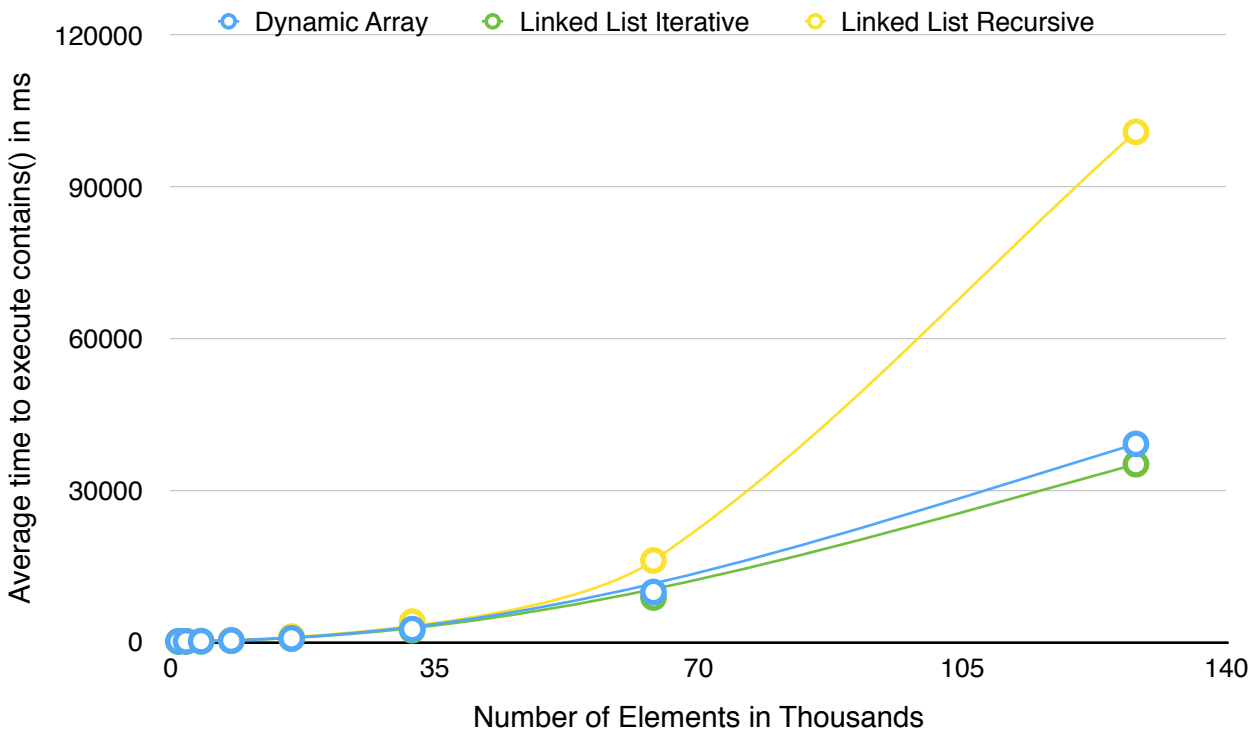
Bret Lorimore & Preston Wipf
Dr. Metoyer
CS261
Assignment3 - Q2

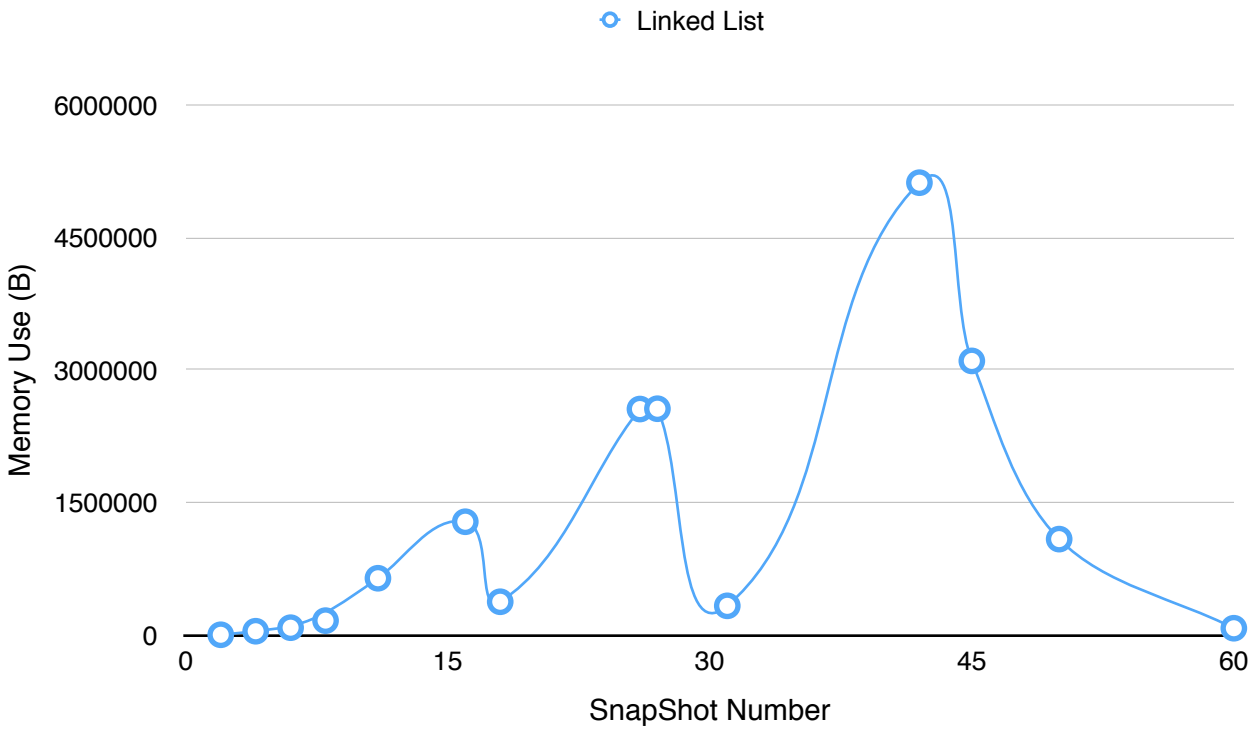
Execution timing data for contains() on a dynamic array and linked list (2 trials each)



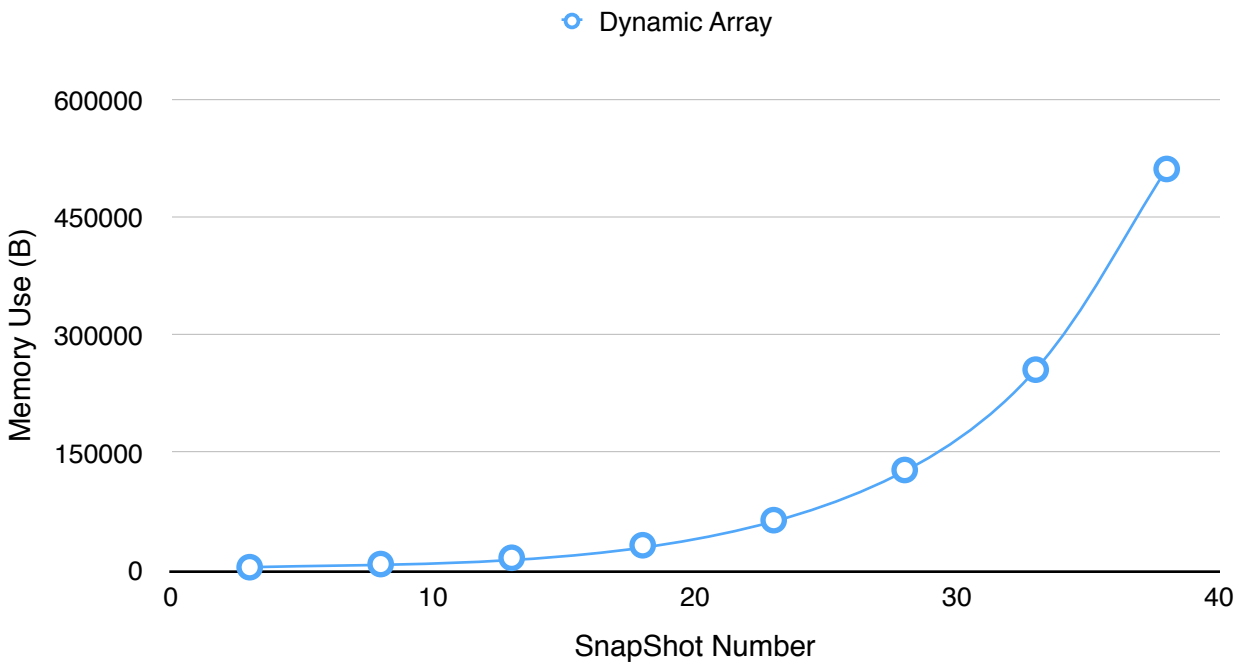
Contains on # elements	Dynamic Array (ms)	Linked List Iterative containsList() (ms)	Linked List Recursive containsList() (ms)
1000	0	0	0
2000	10	10	10
4000	40	30	50
8000	150	140	240
16000	610	550	960
32000	2440	2200	3910
64000	9780	8680	16020
128000	39100	35080	100880

Note: I did both recursive and iterative implementations of containsList() - recursive call is commented out in turned in code

Memory usage data for contains() on a dynamic array and linked list



SnapShot #	Linked List Memory Total(B)
2	0
4	40120
6	80120
8	160120
11	640120
16	1280120
18	373920
26	2556480
27	2560120
31	327760
42	5120120
45	3101680
50	1083280
60	74080



SnapShot #	Dynamic Array Memory Total(B)
3	4032
8	8032
13	16032
18	32032
23	64032
28	128032
33	256032
38	512032

Answers to Questions:

- **Which of the implementations uses more memory? Explain why.**
    - The linked list implementation used significantly more memory - roughly 10x the memory used by the dynamic array implementation. It is expected that the linked list should use more memory, as the array only has to store the memory for each value, and one pointer to the chunk of memory, whereas with the doubly linked list, there are two pointers stored for each value stored, plus two additional links for the whole list (the sentinels).
- **Which of the implementations is the fastest? Explain why.**
    - The dynamic array and (iterative) linked list had similar performance, as in each case they are just iterating through a list (with the list, the are essentially playing “follow the pointer chain” whereas with the array, pointer arithmetic is being used to navigate through the array. The iterative linked list was overall slightly faster than the dynamic array. The recursive linked list was much much slower, as is often the case with recursive implementations because they have to store all their suspended calls in the stack and then come back to them.
- **Would you expect anything to change if the loop performed remove() instead of contains()? If so, what? (Note, it's very easy to run this experiment given the code we've provided!)**
    - I would expect there to be a tiny - nearly insignificant performance hit for our linked list code when comparing the iterative contains() function with the removeList() function, as it has a nearly identical iterative implementation. However in the code we were provided (the dynamic array) the performance would likely be cut in half, as it checks if the array contains the element before going to remove it. Our implementation checks and removes in one - if it does not find the element it just doesn't remove anything and prints an error. Now I am going to check my hypothesis, see data on the following page.

Remove on # elements	Dynamic Array Trial 1 (ms)	Linked List(ms)
1000	0	0
2000	10	10
4000	30	30
8000	120	130
16000	480	540
32000	1920	2120
64000	7700	8560
128000	30760	34300

The Results above are very interesting. As expected, the execution time for our linked list implementation is nearly identical to the execution for the iterative contains() function. However, the execution time for the dynamic array is significantly lower when running removeDynArr() vs containsDynArr(). This may be because each time an element is removed, the list gets smaller and thus there are fewer elements to check. However if this were the case, one would think that removeList() would see a similar performance improvement over containsList().