Method	ArrayList Runtime	LinkedList Runtime	Explanation
add(T element) method	O(1) if not resized O(n) if resized	O(n)	ArrayList: The method must copy all the code from this array to another array when resized. That requires a for loop that reads all the elements of the initial array. However in regular cases when not resizing, it is an O(1) because it simply adds the element to the index. LinkedList: O(n) because linked lists cannot index any element without running from the start to the end. Therefore, adding the element to the end would require running from start to end and setting the end node next to the element. This requires a while loop.
rotate(int n) method	O(n) * O(p)	O(n) * O(p)	ArrayList: The method has two for loops in place. One is to loop as many times as the parameter says, and the other is to shift all the elements to the right manually. These two loops inside each other create an O(n^2) complexity.

			However, since the for loop on the outside is not for the list size, it is not necessarily O(n^2). LinkedList: The method similarly has a for loop that loops through as many times as the user inputs. This encases a while loop that likewise, shifts all the nodes in the loop to the right and ignoring the last element.
merge(List<T>otherList) method	O(n + p)	O(n + p)	Linked List: There is a while loop present that loops through both linked lists. The method has to loop through the entirety of one list and the entirety of the other to add all the elements into the new sorted merged list. ArrayList: Similarly, this method also must loop through both lists. Although there are multiple while loops, none are encased and none repeat past the size of any of the two lists. But, all elements of both lists must be read and compared which means the

			complexity has to be O(n+p)
reverse() method	O(n/2)	O(2n)	Array List: This is only O(n/2) because the method immediately makes sure the for loop does not run past half the list. This for loop is able to switch opposing elements within the list by using the same I method subtracting for size for the end. Linked List: This is O(n) because of the while loop that loops through the entire linked list. This is necessary because the head constantly needs to change its set next until the list is ran through so that the elements have completely reversed. The method then calls the add function to replace the errant head at the front and place it at the back. This is an O(n) function.