HW1 - Getting Started

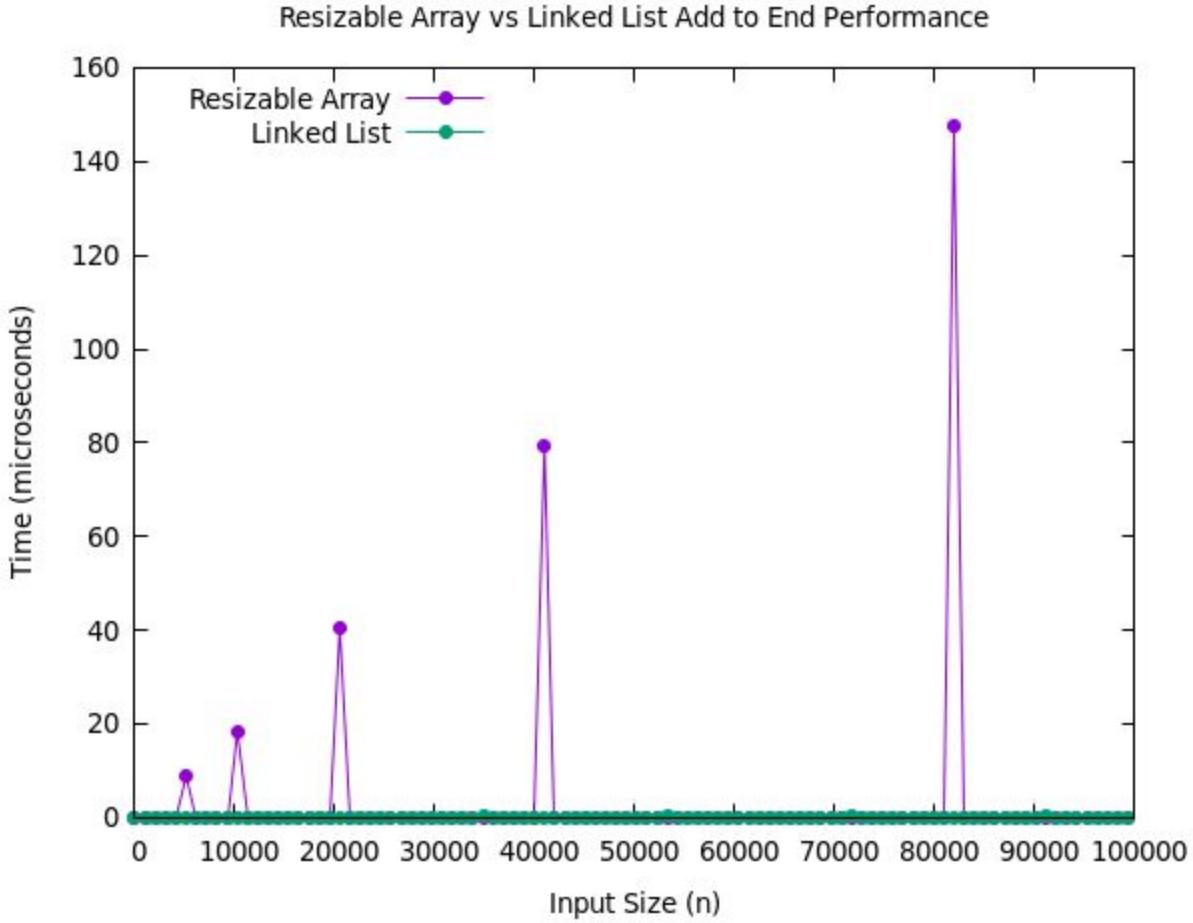
CPSC 223 - Wesley Muehlhausen

Issues and Challenges

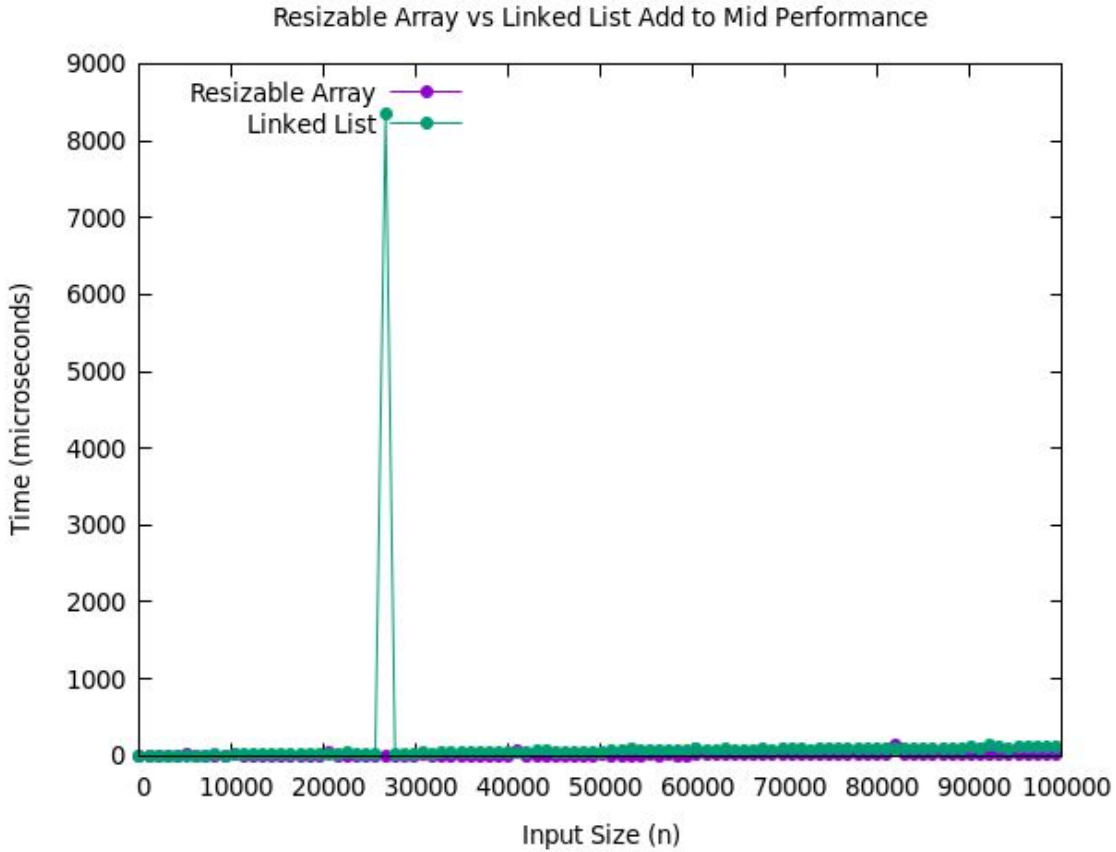
- One of my smaller issues I had in this homework assignment was with the add by index function in the linked list. There were three cases that I made: add to end, add to front, and add to other. Besides those three options there were also other things to think of such as invalid inputs. All of that was not a problem, my huge issue was debugging the parameters of my for loops which traversed the temp pointer though the list. But once I figured that out, my function worked out.

- My huge problem for this project was creating the assignment operator, specifically for the array_list.h file. I understand the way it should work, but I had a really hard time coding it the way I wanted it. This was a bad function to not have in this project, because the helper function “resize()” needed the assignment operator to be working in order to function correctly. Because of this, I could not increase the size of the arraylist even though it was necessary for when the list got filled up. This unfortunately messed up all of my graphs even though all of the other functions in array_list.h were working.

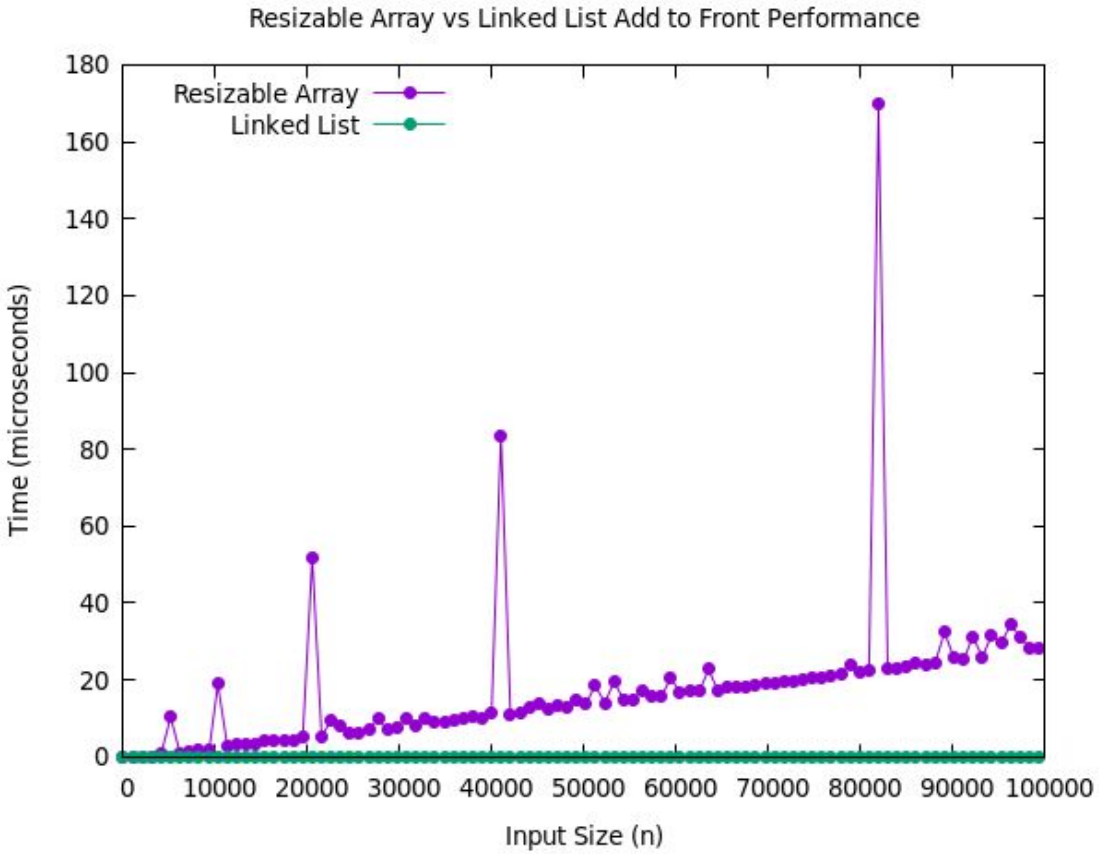
- Another problem was my destructor for the arraylist. At first, all I did was clear the array and not actually delete anything. This was affecting my performance testing because performance testing heavily relies on the destructor.



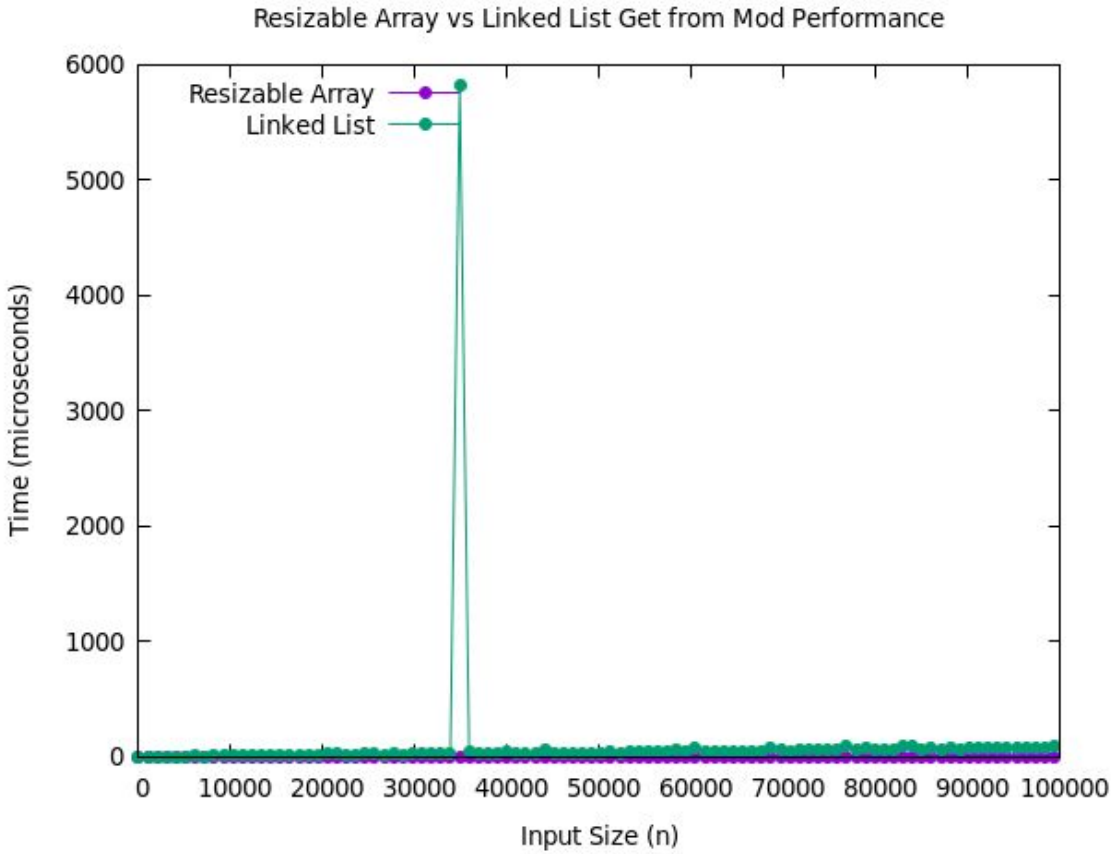
-In this test, both arraylist and linkedlist used add to end which is what was tested. As we can see, both did very well. This is because both of these add functions are O(1) because there is no need for traversal. The graph shows spikes for arraylist. This is probably because of the need to resize when the arraylist gets filled up.



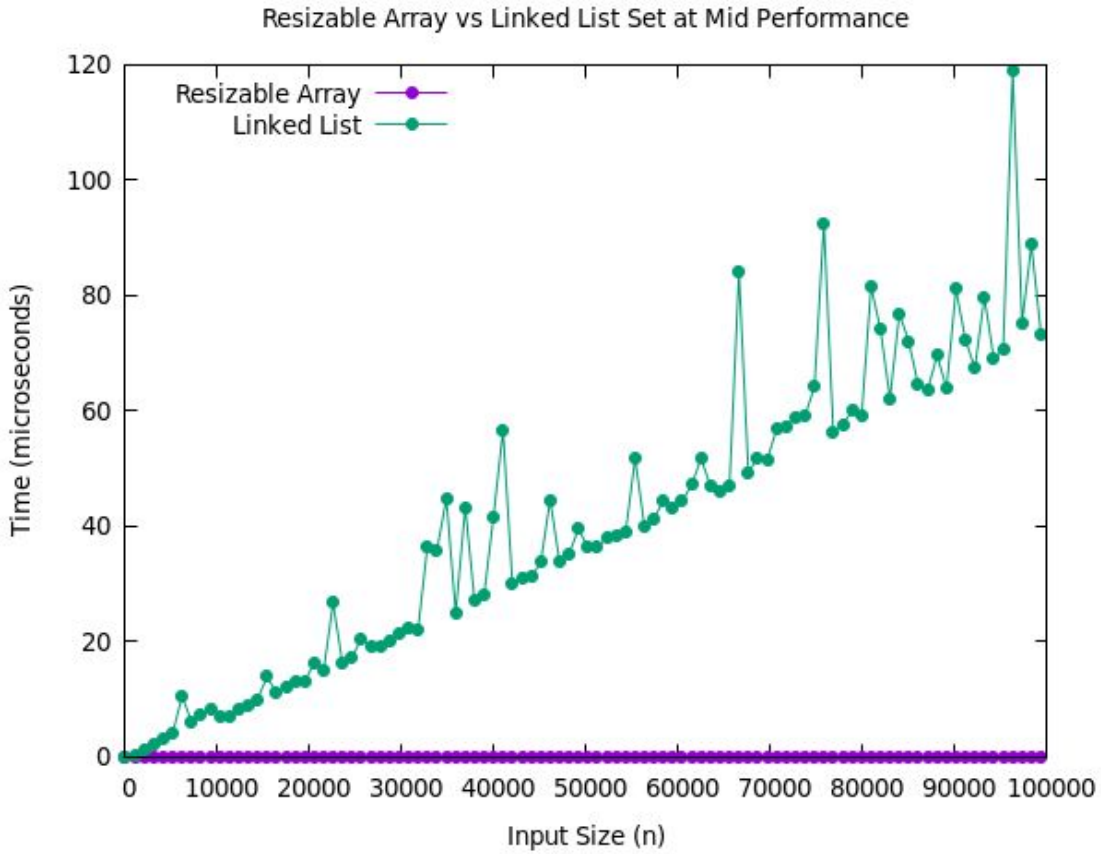
-In this test, both arraylist and linkedlist used add to mid which is what was tested. This would be an add by index scenario for both of them which means that for Linked List, it takes about n/2 moves to get to the middle to do the add, and for arraylist, it takes about n/2 moves to shift the rest of the list over to insert. Even though the graph is hard to read because of the one linked list spike, we can see that Arraylist and linked list do pretty comparable.



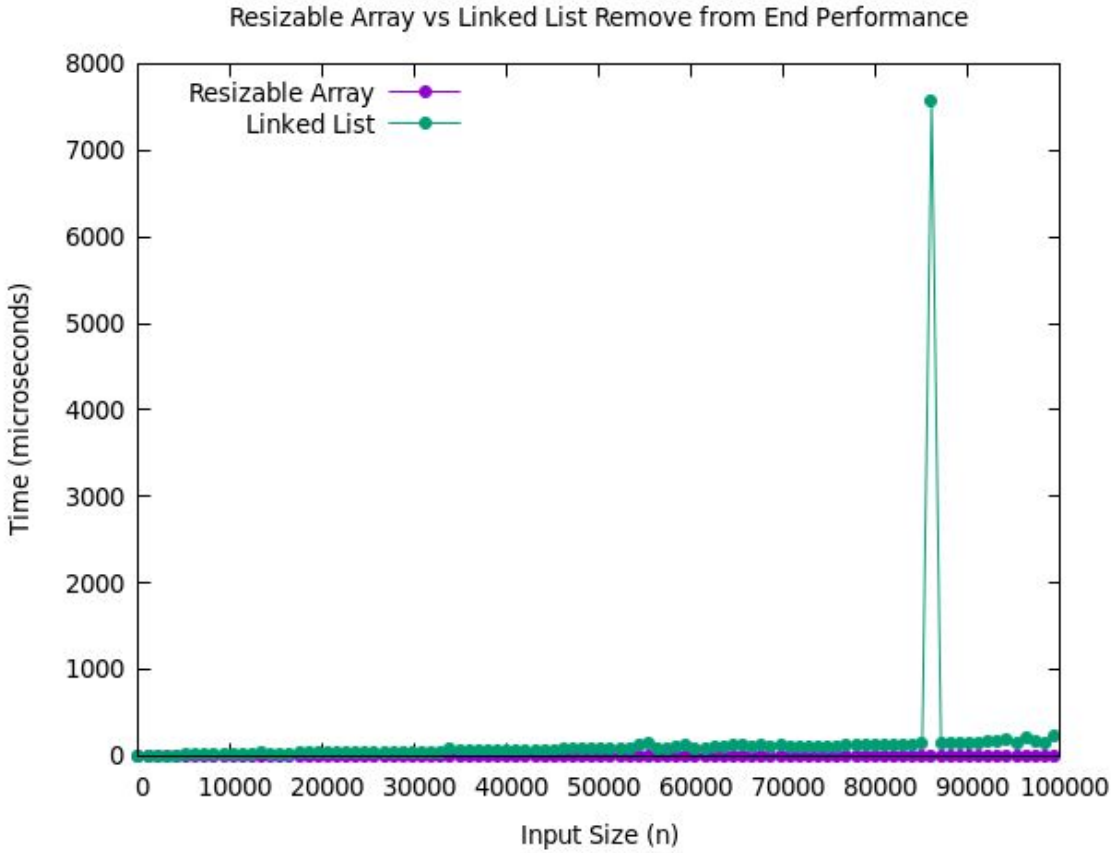
-In this test, we can see that linked list does much better. This is because linked list needs no traversal to add to the front of the list and doesn't have the need to resize. As for arraylist, Every add at the beginning requires the entire list to be shifted over and on top of that it requires resizing which causes the spikes in the graph. So for this, the linkedlist is O(1) and the arraylist is a bad O(n).



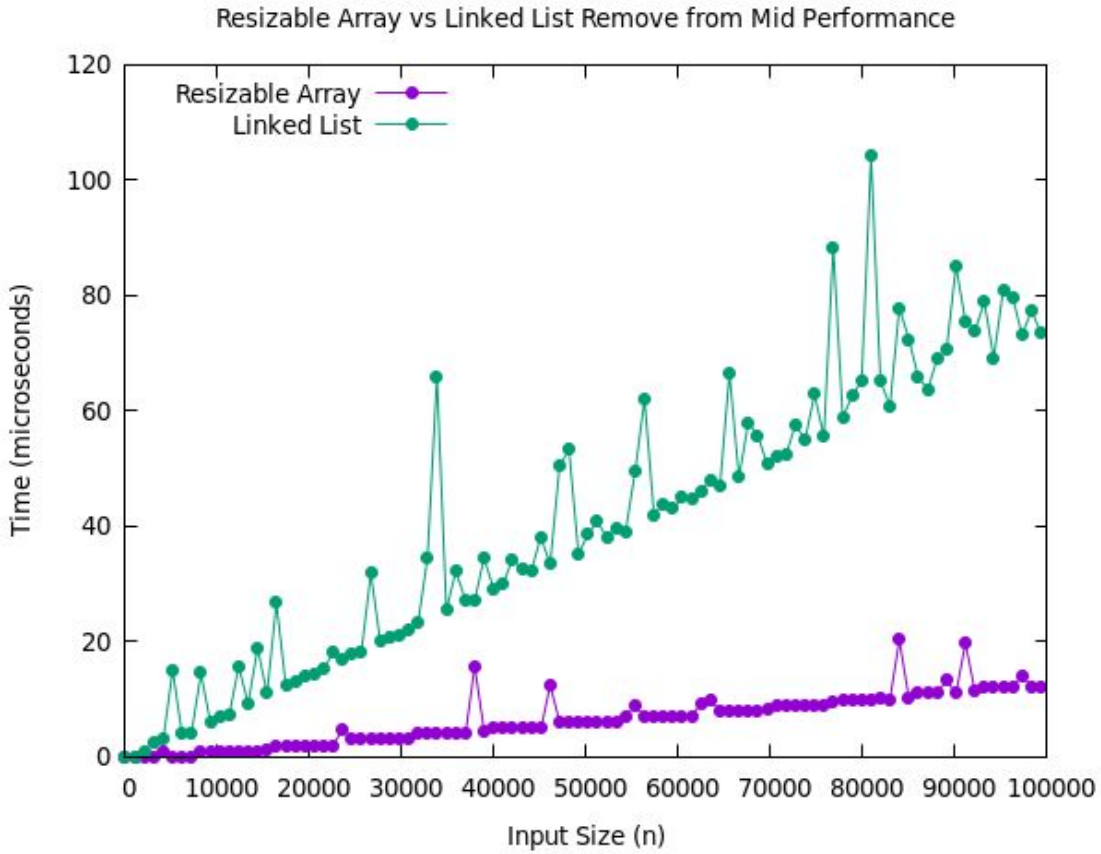
-In this test, get is called at a MID index. Even Though the graph is badly scaled because of the one linked list spike, Arraylist performs much bet rate because it is an O(1) operation to get an item because there is no need to shift. As for the linked list, it performs worse because it has to traverse to the middle of the list to get the item.



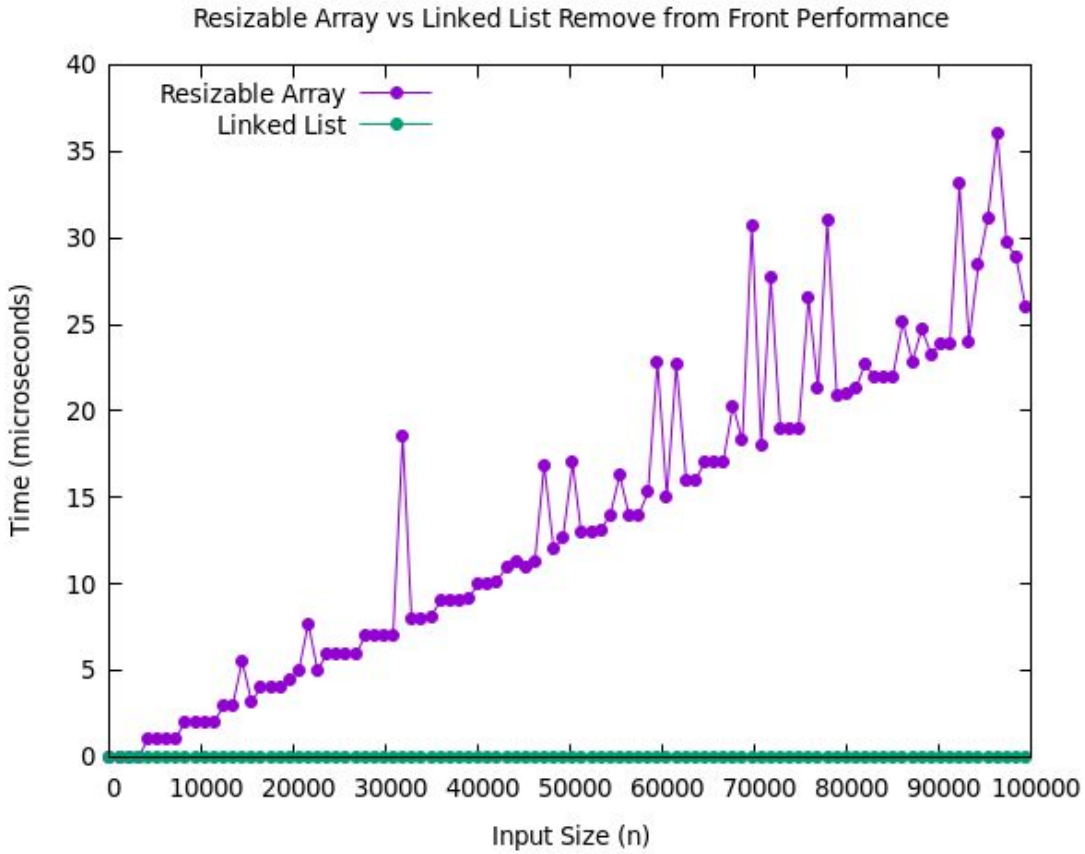
-This test is essentially the same as the previous test. In this test, set is called at a MID index. Arraylist performs much bet rate because it is an O(1) operation to set an item because there is no need to shift. As for the linked list, it performs worse because it has to traverse to the middle of the list to set the item. Also there are no spikes for arraylist because there is no resizing.



-For this, Arraylist does better since removing does not require resizing nor shifting since it is at the end of the list. This means it is an O(1) operation. As for linked list, We need the node behind the tail pointer and because it is not doubly linked, we need to start from the beginning even for a deletion at the end which is why it performs at an O(n) speed.



-Both of the lists perform okay but not great. Linked list has to traverse from the start to the middle which is about n/2 moves, arraylist also has to shift everything to the right of the delete over one which should also be the same number of moves. The linked list could be performing worse because of the more complex and potentially inefficient coding.



-This test makes sense. Linked list remove from front is very efficient because there is no traversal and no shifting. This makes it O(1). As for Arraylist, the list has to shift everything over by one from the start to the end which makes it a O(n) operation.