) |
| Access (by index) | O(1) | O(1) | O(1) | O(n) |
| Traverse | O(n) | O(n) | O(n) | O(n) |

**5. Use Case Scenarios**

| **Data Structure** | **Real-World Applications** |
| --- | --- |
| Array | Lookup tables, buffer management |
| Stack | Function call stacks, undo features |
| Queue | Print jobs, message queuing |
| Linked List | Media players, memory-efficient editing |

**6. Advantages & Limitations**

* **Array**: Best for indexed access; resizing is expensive.
* **Stack/Queue**: Efficient for LIFO/FIFO; limited to linear access.
* **Linked List**: Flexible memory allocation; slower random access.

**7. Conclusion**

The implementation of elementary data structures from scratch provided valuable insights into the fundamental building blocks of computer science. Each data structure exhibits distinct characteristics that make it suitable for specific use cases. Arrays excel in scenarios requiring frequent random access due to their O(1) indexing capability, while linked lists offer superior flexibility for dynamic memory allocation and frequent insertions/deletions at arbitrary positions.

Stacks and queues, despite their operational simplicity, prove indispensable in numerous algorithms and system designs. The LIFO nature of stacks makes them perfect for managing recursive function calls and implementing undo mechanisms, while queues' FIFO behavior is essential for task scheduling and breadth-first traversals.

This hands-on implementation experience reinforced the importance of understanding memory models, access patterns, and performance bottlenecks. The knowledge gained forms a solid foundation for more complex data structures and algorithms, emphasizing that effective software design requires careful consideration of data structure selection based on specific operational requirements and performance constraints.