**Spatial Complexity Analysis of the pop Method in the Stack Class**

The pop method of the Stack class performs the following operations:

**1. Creation of Local Variables:**

**- popped:** Stores the top element of the stack. This variable occupies a constant amount of memory, regardless of the size of the stack.

- **isLastNode:** A lambda function to identify the last node in the linked list. This function occupies a constant amount of memory, regardless of the size of the stack.

- **result:** Stores the result of removing the last node from the linked list. This variable may contain an error message if the removal operation fails. The length of this error message is constant and does not depend on the size of the stack.

**2. Access to the Linked List:**

- The linked list contains the elements of the stack. The memory occupied by the linked list depends on the size of the stack.

**3. Removal of the Last Node:**

- The operation to remove the last node from the linked list involves freeing the memory occupied by that node. This operation may require constant time O(1) for node removal and memory deallocation.

The spatial complexity of the **pop()** method mainly depends on the size of the stack, as the linked list `list` occupies memory proportional to the size of the stack. Therefore, the spatial complexity of the pop() method as a whole is O(n), where n is the size of the stack.

**Spatial Complexity Analysis of the `enqueue` Method in the Queue class**

The **enqueue** method of the class implements the functionality of adding a new element to the end of the queue. Let's see how the spatial complexity of this method is composed:

**1. Local Variables:**

- The enqueue method does not declare additional local variables apart from the `data` parameter, which represents the element to be added to the queue. The memory used by these variables is constant and does not depend on the size of the queue.

**2. Access to the Linked List:**

- The `enqueue` method calls the `addNode` method to add a new node to the linked list, which is the underlying data structure used to represent the queue. Let's analyze the operations within the `addNode` method:

- **Creation of the new node:** A new node is created to store the data provided as a parameter. The memory used for this node is constant and does not depend on the size of the queue.

- **Updating pointers:** The `next` and `prev` pointers of the previous node and the new node are updated to correctly link the new node to the end of the linked list. Each node involved requires a constant amount of additional memory to store these pointers.

The spatial complexity of the `enqueue` method is primarily determined by the number of nodes in the linked list and the additional memory required for each node. Since the `enqueue` method only adds a new node to the end of the linked list, its spatial complexity is proportional to the size of the queue. Therefore, the spatial complexity of the `enqueue` method is O(n), where n is the size of the queue.