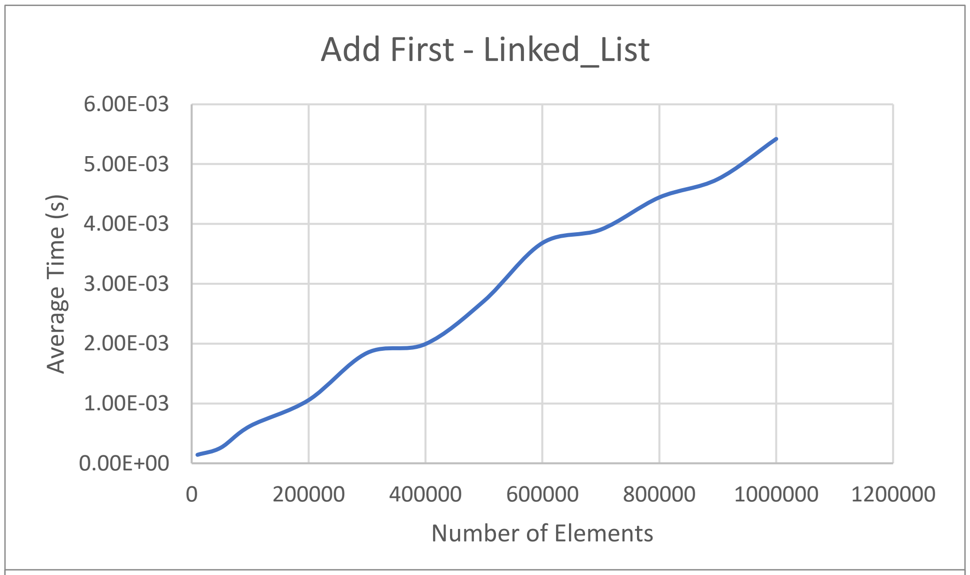
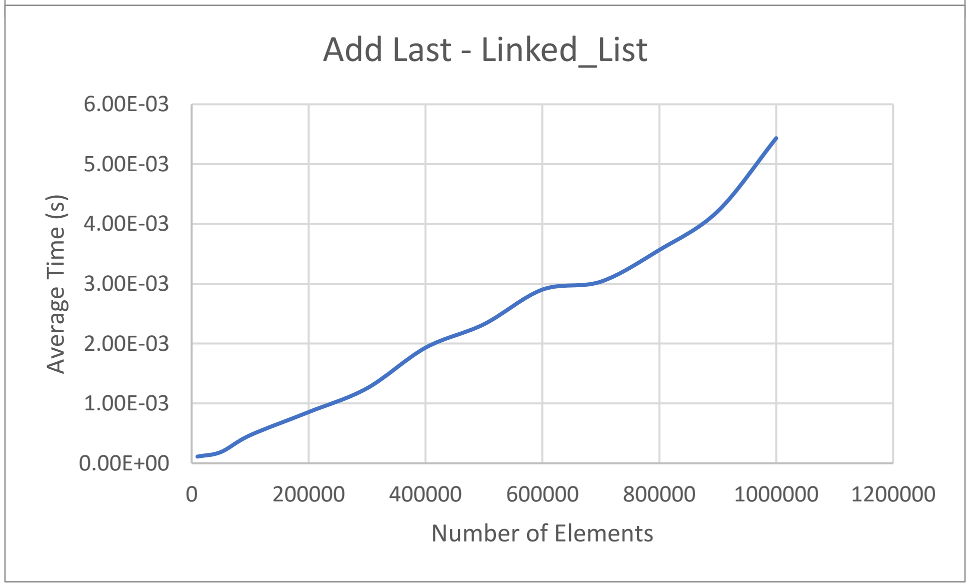
Landon Crowther
U0926601
2/23/17

Assignment 6 Analysis:

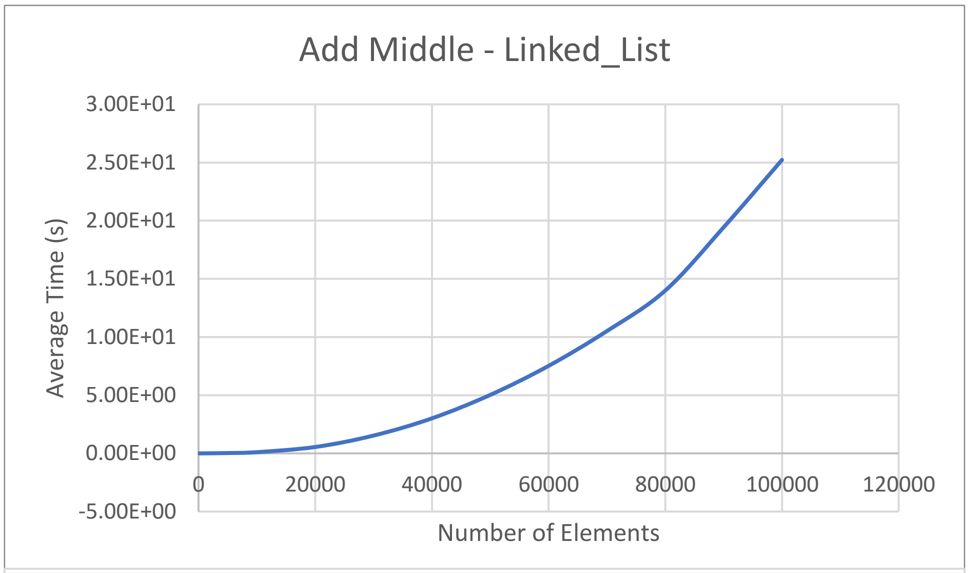
Timing Analysis:

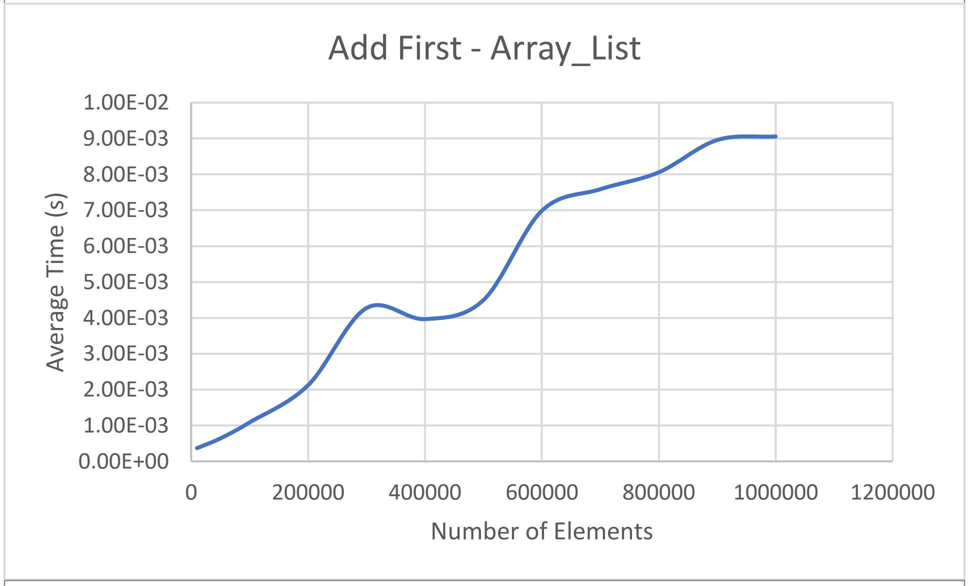
The following figures were produced after following the instructions on the timing code:

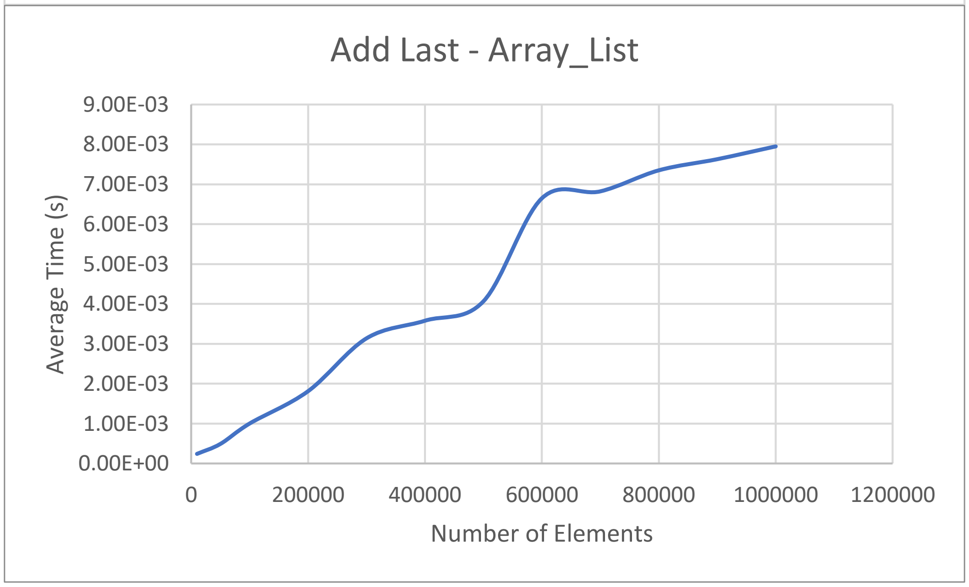




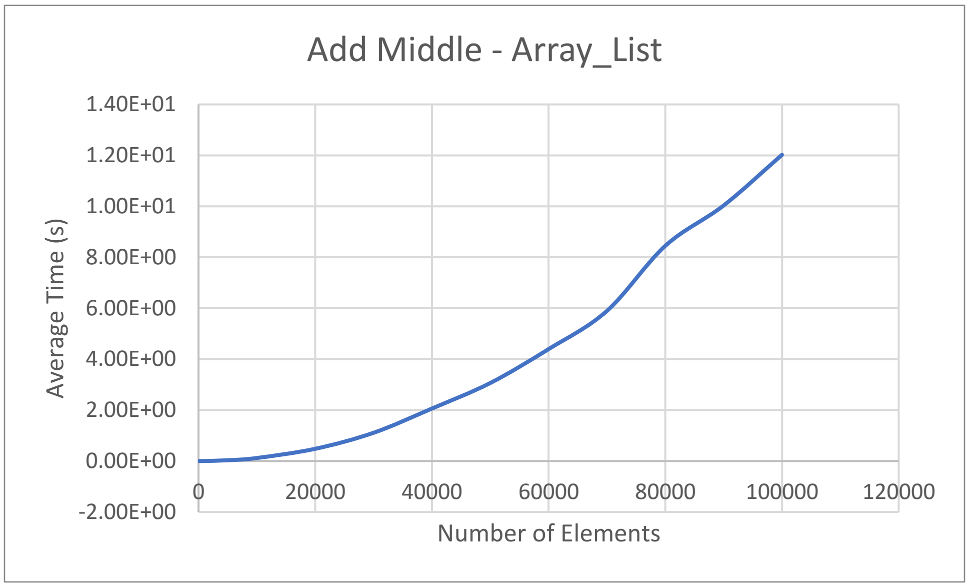
Landon Crowther
U0926601
2/23/17







Landon Crowther
U0926601
2/23/17



Overall, I believe that our implementations follow the big O behavior quite well. I wish we would have been able to have more array sizes for the add_middle() method, but it took significantly longer than the add first or add last methods.

Both add methods should be O(N), because the computer is only cycling through the elements once. However, add_middle() is a bit more than O(N), but not O(N^2). This is because the computer still must search through the array at least once. The first time, it is searching to find the index to add to. The second time, it must shift all elements after that index to make room for the value that is being added.

The remove methods should also be O(N). This is because the computer must shift everything at most once. Removing *first* for the Linked_List implementation would take much longer than the array implementation. This is because for the computer to determine which value to remove, it must traverse the entire chain of Nodes, whereas the Array_List implementation can easily remove the value at the last index. I would predict that the remove methods would be faster for the Array_List overall, however remove_first() would be comparable for both.

The following table summarizes the time required (in seconds) to remove 100 elements from the front, or the end, of both implementations:

	Remove First	Remove Last
Linked List	7.350374000000001E-7	2.568262426E-4
Array List	7.294890000000001E-7	8.798798E-7

As predicted, the Array implementation was just barely slightly for remove first, but significantly faster for remove last.

It is worth noting that for the add methods in the Linked_List class, both add_first() and add_last() are almost identical in computation time. Perhaps this is because the add_last() method uses recursion, and so it can traverse the entire list relatively quick.

Landon Crowther
U0926601
2/23/17

Class Hierarchy

The List_2420 class is an interface which provides the methods that need to be implemented.
Both Array_List and Linked_List implement the List_2420 class, and have the same functionality.
The Node class is part of the Linked_List class, and is the object which holds the data. The Node class is super important, as it contains the actual data, as well as the reference to the next Node. This reference is the foundation of the Linked_List.

Iteration vs. Recursion – Size

Iteration involves looping through every element to determine how many there are, whereas recursion is returning 1 + .next() to determine size. Recursion was a great tool for this assignment, and helped increase speed and efficiency of many of the methods.

Software Development

Regression testing is making tests at each level of progress. For example, we would make a test that validated a feature. We then would add on to the program, test new features, but make sure that the old features didn't change. This form of testing was especially useful in this program. For each method, we would write tests that although simple, effectively showed how the structure of the implementation should behave. After modifying a method, the tests were then re-run, and it made it very clear where the code broke. Testing took a very large amount of time for this assignment, but turned out to be very useful. I probably spent 15-18 hours on this assignment.

Thought:

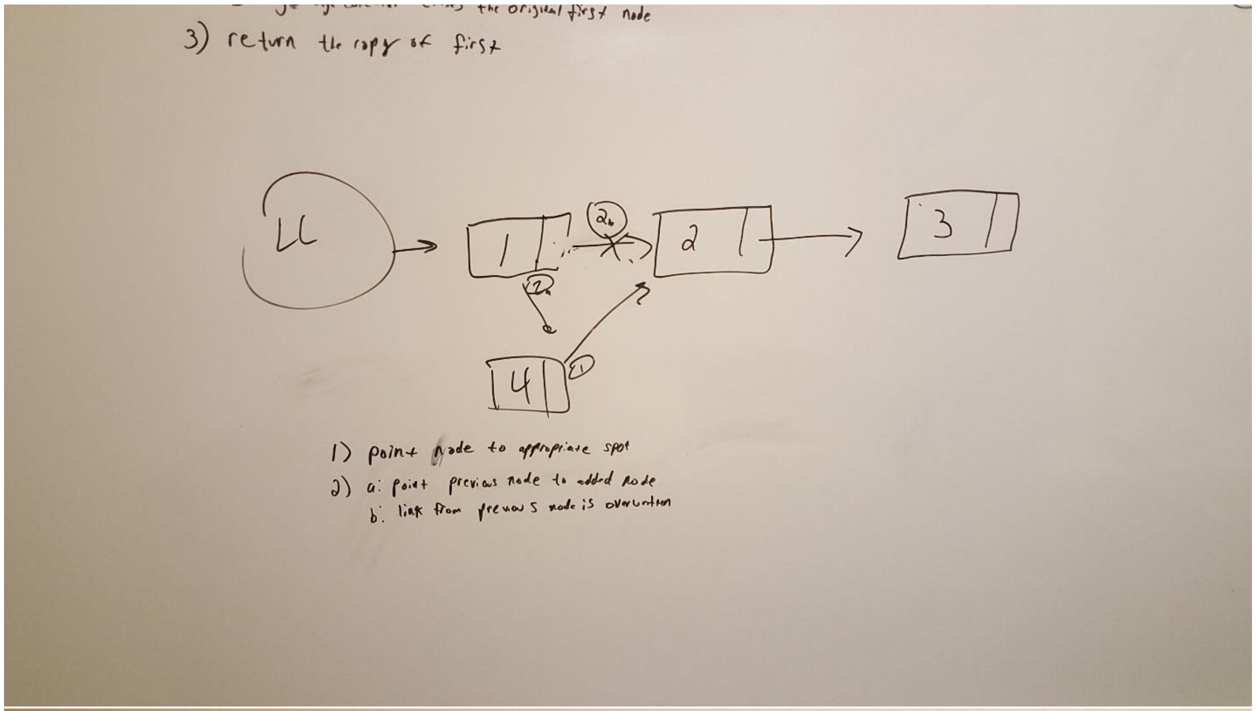
A singly linked data structure wouldn't be good for a que, because to reach an element at the end of a queue, all the previous elements must be released. This would cause a spike in time and demand. To fix this, a "first in last out" approach would be better, where rather than having to remove all elements before it, one could simply take the last element off.

Last Thoughts:

Overall, this assignment was not very exciting. It was very frustrating at parts. I think it would have helped to have a better grasp of the assignment early on. I felt like I didn't really have a lot of information until Tuesday, but by then we only had two days to complete.

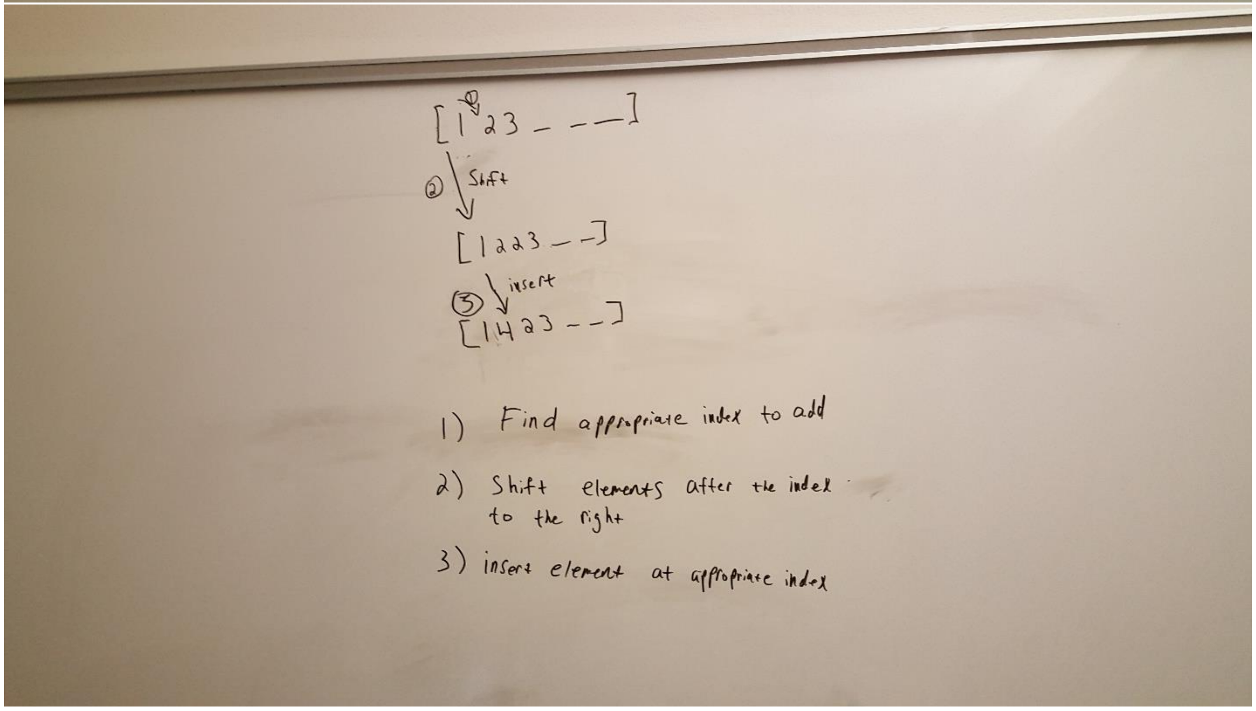
Landon Crowther
U0926601
2/23/17

Pictures:



the original first node
3) return the copy of first

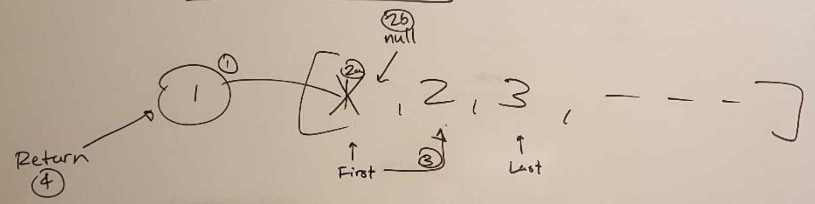
1) point node to appropriate spot
2) a: point previous node to added node
   b: link from previous node is overwritten



1) Find appropriate index to add
2) Shift elements after the index to the right
3) insert element at appropriate index

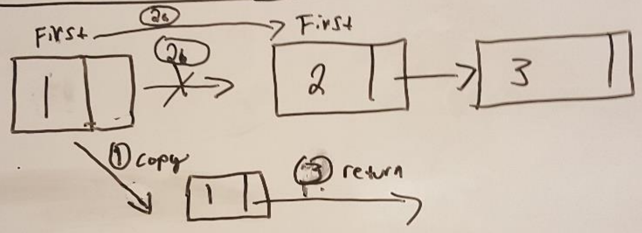
Landon Crowther
U0926601
2/23/17

REMOVE FIRST



1) Save first
2) Set first to null
3) Shift first
4) Return original "first"

Remove first LinkedList



1) create copy of first to return
2) a: assign First to the next node
   b: garbage collector erases the original first node
3) return the copy of first