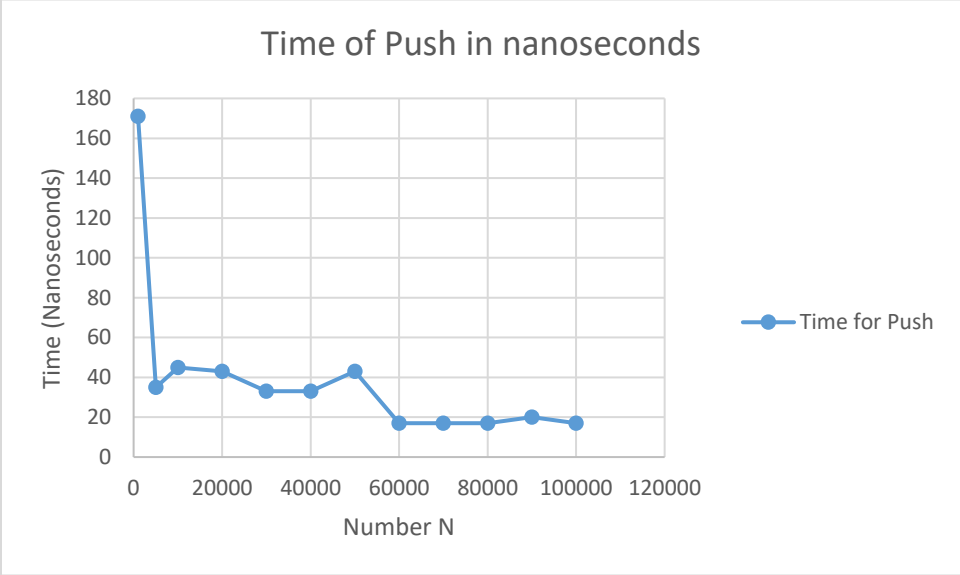
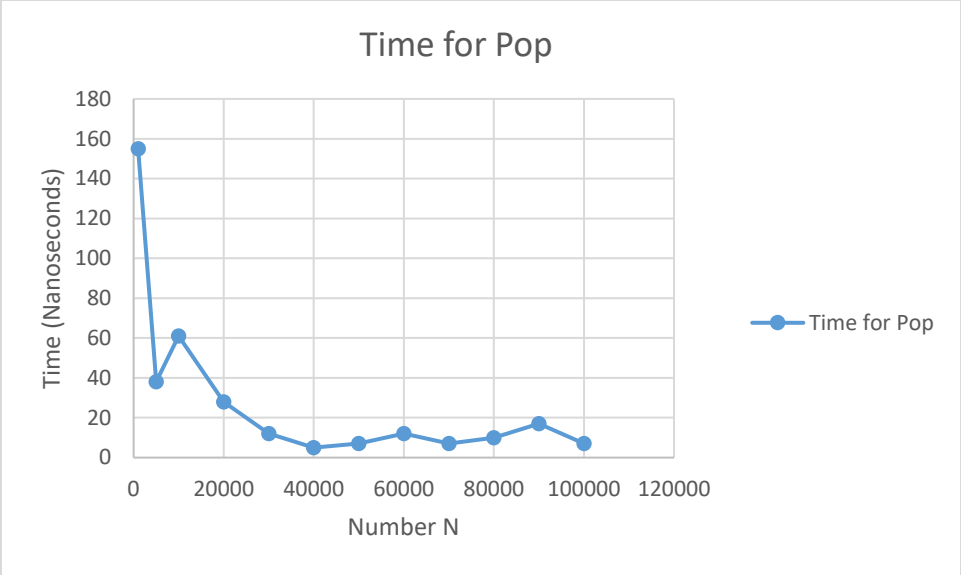
Gabe Brodbeck

U0847035

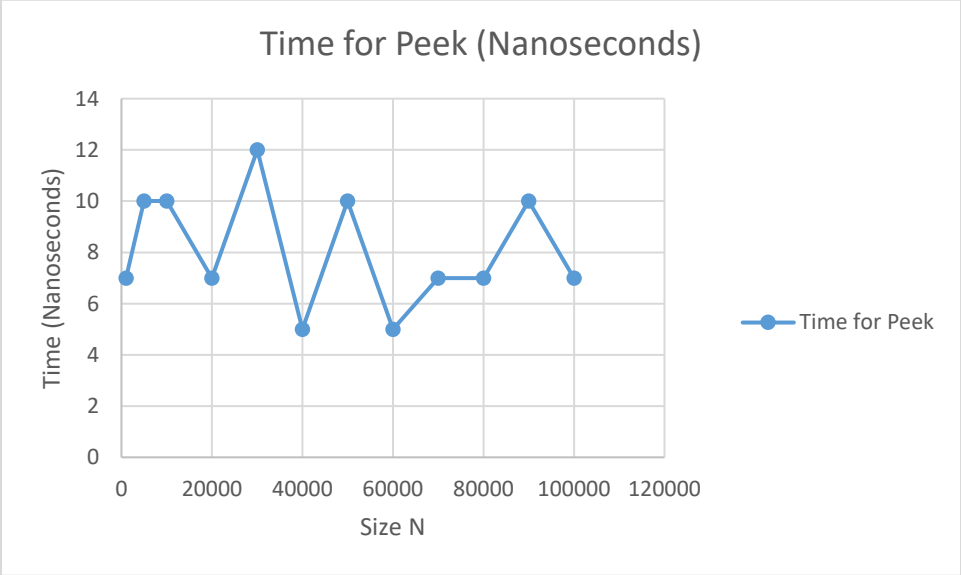
Assignment 07 Analysis

1. Yes, I have already worked with 2 different partners and intend to work with a third.
2. I cannot see any reason why our stack needs to be a doubly linked list. The stack never actually uses a feature which requires any node to move to the previous node, as stacks only act on the first node and the first node never has a previous node. So it is technically advantageous to use a singly linked list instead because it would save a very small number of operations each time a node is pushed or popped.
3. Java's linked list is a doubly linked list with extremely similar features to the list we built in assignment 6. The features of the DoublyLinkedList class from assignment 6 which we use are add first, remove first, and getFirst. Java's linked list possesses all of these features and thus should be able to easily be implemented as a replacement for the class from assignment 6.
4. It took me almost no time at all to develop the LinkedListStack class as almost all of its features are ripped directly from existing features of the class from assignment 6.
5. The best method of doing this would be to create a class, which we will call DataWithChar, which will be used as a node which stores a character, a row number, and a column number. This would mean that if a (was at the top of the stack and we added a } we could then look inside the data in the node holding the (and find its row and column. In a case where we have an empty stack we can just return output in the same way we do now.
6. 

This graph clearly shows that the push operation occurs in O(1) complexity as the graph remains relatively close to the same values as N increases.



This graph clearly shows that the pop operation occurs in O(1) complexity as the graph remains relatively close to the same values as N increases.



This graph clearly shows that the peek operation occurs in O(1) complexity as the graph remains relatively close to the same values as N increases.

Thus all of these times are O(1) exactly as one would expect of a stack. Stacks are backed with a linked list and the time complexity to access, add, or remove at the first element is also O(1), so it is unsurprising that a stack which only performs those functions would have similar complexities.

7. I spent roughly 10 hours on this assignment.