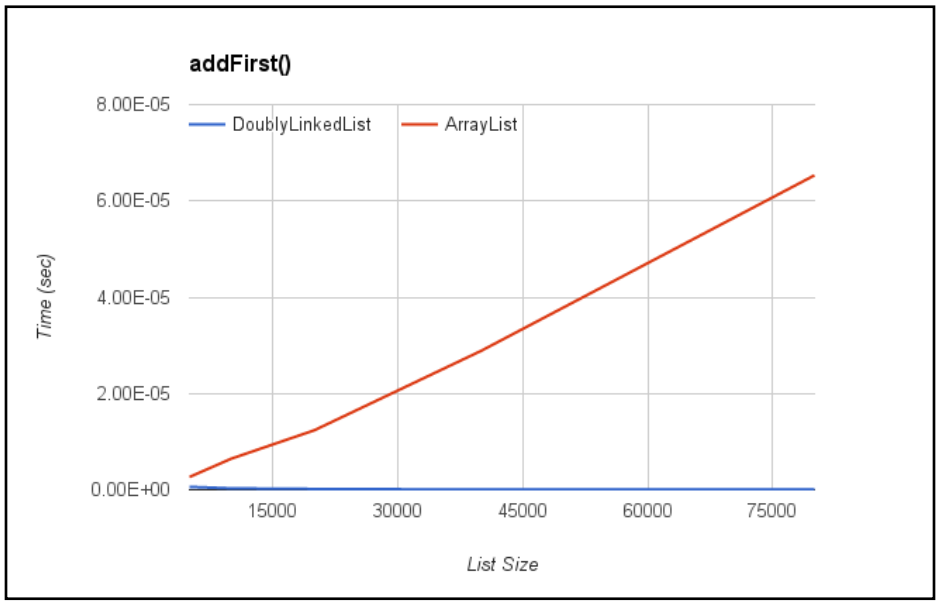
Jana Klopsch
U0854469
CS 2420

Assignment 6 Analysis

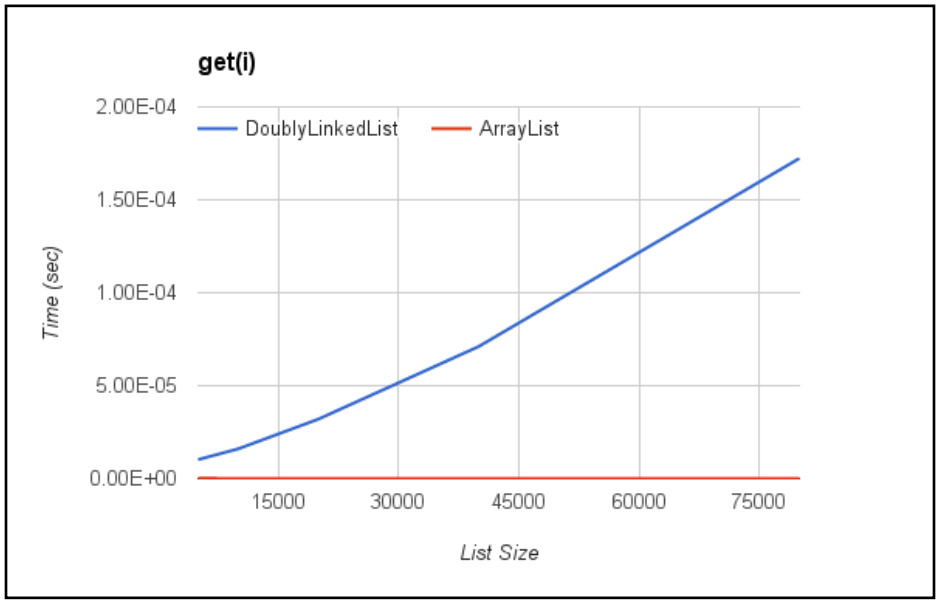
1. Collect and plot running times in order to answer each of the following questions.
Is the running time of the addFirst method O(c) as expected? How does the running time of addFirst(item) for DoublyLinkedList compare to add(0, item) for ArrayList?

The running time of the addFirst was O(c) as expected. Especially when compared to the add(0, item) method for ArrayList, which has running time O(N). In DoublyLinkedList, to add an item to the beginning of the list, only links of the head node and the new node being added are changed, and the new node is set as the head node. This has the complexity of some constant, because the same steps are performed the same amount of times, regardless of how large the doubly linked list is. For ArrayList, however, the new item is set at the index zero, and every item following this has to be shifted down to the next index. For an ArrayList of size N, the complexity is O(N).



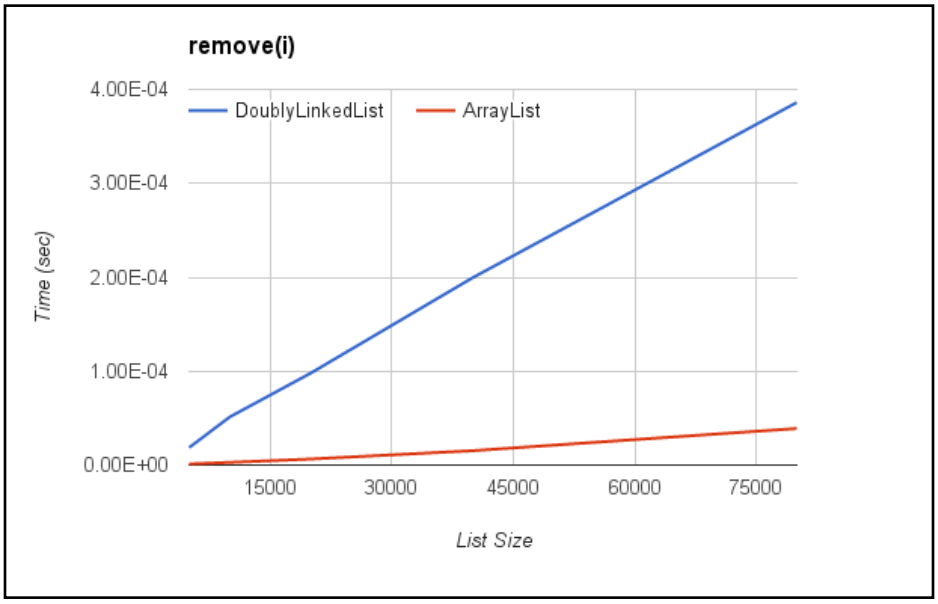
Is the running time of the get method O(N) as expected? How does the running time of get(i) for DoublyLinkedList compare to get(i) for ArrayList?

The running time for get(i) for DoublyLinkedList was O(N) as expected. Comparatively, the running time for get(i) for ArrayList was O(c). For DoublyLinkedList, to find of get an item at a specific index, the most direct ad efficient way to get that item is to traverse through the list, either starting at the tail or head of the list (depending on what end the desired index is closer to). This means stepping over every item passed until that item is reached, which has Big-Oh Complexity O(N). In an ArrayList, because the elements each have a specific index, one can directly access a specific element by going directly to its index in the ArrayList. This only takes a few steps, regardless of the size of the ArrayList, which has Big-Oh Complexity O(c).



Is the running time of the remove method O(N) as expected? How does the running time of remove(i) for DoublyLinkedList compare to remove(i) for ArrayList?

The running time of the remove method for DoublyLinkedList is O(N) as expected. In addition, the running time for the remove method for ArrayList was also O(N). Because the DoublyLinkedList has to traverse through the list to first find the item being removed, it makes sense that this method has running time O(N). For ArrayList, it is easy to access the item being removed through the index provided, however, once this is done, every item following the removed item must be shifted over to fill in the now empty spot. Although graphically, these methods appear to have different running time behaviors, the maximum difference in time taken to perform remove is approximately 0.00035 seconds.

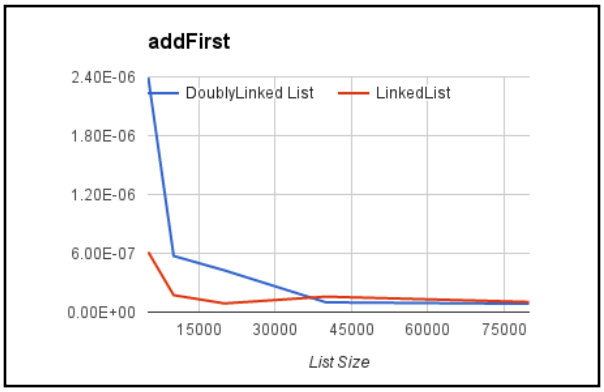


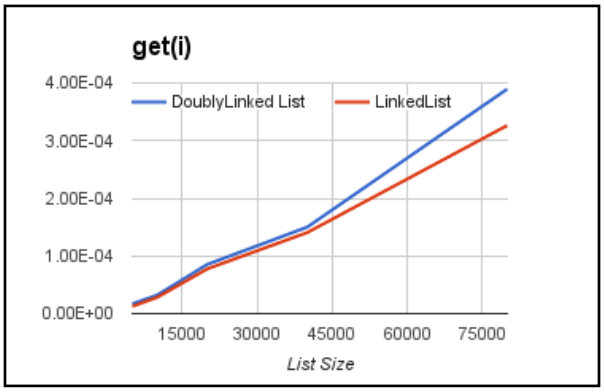
2. In general, how does DoublyLinkedList compare to ArrayList, both in functionality and performance? Please refer to Java's ArrayList documentation.

DoublyLinkedList and ArrayList both could be used for similar purposes, however, due to functionality differences one may outperform the other. In the case where an item is being added or removed from the beginning of a list, DoublyLinkedList will outperform ArrayList. In the functionality of DoublyLinkedList, that item can simply be removed without effecting the rest of the list, while in ArrayList, every item following the first has to then be shifted over. If the list is being used to get an item at a specific index, ArrayList will outperform DoublyLinkedList because it does not need to traverse through the list to find that item. In the case of removing, DoublyLinkedList performs best when the item being removed is close to either the beginning or the end of the list, because the remove method doesn't need to traverse very far to find that item. For ArrayList, remove performs best when the item being removed is close to the end of the list, because then less items have to be shifted after this item is removed.

Depending on which method is going to be used most, one of these structures will perform better than the other. In addition to the performance of these methods, however, ArrayList also uses more memory than DoublyLinkedList for the add methods. Anytime an item is added to an ArrayList (except when it is added to the end), a copy of the array must be used to store the old data from the array, and then it must be copied back over to the updated array. Therefore, if memory needs to be taken into account, DoublyLinkedList has more optimal functionality in its add methods.

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





In the above three graphs, DoublyLinkedList was timed against Java's implementation of LinkedList. The blue line represents the timing for DoublyLinkedList, and the red line represents the timing for Java's LinkedList. All graphs are graphed with list size on the x-axis, and time in seconds on the y-axis.

In comparing DoublyLinkedList and Java's LinkedList, the timing behaviors of both are very similar. Functionality of both classes is the same, with some slight implementation differences. From the graphs, it is easy to see that they have very similar algorithms, however some subtle differences can account for the timing differences in all three methods.

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?

In the add and remove methods, the Big-Oh complexity would change to be linear using a LinkedList. The items in a LinkedList would not need to be shifted over (as they would in an ArrayList) to accommodate the item being added or removed, just a linear search is used to find the index of the item being added or removed. For the contains method, the ArrayList would outperform LinkedList, as a specific index could be directly accessed using ArrayList while a LinkedList would have to be traversed. Using a LinkedList for BinarySearchSet, the complexity for the contains method would also be linear, as opposed to constant for ArrayList.

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

10