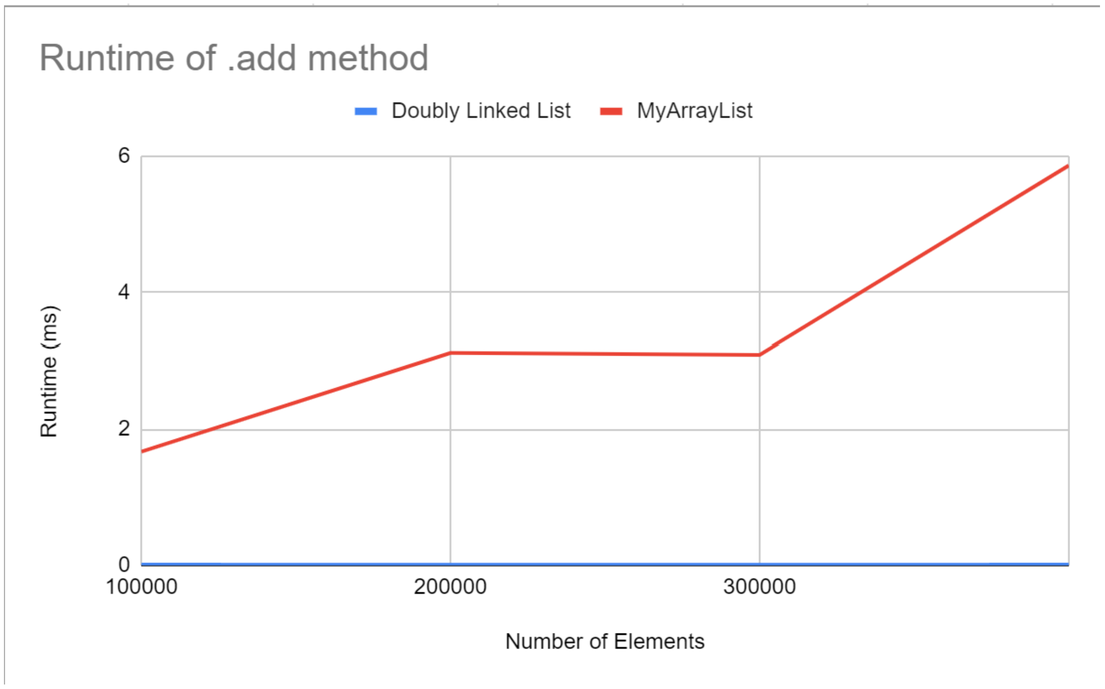
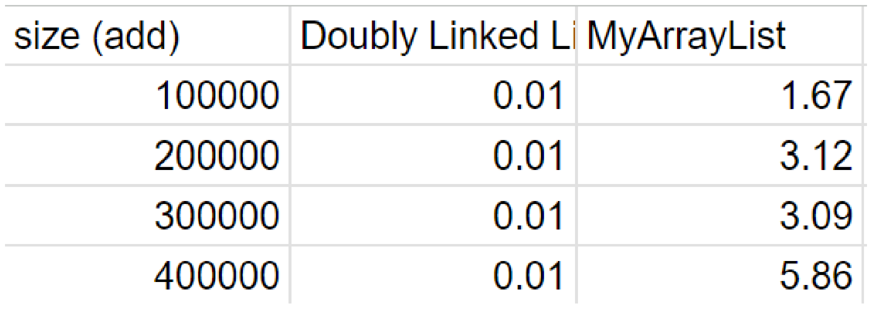
Battle #1: .add method
Winner: Doubly Linked List

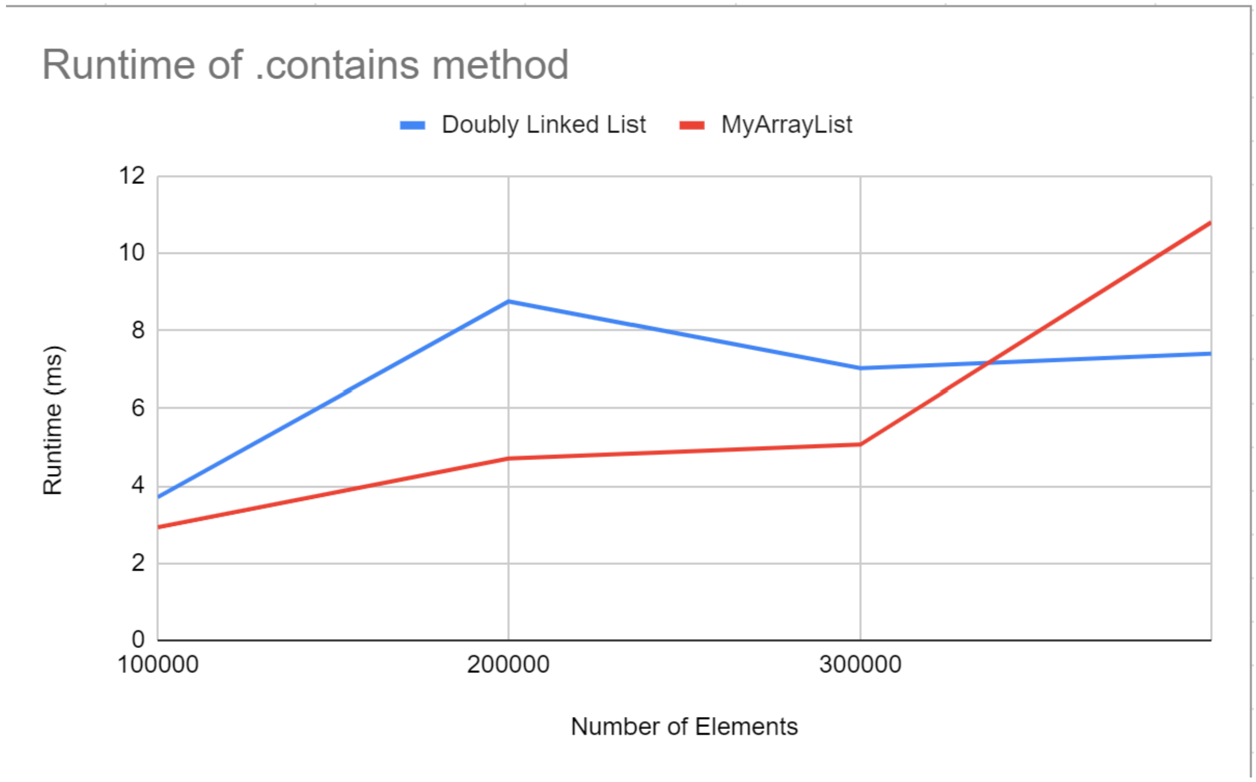


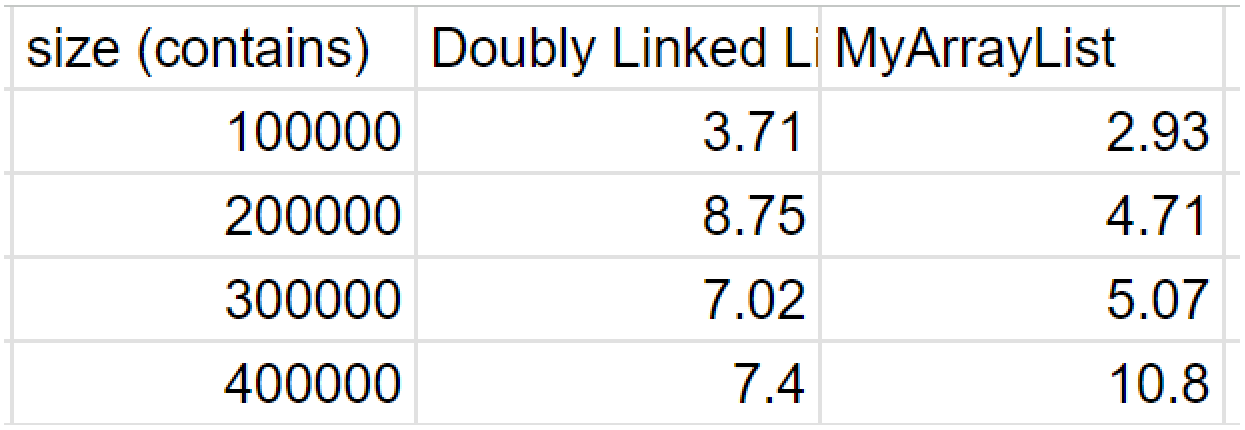


size (add)	Doubly Linked Li	MyArrayList
100000	0.01	1.67
200000	0.01	3.12
300000	0.01	3.09
400000	0.01	5.86

In this battle, the runtime of the .add method increased with the number of elements for the ArrayList, but stayed constantly low for the doubly linked list. This is because in order to add an element to the beginning of the arraylist, all of its elements in the list must be shifted up one index, which is a process with O(n) runtime. Meanwhile, the DoublyLinkedList only has to create a new node and make next and prev references of the head and previous first element respectively point to the new node, which takes O(1) runtime.

Battle #2: .contains method
Tie

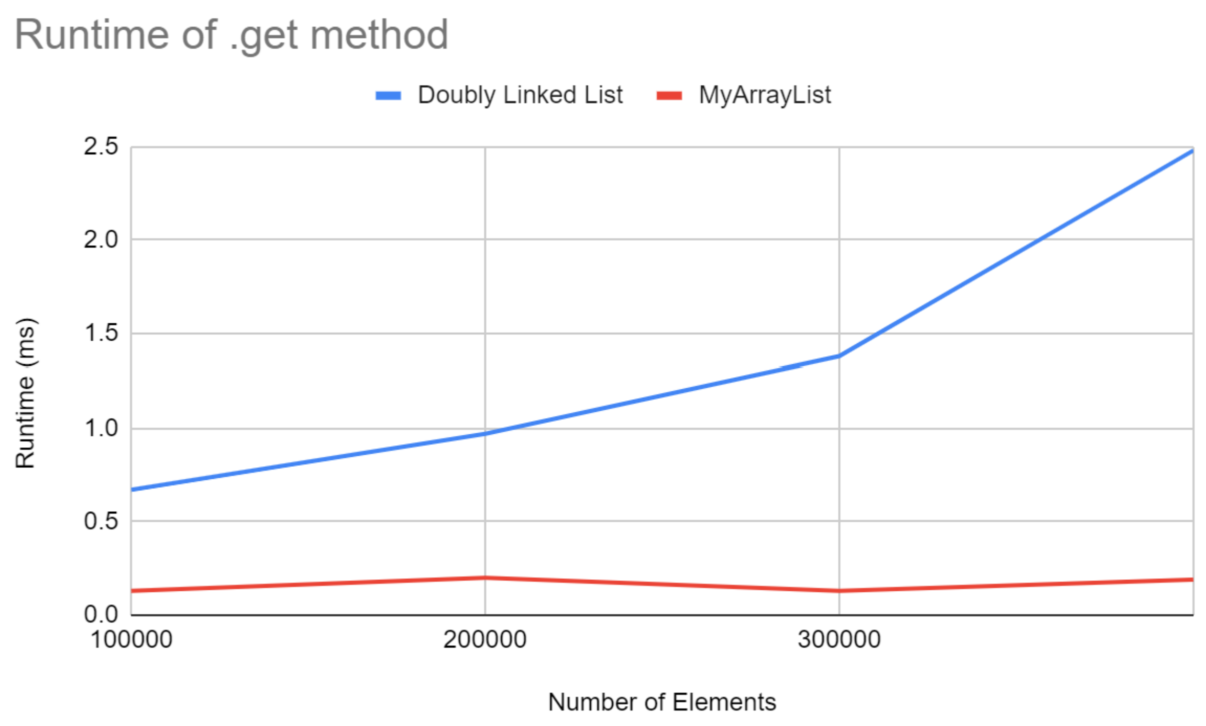


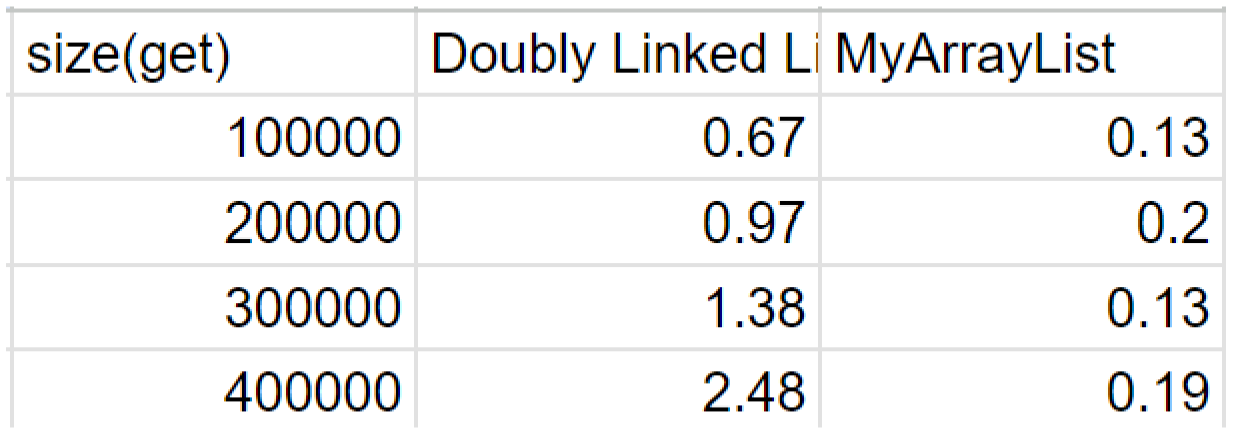


size (contains)	Doubly Linked Li	MyArrayList
100000	3.71	2.93
200000	8.75	4.71
300000	7.02	5.07
400000	7.4	10.8

In this battle, the runtimes for both lists increased with the number of elements added. This is because both .contains functions must iterate through each element in order to search for the given element (which does not exist). Therefore, both functions have O(n) runtime.

Battle #3: .get
Winner: MyArrayList





size(get)	Doubly Linked Li	MyArrayList
100000	0.67	0.13
200000	0.97	0.2
300000	1.38	0.13
400000	2.48	0.19

In this battle, the runtime of the .get method increased with the number of elements for the DoublyLinkedList, but stayed constantly low for the array list. This is because the DoublyLinked List must traverse through the list in order to get to the provided index, while the ArrayList can simply access the element from its array instance variable.