**Module 4 – Critical Thinking Project**

# Option 1 **-** Design and implement an experiment that will compare the performance of the Python list based stack and queue with the linked list implementation.

# Overview

## Stack:

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle.

Elements are added and removed from the top of the stack.

Common operations include push (addition) and pop (removal).

## Queue:

A queue is a linear data structure that follows the First In, First Out (FIFO) principle.

Elements are added at the rear (enqueue) and removed from the front (dequeue) of the queue.

Common operations include peeking (viewing the front element) and checking if the queue is empty.

# Experiment

Here I implemented the common operations for stacks like push and pop using both list and Linked list

Also implemented enqueue and dequeue using list and linked list

Here are the results

A graph with lines and numbers

Description automatically generated with medium confidence

## Inference

**List-based Stack and Queue Times:** The time taken for operations on the list-based stack and queue implementations remains relatively constant across different input sizes. This is expected because Python lists offer efficient insertion and deletion operations at the beginning and end of the list, making them suitable for implementing both stacks and queues.

**Linked List Stack and Queue Times:** The time taken for operations on the linked list-based stack and queue implementations increases significantly as the input size grows. This increase in time is likely due to the overhead of traversing the linked list structure, especially for larger input sizes. Linked lists involve dynamic memory allocation and pointer manipulation, which can introduce additional overhead compared to the contiguous memory allocation of Python lists.

Performance Comparison: Overall, for smaller input sizes (e.g., 1000), the linked list implementations perform reasonably well and exhibit comparable performance to the list-based implementations. However, as the input size increases (e.g., 10000, 100000), the overhead of linked lists becomes more apparent, resulting in slower performance compared to list-based implementations.

Trade-offs: While linked lists offer flexibility in terms of dynamic memory allocation and efficient insertion/deletion at any position, they may not be the most efficient choice for implementing stacks and queues in scenarios where performance is critical, especially for large data sets. In contrast, list-based implementations provide better performance characteristics for these specific data structures, particularly for smaller to moderate-sized data sets.

## Conclusion

The choice between list-based and linked list-based implementations depends on factors such as expected data size, performance requirements, and the specific use case. For scenarios where performance is crucial, especially for large data sets, list-based implementations may offer better performance characteristics. However, linked lists remain valuable for scenarios requiring dynamic memory allocation or efficient insertion/deletion at arbitrary positions.