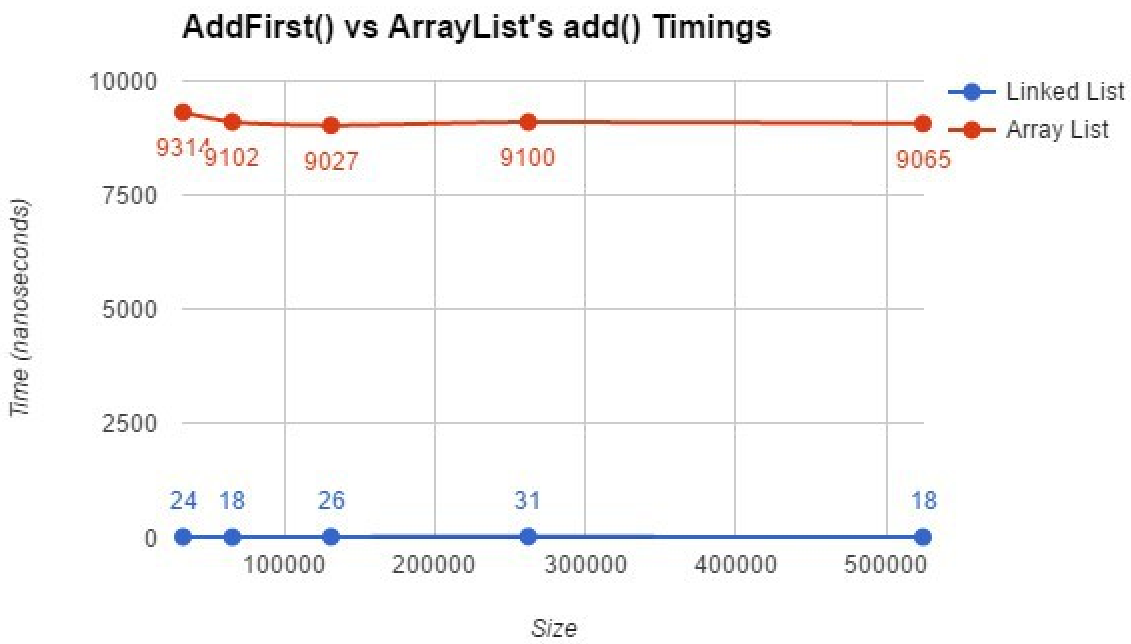
Joshua Shipley
CS2420
Assignment 6 Analysis

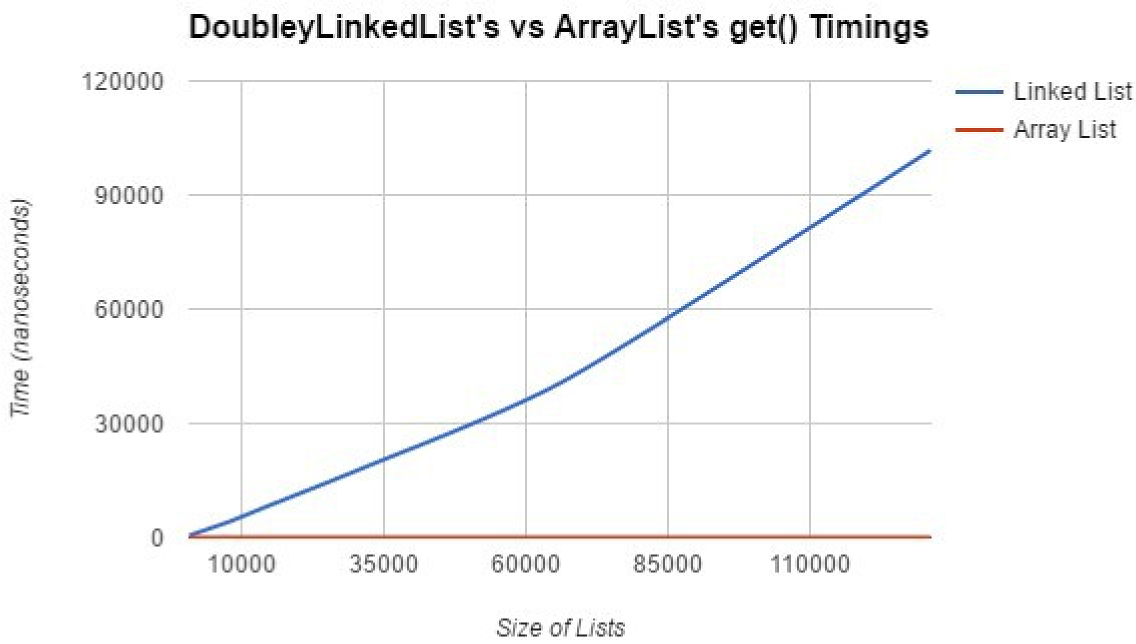
1. Collect and plot running times in order to answer each of the following questions:

- Is the running time of the addFirst method O(c) as expected? How does the running time of addFirst for DoublyLinkedList compare to add(0, item) for ArrayList?

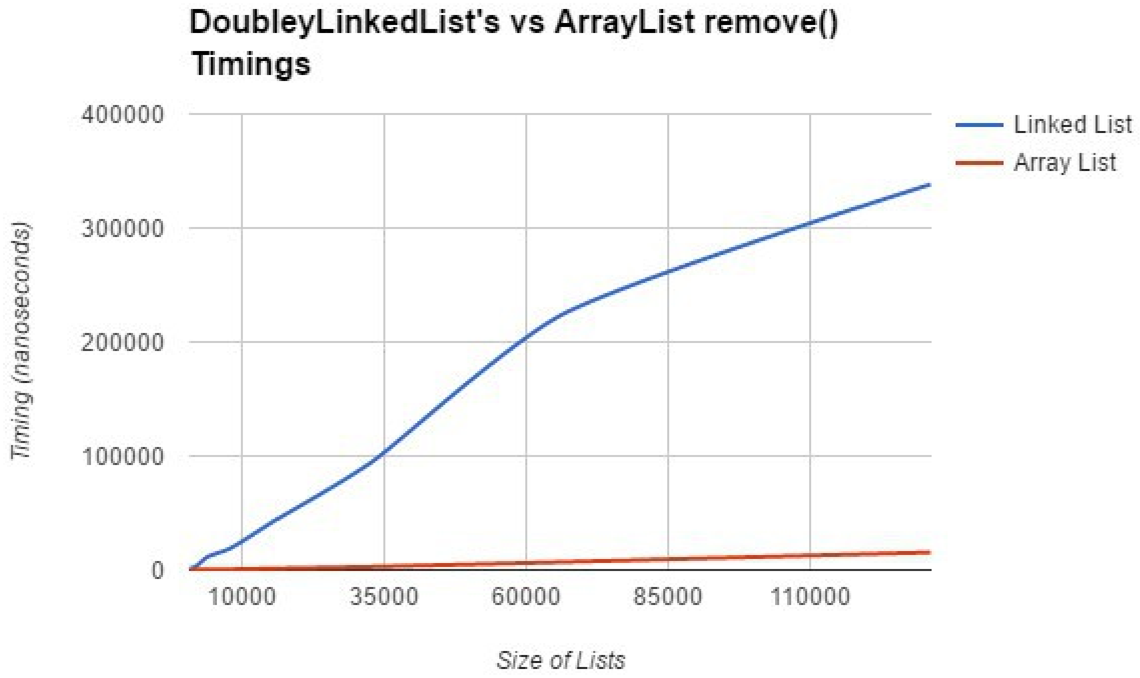


  - Yes, the running time for addFirst was indeed O(c). As you can see in the ArrayList's add() timings, while it is O(c) as well, it is much larger since it has to move over all of the currently present elements first.

- Is the running tme of the get method O(N) as expected? How does the runningtime of get(i) for DoublyLinkedList compare to get(i) for ArrayList?



  - Yes, get() was O(N) as shown in the graph, while ArrayList's was O(c) due to it's ability to directly look up values based on index.
- ➢ Is the running time of the remove method O(N) as expected? How does the runningtime of remove(i) for DoublyLinkedList compare to remove(i) for ArrayList?



  - Yes, remove was O(N), as was ArrayList's. The ArrayList version was much faster, though, since it can go straight to the correct index, even though it had to copy everything back.

2. In general, how does DoubleyLinkedList compre to ArrayList, both in functionality and performance?
   - A DoublyLinkedList isn't directly indexed, so it has to cycle through the entire list to get to the point it needs to look at, making get take much longer than an ArrayList. However, since each element of the list is seperate with simple pointers showing what's "next", and overwriting those is a simple call, adding into or removing out of the list is much faster, as ArrayList's indexing must be almost completely adjusted every time a new element is added or removed.

3. In general, how does DoublyLinkedList compare to Java's LinkedList, both in functionality and performance?
   - A traditional LinkedList only has a pointer to the beginning element of the list, followed by a long, one way chain, making reaching the final element potentially very expensive. However, with the DoublyLinkedList, we also have access to the end of the List, not only making reaching the end an instantanious constant time function, but now reduces the furthest we have to search in an N element array from all N elements down to N/2.

4. Compare and contrast using a LinkedList vs an ArrayList as the backing datastructure for the BinarySearchSet from Assignment 3. Would the Big-O complexity change on add, remove, and contains?
   - Yes, and for the worse for all three, as the reason the BinarySearchSet worked so well was because we could easily skip around, from midpoint to midpoint, cutting down the search size by half until we narrowed it down to where the element should be. This took huge advantage of the indexes within the ArrayList. However, with our current

DoublyLinkedList, even if we had a constantly adjusting pointer to the center of the overall list, we would still have to step through every element just to get to the middle to make a new comparision. Since we would probably pass the element we were searching for on the ways anyways, it'd be better to simply search through the list and check each element as we go, which is what we do at the moment anyways with our current get function.

5. How many hours did you spend on this assignment?
   - I spent about 5 - 6 hours on this assinment total.