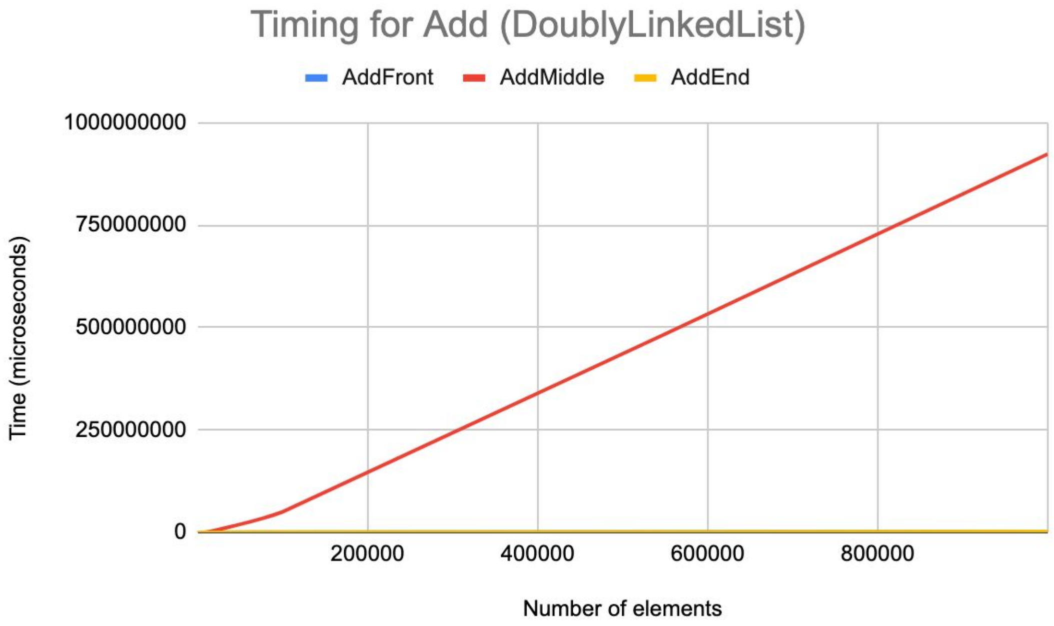
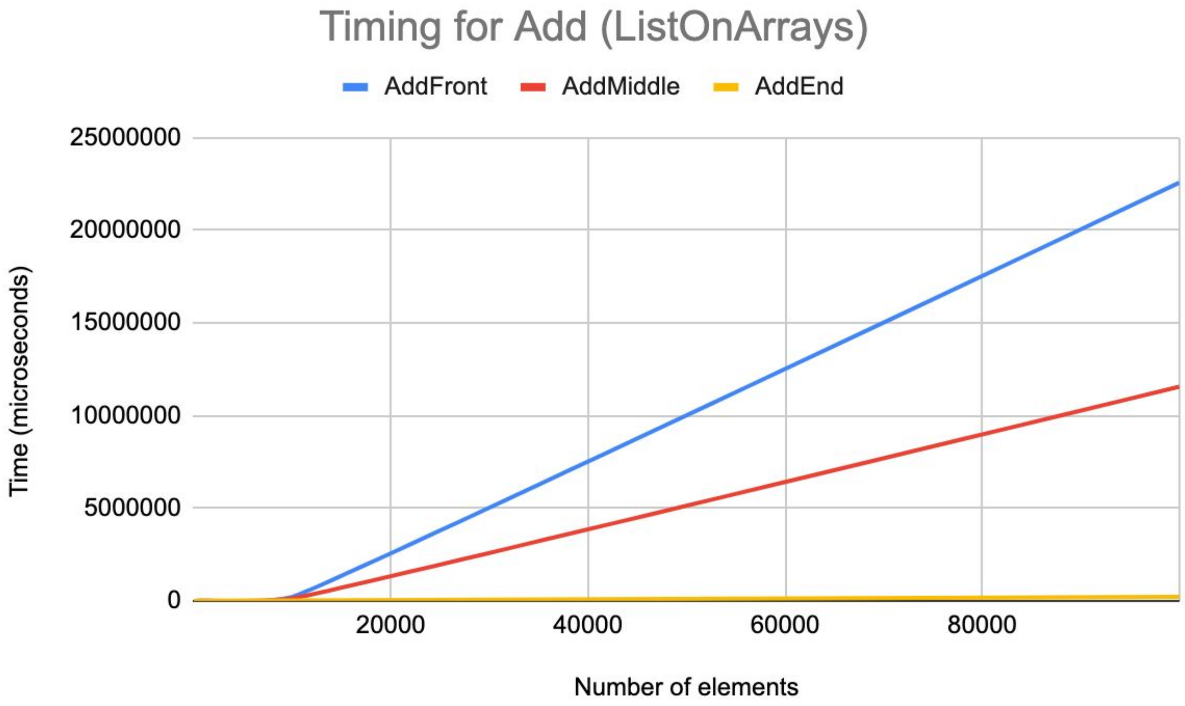
Aaron Morgan and Alejandro Serrano
Prof. Swaroop Joshi
Assignment 06
Wed. 10/23/2019

Analysis on DoublyLinkedList

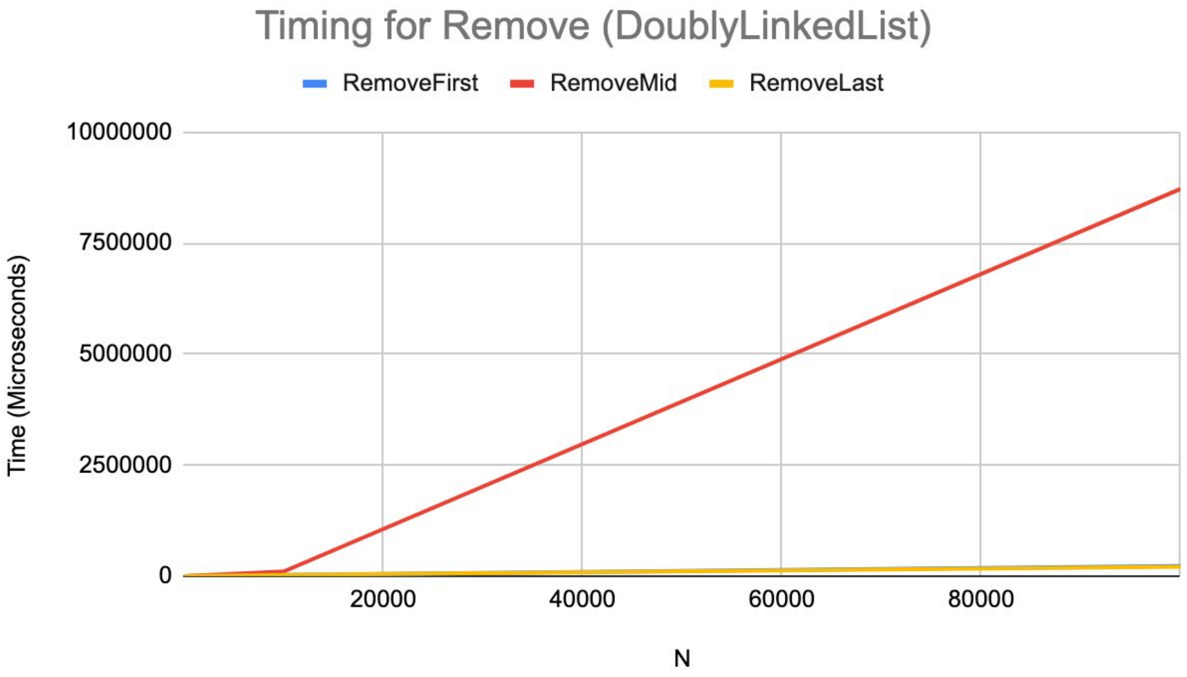
Adding To A6DoublyLinked List			
N	AddFront	AddMiddle	AddEnd
10	110	92	80
100	906	3502	574
1000	6402	5662	3362
10000	35067	215585	24557
100000	230522	49855140	216549
1000000	1997887	923434964	1919176



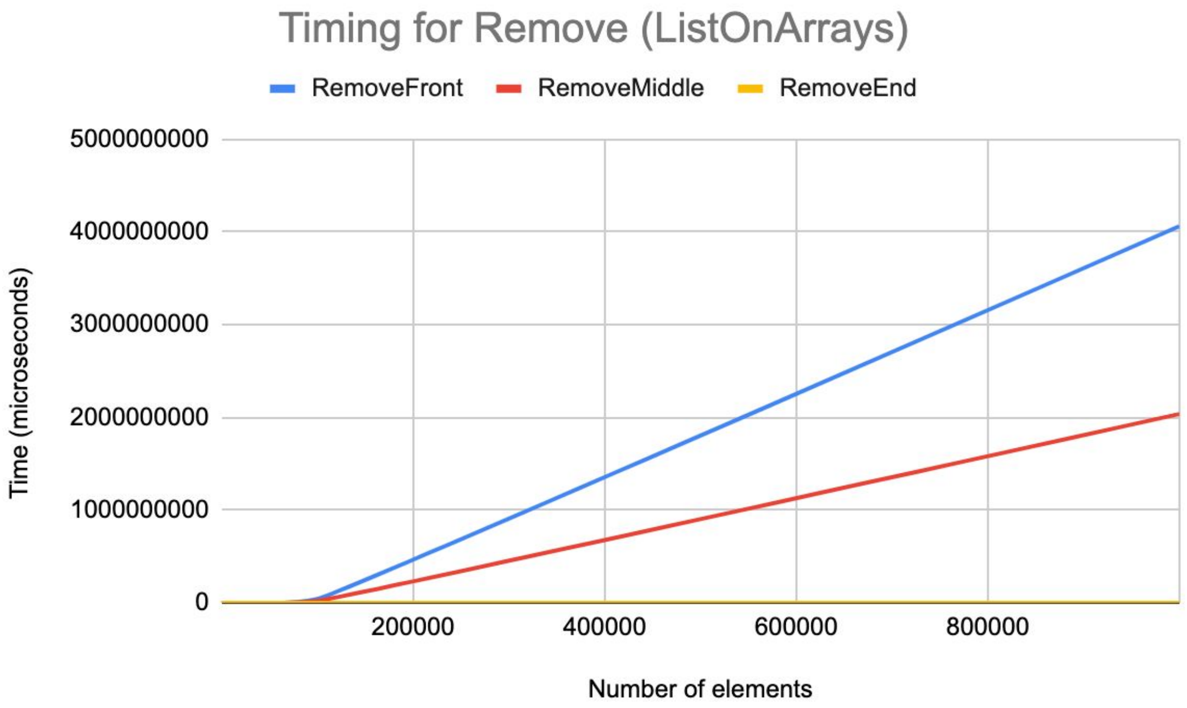
Add on ListOnArrays			
N	AddFront	AddMiddle	AddEnd
10	65	62	77
100	655	564	492
1000	9894	7835	3983
10000	212555	131758	28787
100000	22573734	11560339	220287



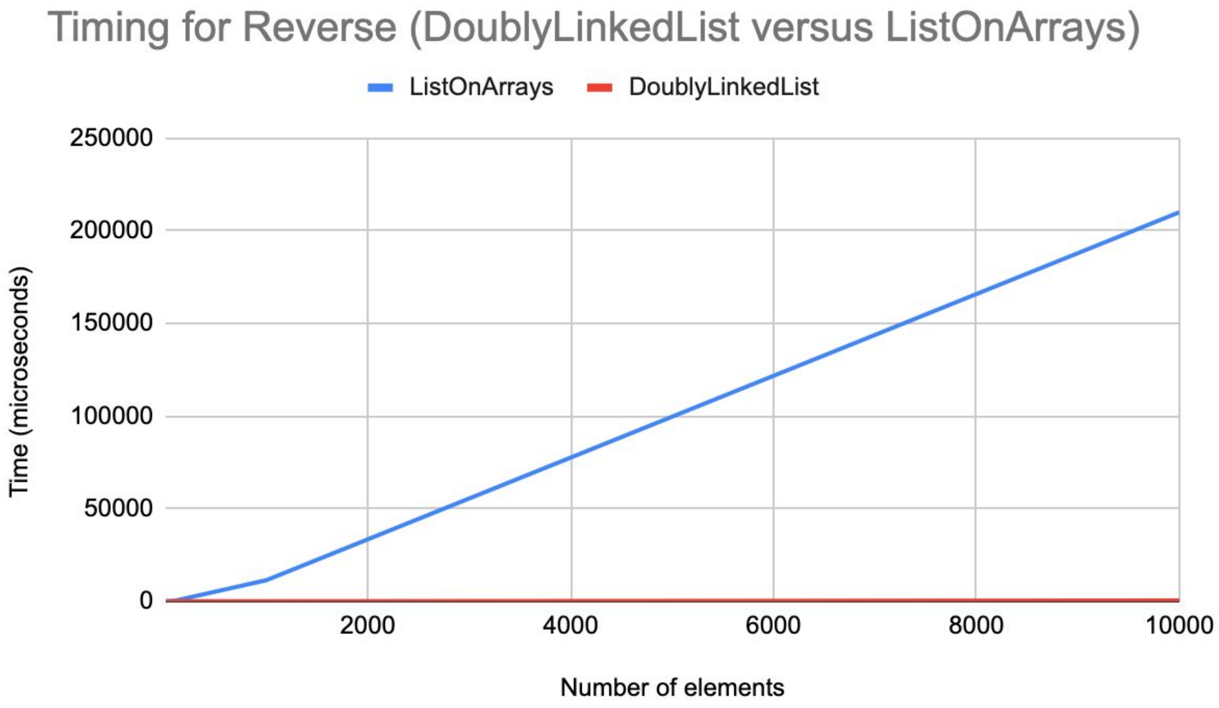
Removing from A6DoublyLinkedList			
N	RemoveFirst	RemoveMid	RemoveLast
10	71	76	68
100	410	482	445
1000	5172	7374	4847
10000	27334	101549	27852
100000	225093	8723155	210339



Removing from ListOnArrays			
N	RemoveFront	RemoveMiddle	RemoveEnd
10	2	10	1
100	85	55	4
1000	1438	592	40
10000	90158	46969	403
100000	40519946	20193842	1439
1000000	4062623192	2035958828	4136



Reverse: ListOnArrays Vs DoublyLinkedList		
N	ListOnArrays	DoublyLinkedList
10	48	20
100	355	28
1000	11425	90
10000	210027	540



The reason why it is less costly to add values to the front and end of a DoublyLinkedList is because we have direct access to the front and end of the list. For the middle, it potentially has to traverse the entire list to add.

For ListOnArrays, it seems to perform the fastest when adding to the end of the list. It's the slowest when adding at the start of the list because it has to move all the values to the next position. The performance of the middle is in between because of the index, since it potentially only has to move the values starting from the index.

The performance of remove for DoublyLinkedList is similar to the performance of adding, because the code is checking whether the index is at the beginning or end of the list. Therefore, keeping both operations constant. Nonetheless, middle still performs quite slow since it needs to traverse part of the list to reach the desired index. This is the reason why we did not go past 100 000 elements. With 1 000 000 elements the program was taking too long to complete.

The performance of remove for ListOnArrays is identical to the performance of the add due to the same reasoning; it needs to shift values everytime it removes a value both at the start and the middle.

The assignment took us around 15 - 20 hours to complete. The most time consuming was understanding the concept of how DoublyLinkedLists worked. Once we correctly grasped the idea, then the add(int index, E val) took the longest to implement. One thing we could do better in the future is to test the performance of more sizes. We planned each session with tasks we needed to complete, and we allocated enough time by starting as soon the assignment was available.