Michelle Sang
mcs200006
CS 3345.002

ArrayStack:
Complexities:

- isEmpty(): O(1) because we are only checking if the top element is null or not.
- count(): O(N) because we iterate through the whole array of size N to count all the elements.
- resize(): O(N) because we copy N elements from the old array to the new array.
- push(double d): O(1) because we are pushing to the end of the array, unless the array needs to be resized, then it would be O(N) because we copy N elements from the old to new array.
- pop(): O(1) because we are only decrementing topIndex and accessing the value.
- peek(): O(1) because all we need is to access the value at topIndex.

The ArrayStack program implements all the functions that were in BKStack, as well as a function called resize() that creates a new array that's twice the size of the old array, and copies all the elements from the old one to the new one. It starts with an array of INITIAL_CAPACITY which I set to 10. It also begins with the index for the head of the stack to be -1, which I later modify to work with the methods I used. The pop() and peek() functions throw a new EmptyStackException() if the stack is empty, which makes sure that we aren't using those functions on an empty stack.

LinkedStack
Complexities:

- isEmpty(): O(1) because we are only checking if the top element is null or not.
- count(): O(N) because in the worst case, we need to iterate through all N elements in the stack.

- push(double d): O(1) because we are inserting a new item at the top of the stack which has a fixed number of operations regardless of the size of the stack.
- pop(): O(1) because like push, we are deleting an item from the top of the stack which has a fixed number of operations.
- peek(): O(1) because we return the item at the top of the stack and don’t need to iterate through it.
- iterator(): O(1) because we are only returning the iterator.
- StackIterator.hasNext(): O(1) because it only checks whether the current node is not empty and compares the expectedModCount with modCount, which is constant time.
- StackIterator.next(): O(1) because it just checks if it’s empty, updates current, and returns what was at the old spot.

Unlike the ArrayStack implementation, for the ListStack implementation, I had to implement the iterable. I did so by creating hasNext() and next(), which both throw an exception to ensure there aren’t any errors. For hasNext(), I throw a new ConcurrentModificationException() which is for if the list is modified while the methods are being used. For next(), I throw a new EmptyStackException() which is to make sure that we aren’t moving the node pointer when the stack has no other element.

One thing I want to mention is that while there are more methods in ListStack than in ArrayStack, there is only one method that has a time complexity of O(N) which is count(), and in ArrayStack, there are three methods that have time complexities of O(N), which are count(), resize(), and push(double d). It is also important to note that ListStack did not need a resize function because of the use of nodes, so there was no time complexity for resize in ListStack.