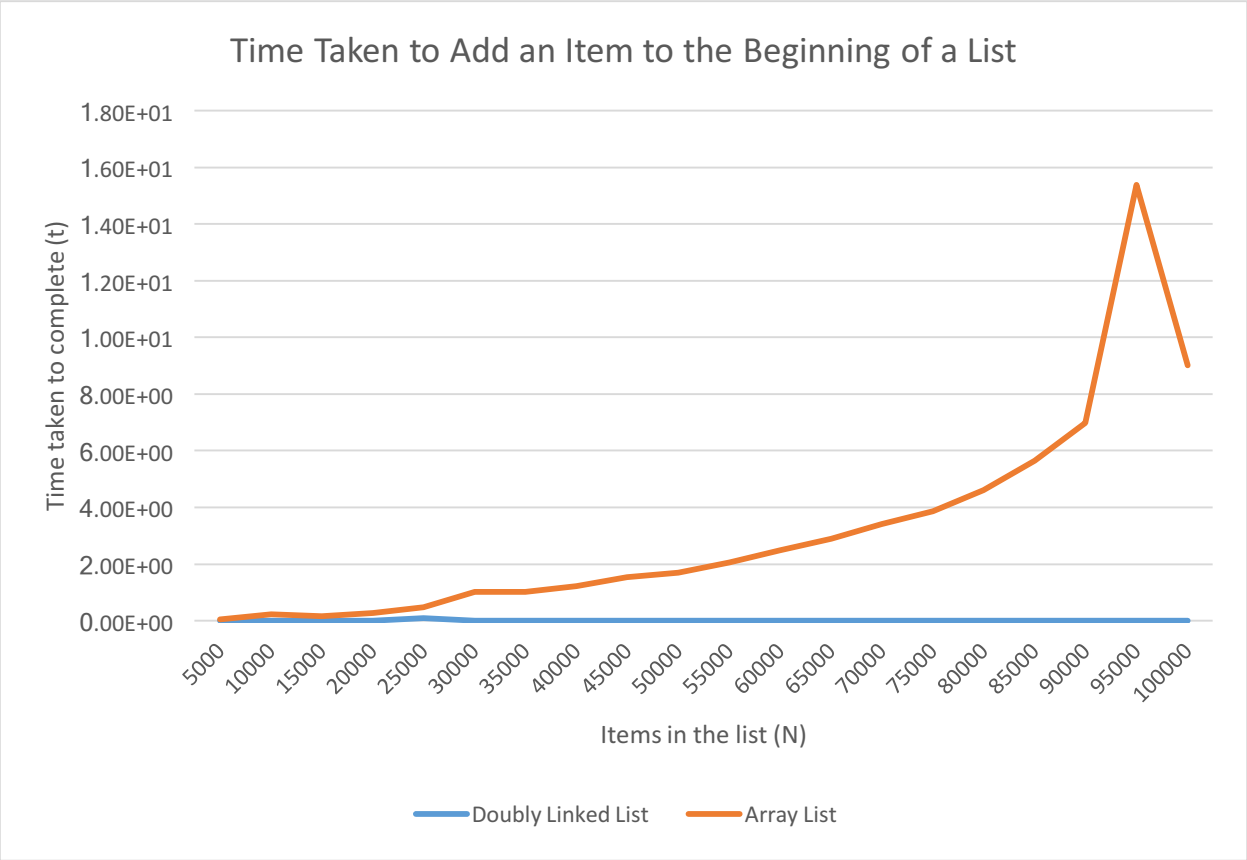
Jacob Virgin, u0832933

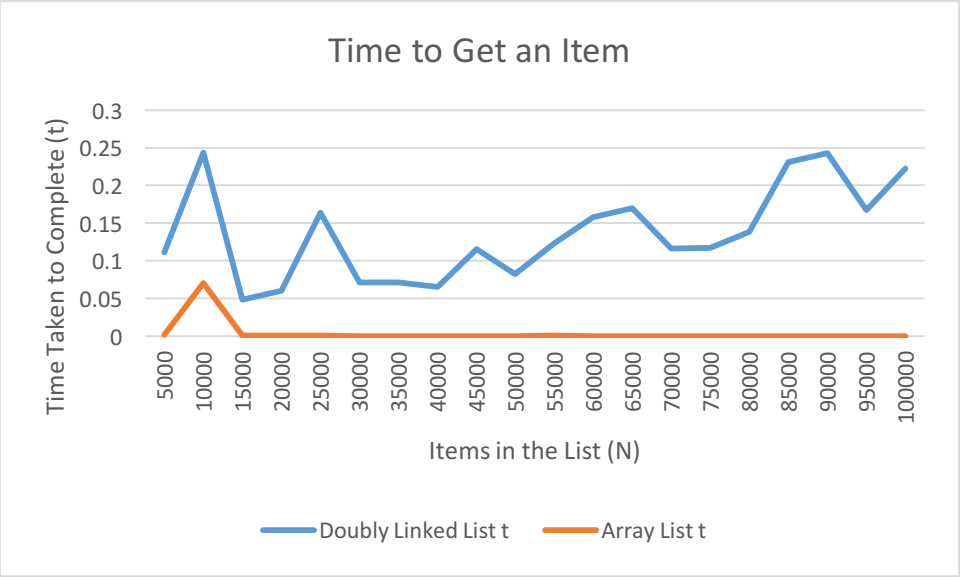
Analysis Document, Assignment 06

1. The running time for AddFirst was just as expected. The time it took to add an item at the beginning of the DoublyLinkedList was constant, exactly as was predicted. Compared to an ArrayList, it also made sense that it would take much less time, as show in the graph below



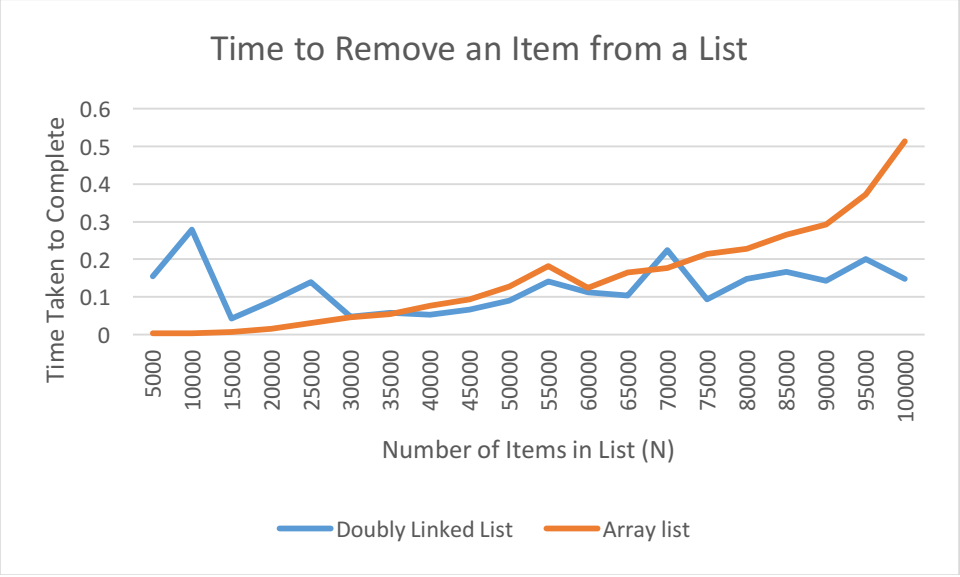
While there was an unexpected spike, the time to add an item to an Array List grew exponentially, while the Doubly Linked List's time remained constant.

The running time for getting an item was also the same as expected in both the Doubly Linked List and the Array List.



The time to get an item was as random as was the number the function was trying to get. No matter what the item was, though, it always took longer to complete than the Array List's method, which was always in constant time.

For a random item in the list, the Doubly Linked List took about the same amount of time continuously with only a little bit of growth. For the same random number, the Array List seemed to grow exponentially to remove the item.



The Douby Linked List needed only to find the item (O(N)) and then remove it (O(c)). The ArrayList had to find the item (O(c)) but then had to remove it and shift everything over(O(N)). In theory they should take about the same amount of time, but the ArrayList graph seemed a lot more in line with what was expected.

2. The Doubly Linked list seems to work better than the ArrayList for accessing and changing items at the beginning and at the end of a list, and for actual addition/removal time. The only thing that is easier on an ArrayList is accessing a random item in the array. However, accessing an item is required for adding and removing items in a Doubly Linked List, and therefore make it more time consuming to add and remove items. However, unless the sole purpose of the data structure being created is to access items randomly, the Doubly Linked list does everything asymptotically faster than an ArrayList.
3. A Doubly Linked List works faster than a regular Java Linked List in regards to the fact that getting an item at the second half of the list takes much less time. It takes about half the time to get an item (on average). It is a marginal victory, even though the asymptotic behavior is still the same.
4. Using a Linked List for a BinarySearchSet would be much less effective than using an ArrayList. A BinarySearchSet uses getting a specific item to its advantage to compare whether the item is in the first half or the second half of the array. Using a Linked List would ruin all of that optimization, needing to go through the entire first half of the array before testing whether or not the item is in the first half or the second half. It would be more efficient just to go through the entire array to find the item. Adding, removing, and getting an item would change from a O(Log N) to more of a O(N) complexity.
5. I spent about 5-8 hours on this assignment