Dallin Van Mondfrans
u0717113
October 19, 2016

Assignment 7 Analysis

1. No, I have not yet worked with more than one partner. I have already found a new partner to work with for the next set of paired programming assignments.

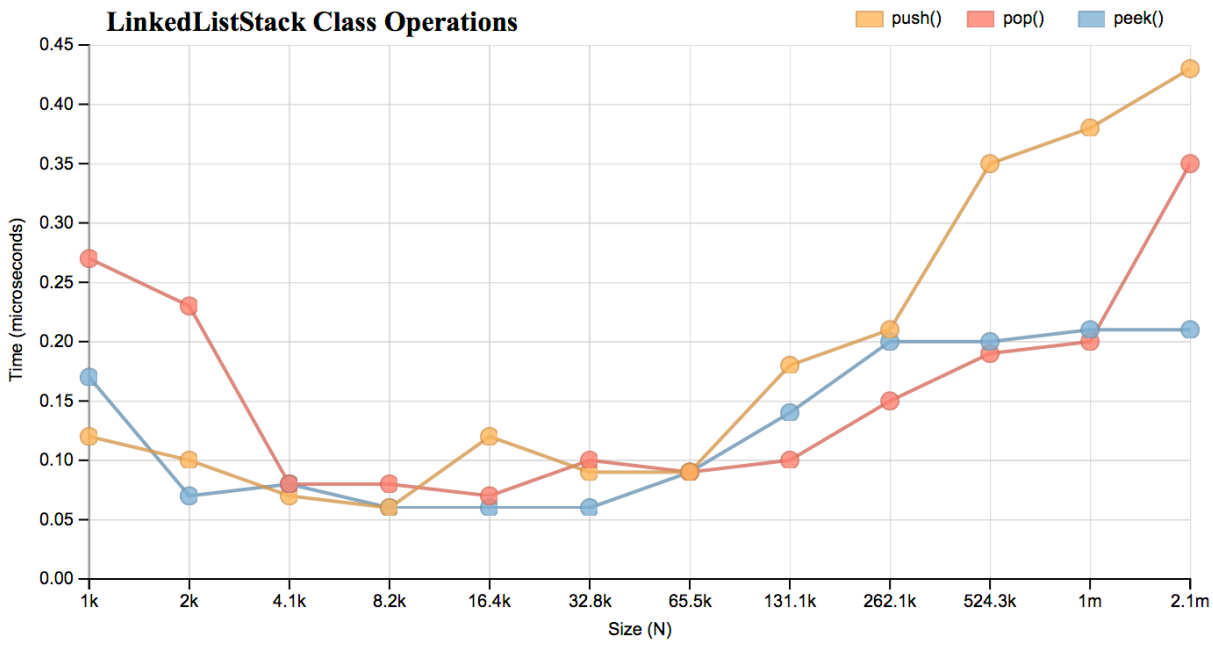
2. No, it would not be better to use a singly-linked list to back the LinkedListStack class. However, I don't think that it would be any worse (performance-wise) either. The way that I implemented my LinkedListStack class is to add elements at the end of the doubly-linked list, which is a constant time operation for a doubly-linked list, but a linear time operation for a singly-linked list. In other words, the "top" of my stack is the tail of my list. Thus, when I pop or peek an item from my list, I am looking at the the tail of my list. However, the "last" item is arbitrary. Thus, if we were to use a singly-linked list, we could call the first item in the list the top, and thus avoid the linear time functionality of adding, or deleting from the list. Thus, in this case, it does not make a difference whether we use a singly- or doubly-linked list to back our stack with to regard to performance. However, in my opinion, thinking of the top of the stack as the end of the list makes more sense to me, in which case, I think that using a doubly-linked list was the better choice.

3. Yes, we certainly could replace our instance of DoublyLinkedList in the LinkedListStack class with an instance of Java's LinkedList. This would be very easy because Java's LinkedList can be treated as a stack, and already has the push, pop, peek, isEmpty, clear, and size methods implemented. We could even say that this would be less ambiguous because we do not need to think about which end of the list is considered the "top" or "bottom" of the stack, Java's class will handle this for us.

4. It took me less than 5 minutes to implement all the required methods of the LinkedListStack class. This is because all methods needed were already implemented but were just called something else (i.e. the pop() method in LinkedListStack is the removeLast() method in DoublyLinkedList). This is a great example of the power of code reuse.

5. One way that we could keep track of the row and column of the opening bracket symbol would be to have two additional stacks, one for the row numbers and another for the column numbers. This way, when we push and opening bracket symbol onto the stack, we can push the corresponding row and column numbers onto their stacks. Then, when we pop a symbol off the stack, we can also pop it's corresponding row and column numbers where we found the symbol.

6. The graph below shows the running times of the pop, push, and peek methods. Although they do seem to follow some sort of a similar trend, the difference in running time of each of the three methods for small N and large N is very small. Thus we can say that all three methods run in constant time, or O(c)



7. I spent about 6 - 7 hours on this assignment.