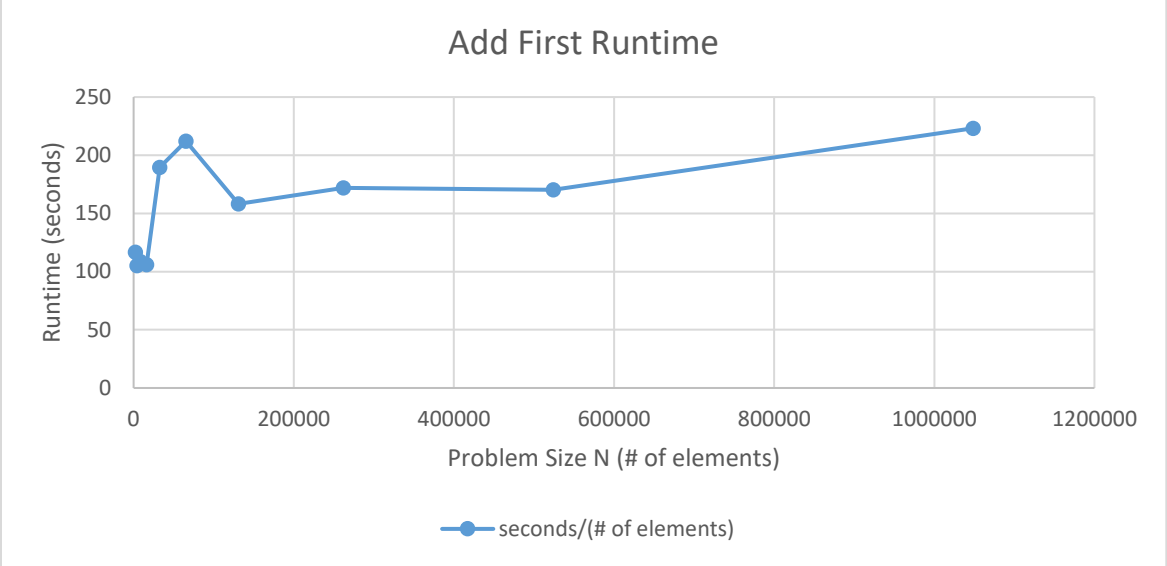
Prathusha Boppana

U0778008

CS 2420

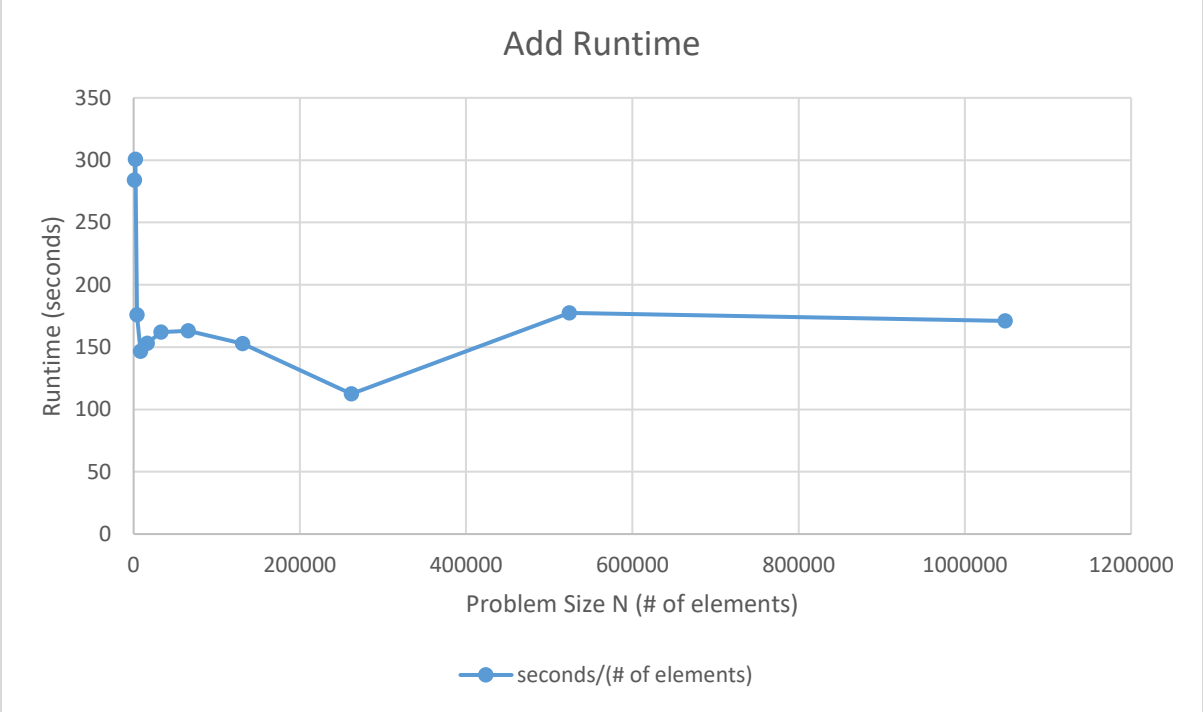
Assignment 06: Analysis

1. Our addFirst method should be O(c) and our add method should be O(N).
   Here is the addFirst method graph:




   There is a bit of skew in the beginning, but it levels to near constant runtime afterwards.

   Here is the add method graph:



There is quite a bit of skew in the beginning, but it changes to O(N) afterwards. Then it changes to constant time.
We expect to have addFirst produce a better graph than add and it did. Plus, the add method could be inserting an element at any point in the list, so we can expect that the data will be a little scattered.
The addFirst method for DoublyLinkedList is O(c) because the linked list just has to change a few links to make a new head. Meanwhile the add(0, item) method for ArrayList is O(N) because it has to copy the elements before the item, add the item, and then add the rest of the elements to a new ArrayList causing more work than a linked list.

The get and remove methods have not been working for me, but the get method should be O(N) and the remove method should be O(c).
For the remove(i) method, DoublyLinkedList has to traverse the linked list in order to calculate get(i), which is O(N), and then removes it by reassigning a few links, removing access to the element. This is accomplished in O(c) time whereas the remove(i) method in ArrayList has to shift all of the elements back one index after it has found the index. Although finding the index with an ArrayList is O(c) because the ArrayList just has to access an address in order to calculate get(i).

2. DoublyLinkedList functions by keeping track of a head and tail node and using links from one node to another to keep track of the elements. ArrayList functions by keeping track of a list that is referenced by an address and doubles the space of ArrayList when the space runs out. Overall, DoublyLinkedLists are useful when adding and/or removing items frequently because those operations are O(c) for DoublyLInkedList. ArrayLists are useful when accessing items frequently is a priority because those operations are O(c) for ArrayList.

3. DoublyLinkedList is a LinkedList, but with a tail node to use as well. The complexity of a DoublyLinkedList is actually half the complexity of a LinkedList because the user can traverse the list from either side, but ½ is just a constant so it doesn't end up mattering in the overall complexity.

4. The best data structure to use for BinarySearchSet would have been a LinkedList for the add and remove methods, but not for the contains or splitting parts of the BinarySearchSet. For a LinkedList, the add method would be O(c), the remove method would be O(c), and the contains method would be O(N). For an ArrayList, the add method was O(N), the remove method was O(N), and the contains method was O(N).

5. I spent around 16 hours on this assignment.