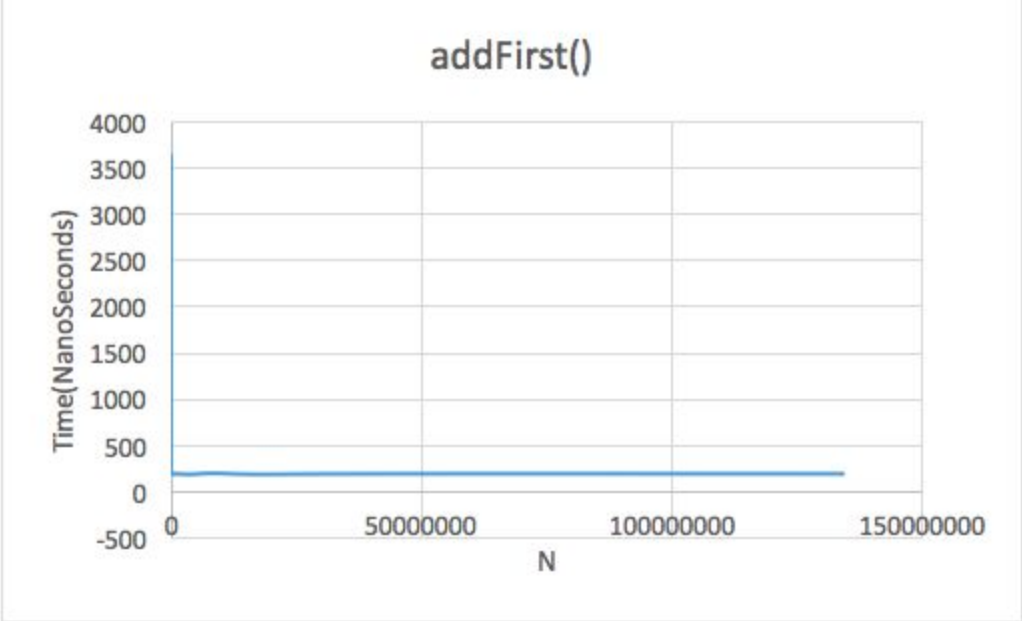
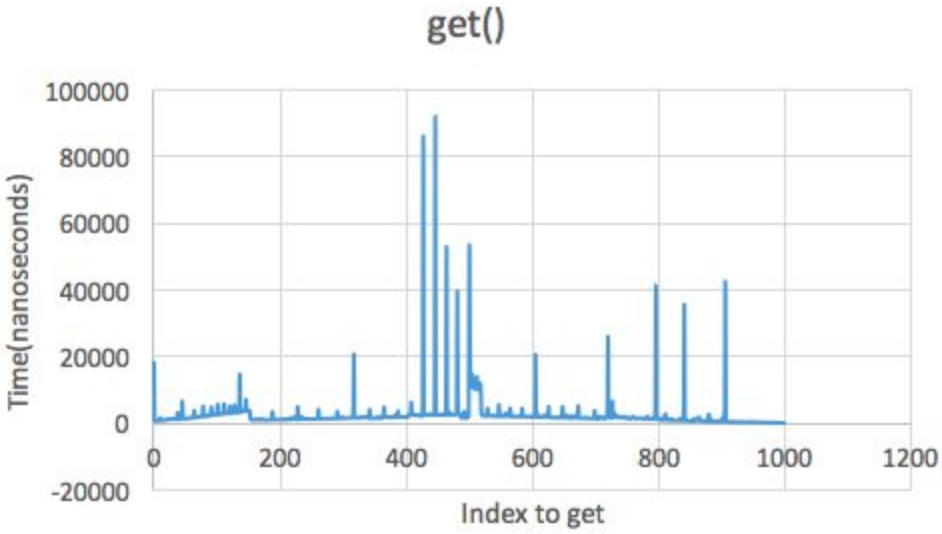
Anthony Iovino
Assignment 6 Analysis

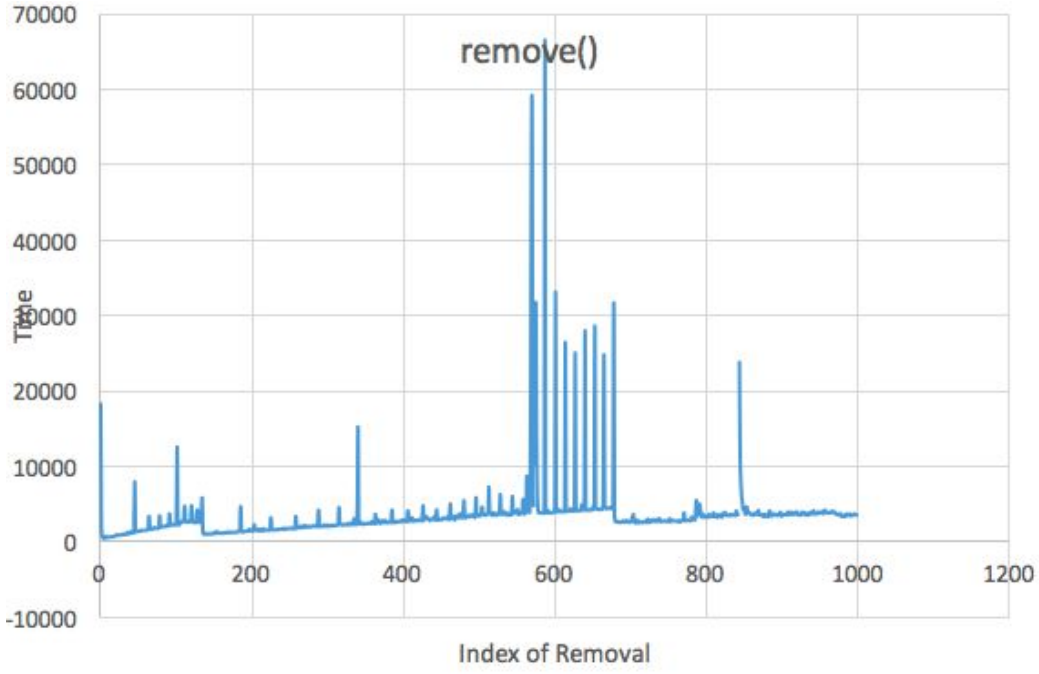
1.



Yes, the running time for this method exhibits constant behavior as expected. The constant time makes sense considering that a similar computation is a used each time to add the element; there is a no linear array like structure to traverse. The observed behavior differs from the behavior of ArrayList's add method in so far as it doesn't require that each element(N) in the linear array is moved to the right in order to insert the new element in the front (0 index) of the array.



For this graph, I created an array of size 1000 and tested the performance relative to the indices of insertion. The get method does appear to exhibit the behavior that was expected. The performance is dependent on the index position at which the item is to be retrieved. The pattern stems from the fact that the index of the item to be gotten will vary in its relative position to the left and right side of the list. Items towards the center of the list will inherently take longer because they are farthest from the extremes. The running time behavior of this method is the same as that which would be demonstrated by ArrayList's get(index) method purely because a for each loop is used in Java's implementation. In other words, a linear traversal that is not needed to retrieve the item.



For this graph, I created an array of size 1000 and tested the performance relative to the indices of insertion (code attached). Again, the plot does appear to demonstrate dependency on the index position. Furthermore, there appears to be a somewhat linear slope in the overall trend of the graph. If the outliers are ignored, a larger N appears to influence the performance by increasing the time it takes to remove the item. This is unlike the ArrayList's get() behavior which exhibits N like behavior because of the necessary shifting of items to the left following the removal of an item.

2. The main functional differences between these two classes lies in the need to shift items in a linear data structure, in one direction or another, following an insertion or removal of an item. ArrayList data structures are forced to shift every item to the right following insertion and to the left following removal. This limitation does apply to the DoublyLinkedList, in which a simple computation is used in the addition of an item. For these reason the overall performance of our implement linked list appears to be better than what would be expected from ArrayList.

3. The functionality and performance of these two classes is rather similar. Unlike java's Arraylist, LinkedList is not limited to using a linear array like data structure to add and remove items. However, the overall implementation of Java's LinkedList is probably more stream lined.

4.The first difference lies in the need to reposition every element in a list of values following the add or remove method. LinkedList, unlike the ArrayList, is free of this performative burden and would thus not be compounded by N during a BinarySearchSet call. The contains methods however, at least for large N, would ultimately be the same for ArrayList and LinkedList because there is no need for repositioning any items. With contains, an item is not moved, it is merely found.

5. 7 hours