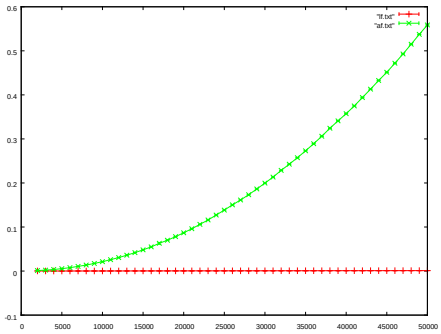
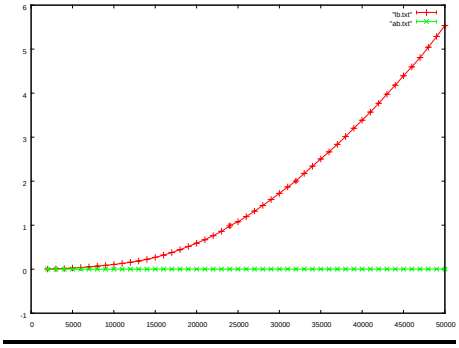
Alexander Niema Moshiri (cs12smj)

## Case 1: LinkedList and ArrayList Both Add From Front




In the first case, I tested adding elements to both list types from the front. As expected, the LinkedList type (shown in red) stays linear at about 0 seconds to make the add, since its add method just creates a new node and sets the previously-first node to be its successor and sets the new node as head. Because of this, no matter how many numbers we have, it's always a simple 2-3 step process and will always be approximately 0 seconds (the graph looks flat at 0 because it is so much smaller in comparison to the ArrayList graph). Because of this linear time complexity, we say that this process was O(n). ArrayList (shown in green), however, needs to move each element after the first one to the right, causing a time complexity of $O(n^2)$, which we can see visually as the curve starts quadratic, but then levels off as linear.

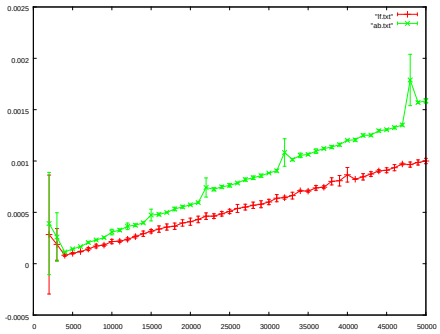
## Case 2: LinkedList and ArrayList Both Add From Back




In the second case, I tested adding elements to both list types from the back. This time, as expected, ArrayList has a time complexity of O(n) since it just adds items to the end. It won't be perfectly linear since it still has to do a copy method whenever it hits its capacity, but it's pretty close. LinkedList, however, has a time complexity of $O(n^2)$. This is because, to add to the end, it has to first iterate through ALL of the nodes until it finally reaches the last node, and THEN it can add an element to the end. As the number of nodes increases, this takes longer and longer.

Again, the ArrayList graph seems to be approximately 0 the whole way since it's so much smaller than the LinkedList graph, but it's actually a line with small slope.

## Case 3: LinkedList Add From Front; ArrayList Add From Back




This time, I put the two to the test by having them do each of their optimum methods: add elements to a LinkedList from the front and add elements to an ArrayList from the back. Now we can see more clearly that both are linear (time complexity of O(n)), and NOT constant time (horizontal line). As expected, we see the small spikes in the ArrayList, which are attributed to what happens when the ArrayList reaches capacity (and has to do the copy stuff to expand its capacity).

## Additional Notes

To avoid having large variance, we needed to have large values for n and many runs for each n. This ensured that the standard deviation for each n would be small (so each plot on the graph would have small error bars) and that we would have many n's to see (to see how the curves continued). What stumped me was the fact that, for ALL values of n, the LinkedList adding from the front was faster than the ArrayList adding from the back. My understanding was that it had to do with the checks and whatnot, the number of instructions, that the ArrayList has to do for each add. Either way, it's VERY close.