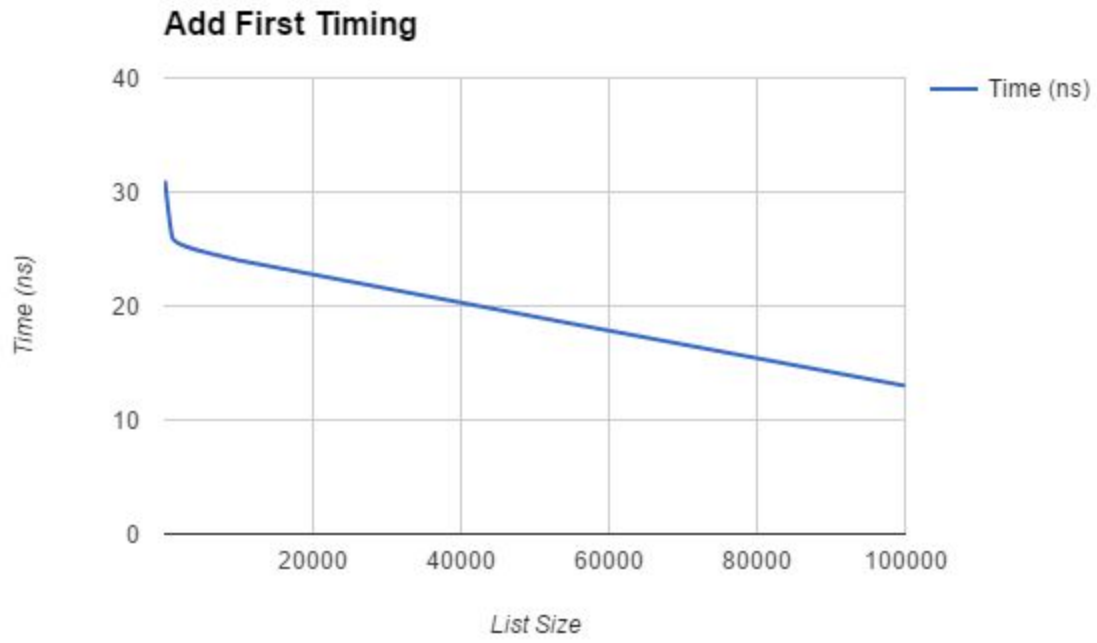
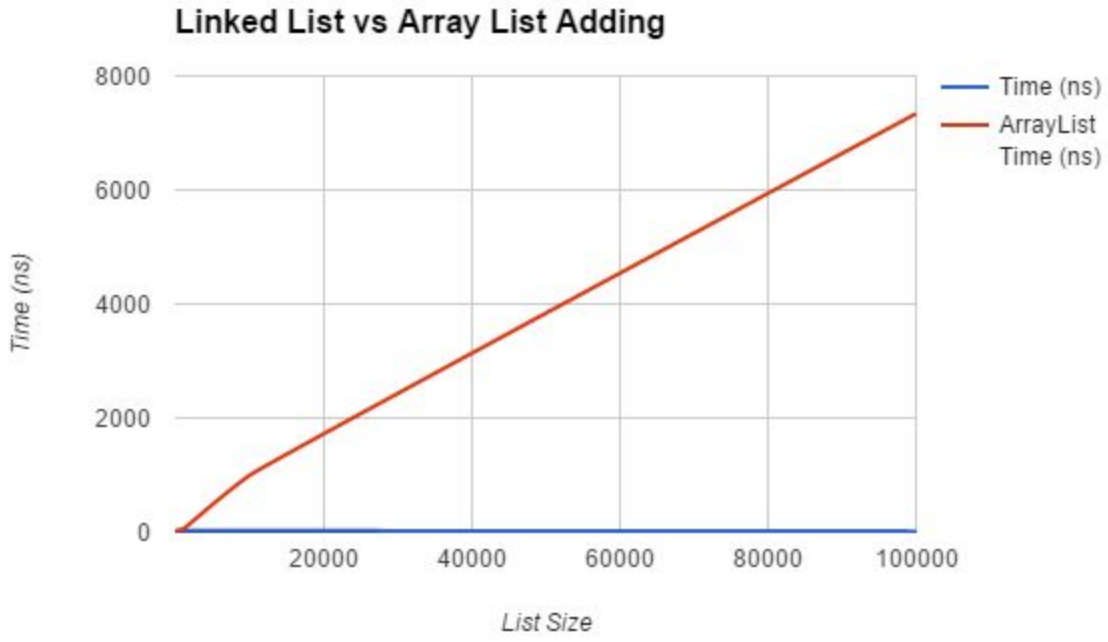
Calvin Kling u0538757 Assignment 06 Analysis

1. Is the running time of the addFirst method O(c) as expected?



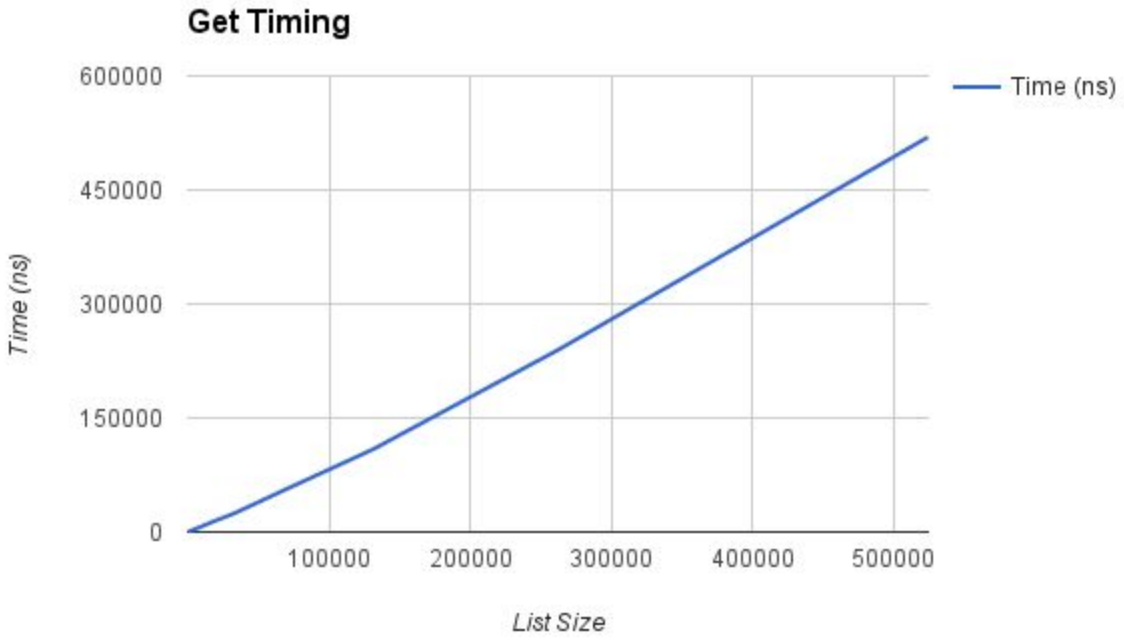
Given the extremely quick execution time and accounting for slight error when calling Nanotime, we are getting the expected O(c) running time.

How does the running time of addFirst(item) for DoublyLinkedList compare to add(0, item) for ArrayList?



No real competition here, a linked list blows the ArrayList away when adding at a known location.

Is the running time of the get method O(N) as expected?

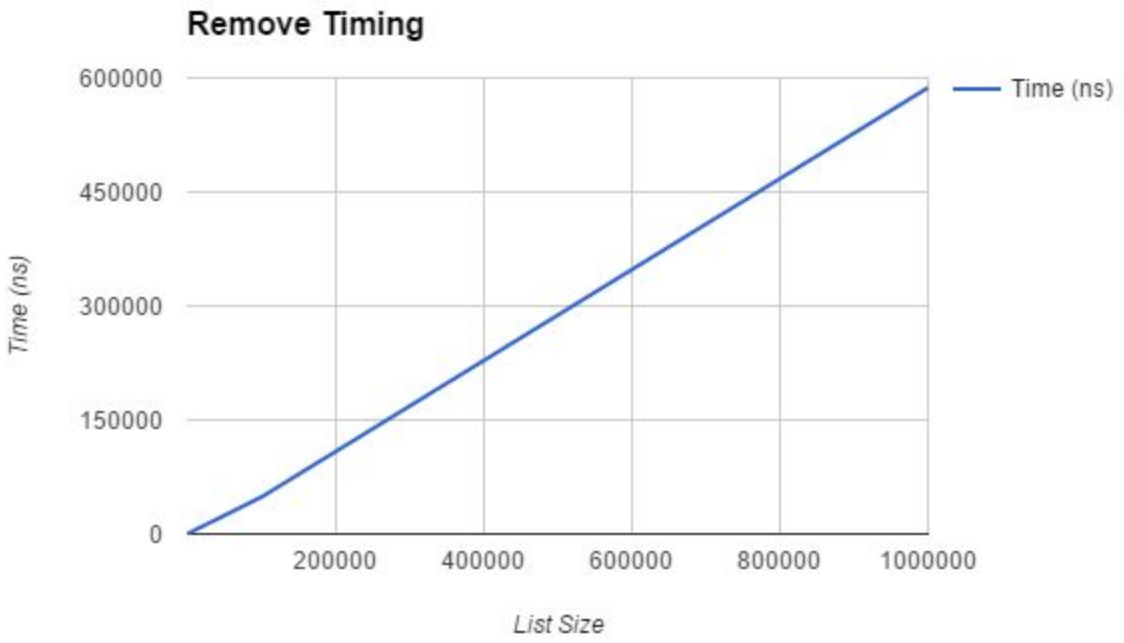


The get method displays almost exactly O(N) running time as expected.

How does the running time of get(i) for DoublyLinkedList compare to get(i) for ArrayList?

As expected, an ArrayList is far more effective at getting elements from a specified index. O(c) ArrayList vs O(N) Linked List.

Is the running time of the remove method O(N) as expected?



The remove method does exhibit a similar growth rate to the get method, which is O(N) behavior.

How does the running time of remove(i) for DoublyLinkedList compare to remove(i) for ArrayList?

ArrayList remove time is much better than I expected. I can only speculate that the major difference is the linked list has to find where to remove before the item can be removed, while the ArrayList knows exactly where to remove and just quickly shifts the rest of the data.

2. In general, how does DoublyLinkedList compare to ArrayList, both in functionality and performance?

A doubly linked list is terrible when searching for an element, but if adding and removing is restricted to the front and end, like a queue, then those operations can be completed almost instantly. While the Arraylist is capable of performing searches in constant time, but any element modification requires linear time.

3. In general, how does DoublyLinkedList compare to Java's LinkedList, both in functionality and performance?

Java’s linkedlist is more complete, however, our DoublyLinkedList implementation contains all of the basic functionality. I find it likely that a professional has achieved better performance than my implementation, but for the methods that are shared, they should have the same Big-Oh complexity.

4. Compare and contrast using a LinkedList vs an ArrayList as the backing data structure for the BinarySearchSet (Assignment 3). Would the Big-Oh complexity change on add / remove / contains?

The contains method would have horrible Big-Oh performance using a linked list, however if add and remove were constrained to only the front and end of the list, we could see major improvements there.

Meanwhile, an ArrayList would offer excellent contains timing and decent random access adding and removing, but would be lacking when adding or removing from the front or end of the list.

5. How many hours did you spend on this assignment?

I spent five hours on this assignment.