Method	ArrayList Runtime	LinkedList Runtime	Explanation
check_isSorted()	O(4n + 1) where n = size of arraylist	O(3n + 3) where n = length of linkedlist	Linkedlist is a bit more efficient given 3n<4n, however this helper method in both classes probably decreases the inefficiency overall.
boolean add(T element)	O(9) where n = index 0 size of arraylist - 1. Also including runtime of check_isSorted()	O(2n + 11) where n=length of linkedlist Also including runtime of check_isSorted()	Arraylist is much more efficient as it is 9 < 2n + 11.
boolean add(int index, T element)	O(4n + 9) n=length of arraylist. Also including runtintime of check_isSorted()	O(4n + 18) where n=length of linkedlist Also including runtime of check_isSorted()	ArrayList is a bit more efficient since 9 < 18, however given a large value of n, the difference in efficiency is negligible.
void clear()	O(3)	O(3)	The time complexity is the same for these. Neither is better.
T get(int index)	O(3)	O(4n+4) where n=length of linkedlist	Arraylist is far more efficient. That is because it is easy to access a specific index inside an arraylist where in linkedlist you have to iterate through the list to get to a specific index (unless that index = 0).
int indexOf(T element)	O(5n + 2) where n = length of arraylist	O(7n+4) where n=length of linkedlist	Arraylist is more efficient since 5n<7n.
boolean isEmpty()	O(1)	O(2)	Arraylist is more efficient because 1<2.
int size()	O(1)	O(1)	The time complexity is the same for these.

			Neither is better.
void sort()	O(7k(4n)+2) where n=size of arraylist and k=n-1.	O(7n(4k)+1) where n=length of linkedlist and k=n-1	Linkedlist is a bit more efficient, but given a large value of n and/or k, the difference in time complexity between the two is negligible.
T remove(int index)	O(4n + 6) where n = length of arraylist Also including runtime of check_isSorted()	O(4n+12) where n=length of linkedlist Also including runtime of check_isSorted()	The time complexity is the same for these. Neither is better.
void equalTo(T element)	O(5n+4) where n= size of arraylist	O(6n+9) where n=length of linkedlist	Arraylist is a bit more efficient because 5n<6n.
void reverse()	O(3n + 5) where n = length of arraylist. Also including runtime of check_isSorted()	O((k+5)n+4) where n=length of linkedlist And k=runtime of add() Also including runtime of check_isSorted()	Arraylist is more efficient because it does not utilize another method (have n^2 time complexity) like linkedlist does.
void merge(List<T> otherList)	O(6n + 18) where n = size of arraylist / 2. Including sort runtime(one call for this list and another for otherList)	O(16n+15) where n=length of linkedlist Also including sort runtime x2 (one call for this list and another call for otherList)	Arraylist is more efficient because 6n<16n.
boolean rotate(int n)	O(9n + 5) where n = length of arraylist. Also including runtime of check_isSorted()	O(((k + j + 3)n)+4) where n=length of linkedlist Where k=runtime of add() and j=runtime f remove() Also including runtime of check_isSorted()	Linked list is more efficient because 3n<9n.
boolean isSorted()	O(1)	O(1)	The time complexity is the same for these. Neither is better.