1.1 Analyze the total time required to push n elements to the Stack and express it using the Big O notation

To provide the necessary context to address this question, the **Stack ADT class** implemented in Question 1 is defined by the following characteristics:

- The underlying data structure is a singly headed singly linked list (SHSL)
- The stack must be maintained in Last-In-First-Out (LIFO) order
- The location of the Stack's top begins at the back (Last node of the linked list)

Since we are working with a SHSL list (Single reference to the head node), we must traverse through the entire list of n elements before adding the next node to the Stack. In other words, the time complexity of push() is O(n) as it is dependent on the size of the linked list.

In the scenario of pushing n elements to an arbitrary m sized linked list, there are two scenarios. First suppose our linked list is empty ($m = 0$), then we just need to create a new Node, set it's data value, and make the head point to this Node. Second, suppose our linked list has non-zero elements.

Iterations Required	**Elements in Linked List**	**Elements To Add**
0	m	n
m	m+1	n-1
m+1	m+2	n-2
m+2	m+3	n-3
......	......	......
m+n-1	m+n-1	1
m+n	m+n	0

We need to also create a new Node like the first case, but we need to traverse from the head all the way to the address of the last Node (m >= 1); setting its next pointer to our new Node. Each successive push to the list will require an additional iteration because m has increased by 1 in the previous push operation. This is formally defined in a mathematical definition below:

$$
\begin{aligned}
&= (m+1) + (m+2) + \ldots + (m+n-1) + (m+n) \\
&= m * n + \frac{(1+n)n}{2} \\
&= m * n + \frac{n}{2} + \frac{n^2}{2} \\
&= n + \frac{n}{2} + \boldsymbol{\frac{n^2}{2}}
\end{aligned}
$$

Therefore: Pushing n elements has an overall time complexity of O(n^2)

1.2 Analyze the total time required to pop those n elements from the Stack and express it using the Big O notation

Similarly to the execution of push, the pop() method requires us to traverse through the linked list in order to remove the "top" element. In other words, the time complexity of pop() is

O(n) as it is also dependent on the number of elements in the Stack.

In the scenario of popping our n elements from an arbitrary m sized linked list, we need to traverse from head to the last Node in the linked list.

Iterations	Elements of a linked list	The element to be remove
0	m	n
m	m-1	n-1
m-1	m-2	n-2
m-2	m-3	n-3
......	......	......
2	1	1
1	0	0(this case m = n)

To remove the first element, we need to loop m times to reach the last node from "top" and remove it. To remove the second element, we need to loop $m-1$ times to reach our new "top" element. This pattern continues until the linked list as only one element remaining in the linked list as defined in the mathematical definition below:

$$= (m-1) + (m-2) + (m-3) + ... + (m-n+1) + (m-n-1)$$
$$= m * n - (1 + 2 + 3 + ... + n)$$
$$= n^2 - \frac{(1+n) * n}{2}$$
$$= n^2 - \frac{n}{2} - \boldsymbol{\frac{n^2}{2}}$$

Therefore: Popping n elements has an overall time complexity of O(n^2)