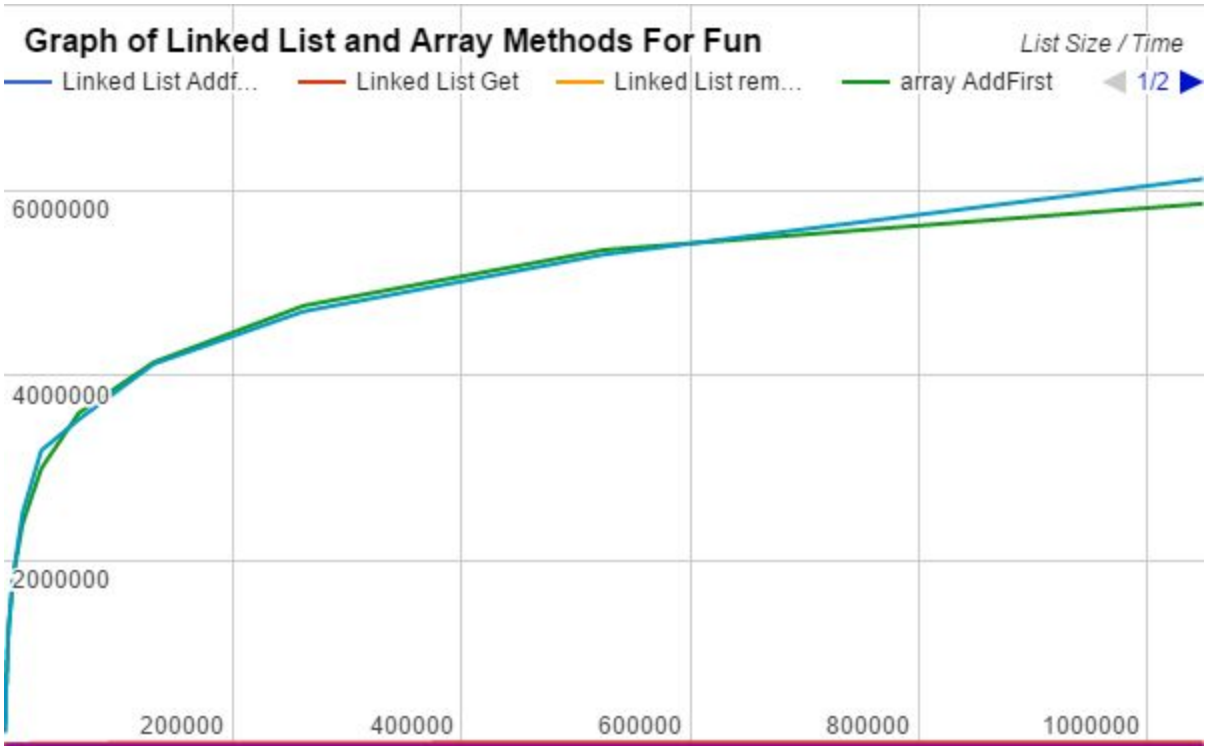
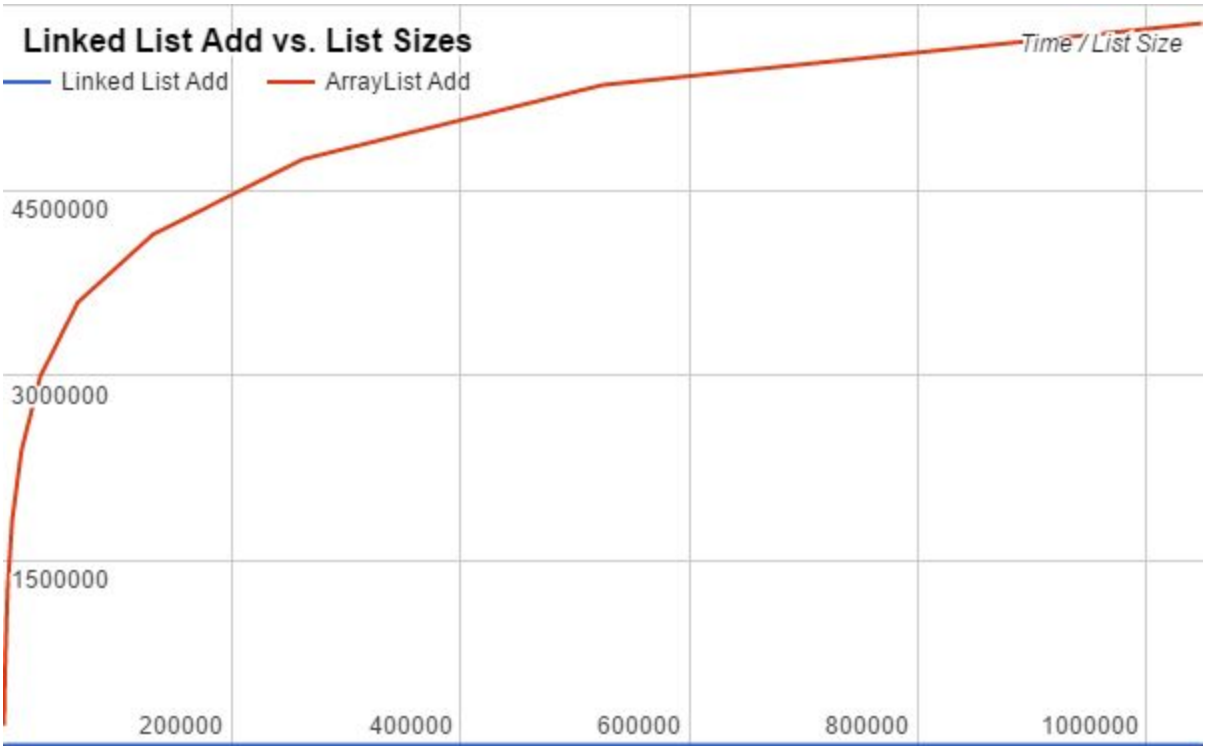
Travis Taylor Cassity
Assignment 06
10/5/2016



1.Collect and plot running times in order to answer each of the following questions. Note that this is this first assignment that does not specify the exact procedure for creating plots. You must design your own timing experiments that sufficiently analyze the problems. Be sure to explain all plots and answers.

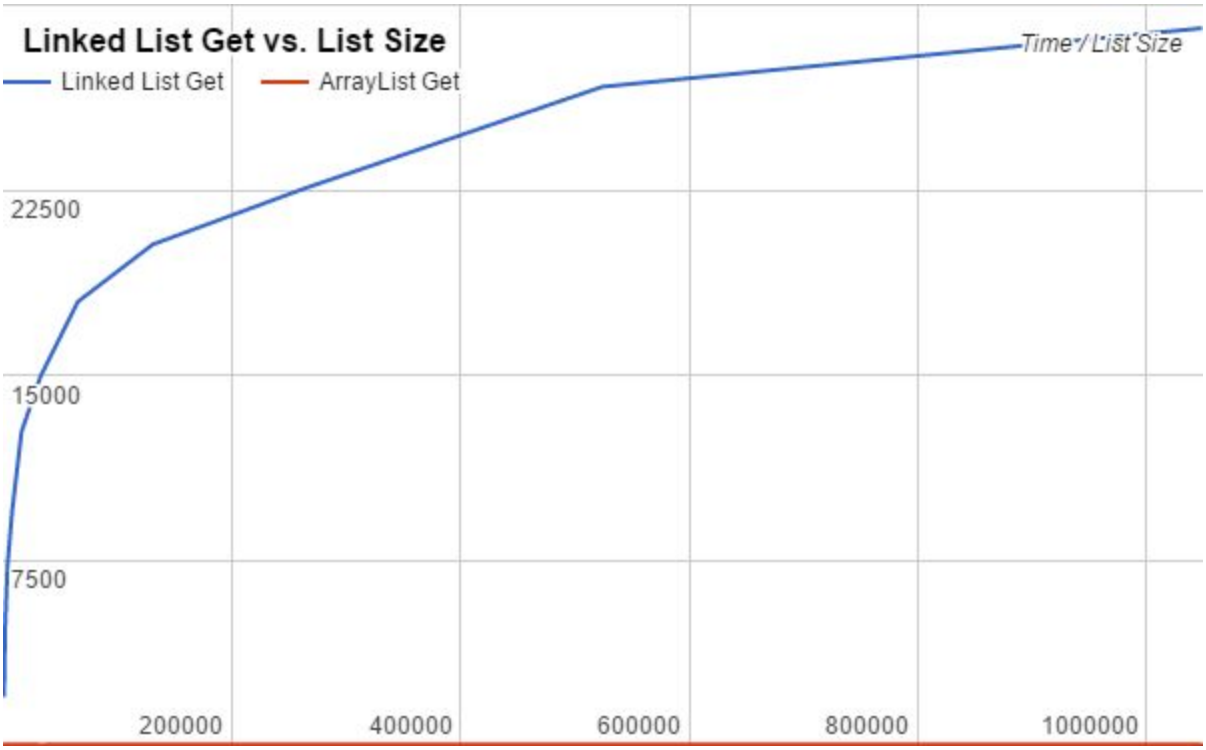
- **Is the running time of the addFirst method O(c) as expected?**
- addFirst is O(c) as expected. Hard to see it on the graph, but it's at the bottom, going in a flat line, as it should for Linked Lists. ArrayList is O(N), as you can see, though it looks a bit funky as the arrays I used were exponential, rather than linear.
- As a heads up, for readability, I'm putting the charts below the first question that references that chart, as you can see just underneath here.

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

- 

- addFirst for Linked List is O(C), as it's just connecting the next/previous to the new item. For ArrayList, it's O(N), as all of the items to the right must be shifted right one, then the new variable is added. This is made even worse if the ArrayList has to grow.
- **Is the running time of the get method O(N) as expected?**
- Assuming this question refers to Linked List. Yes, Get flies off, gradually, into the sunset. Not as steep as N^2, but enough to (roughly) be N. The graph below has the LinkedList N graphed on it, and it's that itty, bitty, tiny blue line at the bottom, that never grows or shrinks, so it's easy (sorta) to visually see that it's just O(N), on top of just understanding that it has to iterate through each node one at a time to get to where it's going. While it does look like it "spikes", it's really just following the list size.

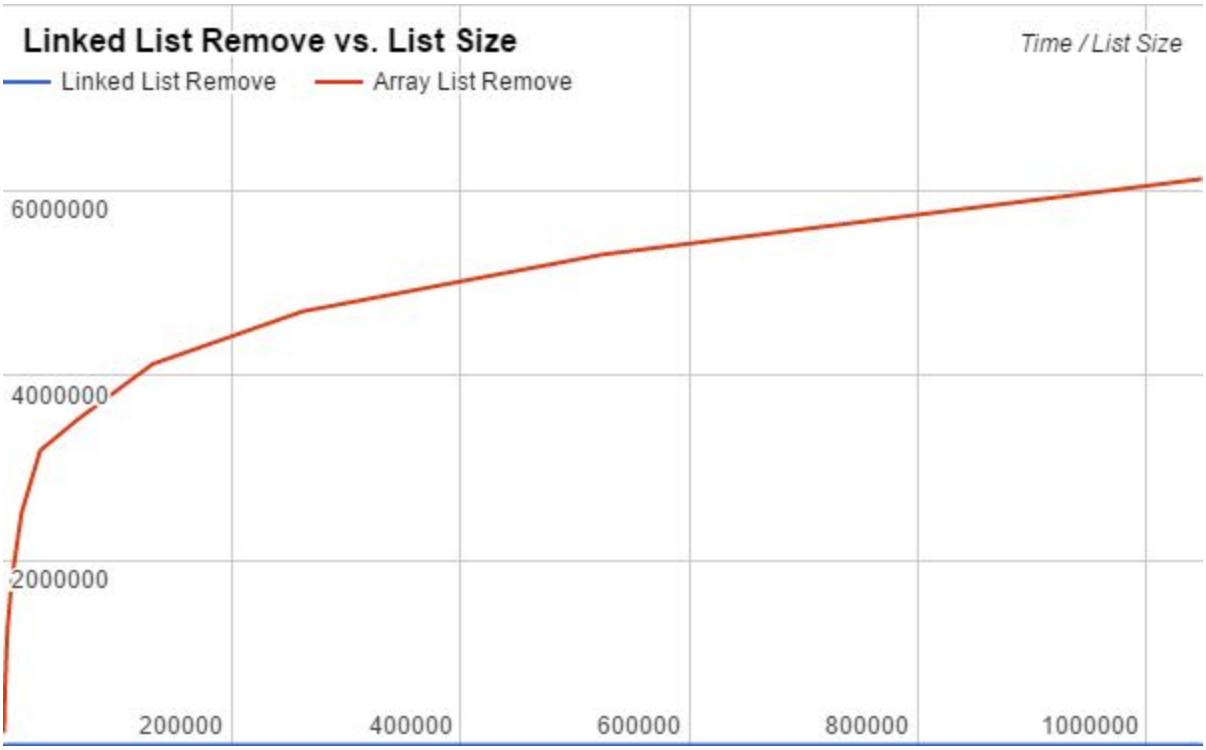
- **How does the running time of get(i) for DoublyLinkedList compare to get(i) for ArrayList?**



- ArrayList's Get is significantly faster; specifically, it is O(C). Linked lists's is N, as it must iterate through the linked list one at a time. ArrayList uses multiplication to very quickly jump to the given index. Although the graph looks somewhate skewed, it's just N following along with the size of the list.

- **Is the running time of the remove method O(N) as expected?**
- Assuming this refers to ArrayList. Yes, it is, as it must remove the value, then shift everything to the right of the index, left one. The graph below visualizes this; ArrayList grows at a roughly N rate (though it looks skewed due to the increasing size of the arrays).

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



- For Linked List, it is O(c), as it is merely a matter of disconnecting the node from the previous/next nodes, and connecting those two nodes together. For ArrayList, as answered in the previous question, it is O(N) due to needing to shift all the elements down one. As you can see on the graph above, LInked List's remove is basically just a flat line at the bottom. Almost looks like it's just a border for the graph. This clearly tells us it's just a constant O(C). While Array List's sort of tilts in an odd way, it is just N. It looks a bit funky since the array sizes I tested with were not linear, and scaled up very quickly.

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

- "In general", I would say Linked List outperforms ArrayList, since two common functions - add and remove - are O(C). However, for retrieving data, ArrayList is far better, so both have their own merits. The Java Documentation roughly agrees, pointing out ArrayListh as some O(C) advantages, whereas Linked Lists have other fuctions with O(C). What I want to know, is it possible to combine an ArrayList and LinkedList DataStructor to make all three methods O(C)? I would assume so; I'd bet Google uses something like that.

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

- ArrayList is far better at retrieving data (O(C)), whereas Linked Lists are much better at "handling" data (moving, adding, and removing [O(C)]). Both have similar functionality, such as having indexes and growable sizes and many of the same functions (add, remove, etc). However, ArrayLists use multiplication to jump to indexes, giving it O(C) for

retrieving data, and capable of easily performing different searches, such as Binary Search. Linked Lists must iterate one node at a time, giving them O(N) for retrieving data. They also cannot take advantage of some methods like Binary Search as easily. However, they can be sorted just fine, although the overhead might be more than that of an ArrayList to sort.

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?**

- Not for the best, no. Everything uses a Binary Search, which it itself takes advantage of jumping to indexes to search for values. The LInked List data structure has to slowly wobble its way through the list, one node at time, making the Binary Search far less potentially optimal. The Add, Remove, and Contains all use this, meaning the Linked List would have to slog its way through the list countless times as it runs the Binary Search to execute all of these methods. It would not only not be any faster, it could potentially be much slower (Again, this all depends on how sorted the data we're dealing with is). Just to start the Binary Search would have O(N), as it has to slog to the middle of the Linked List just to start it (N/2 is still O(N)).

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

- About 11~ hours in total.