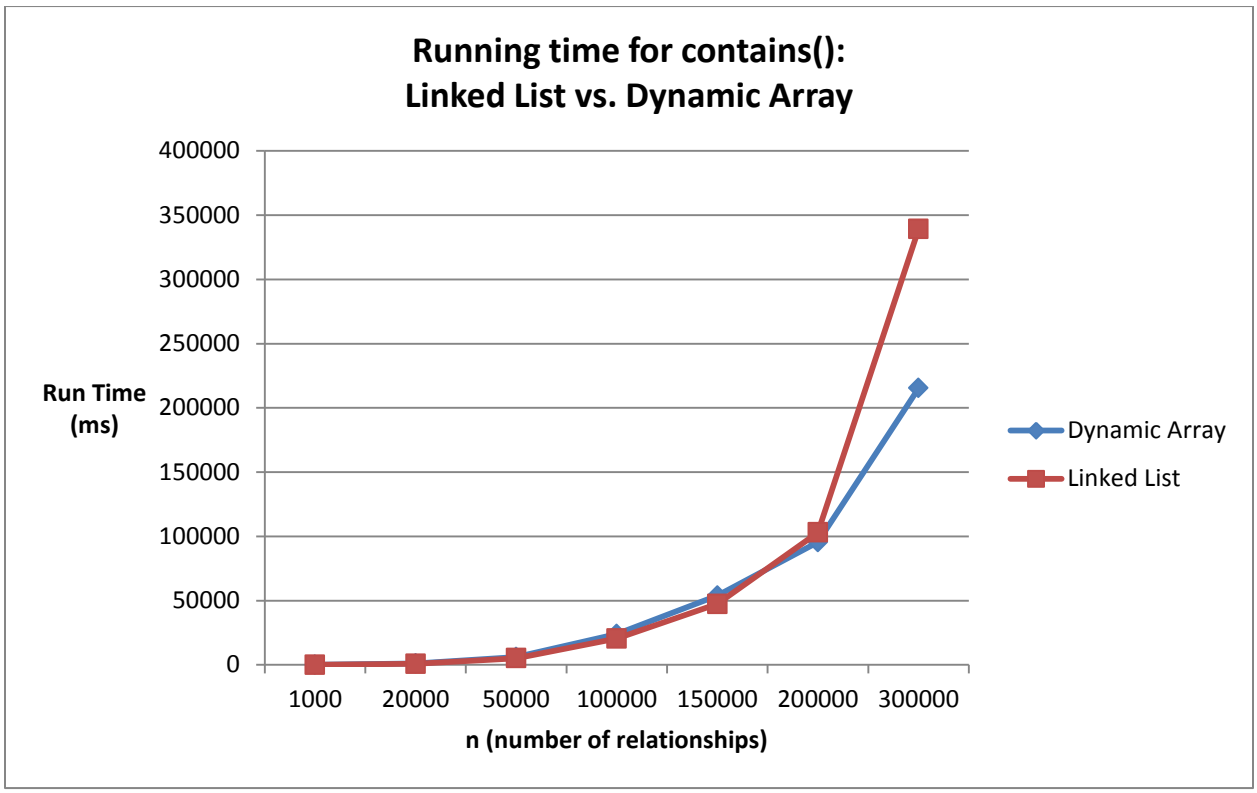
Kamal Chaya, Cam Tu Vo
CS261-Assignment3 Problem2
10.23.2013

Runtime for contains() (ms)

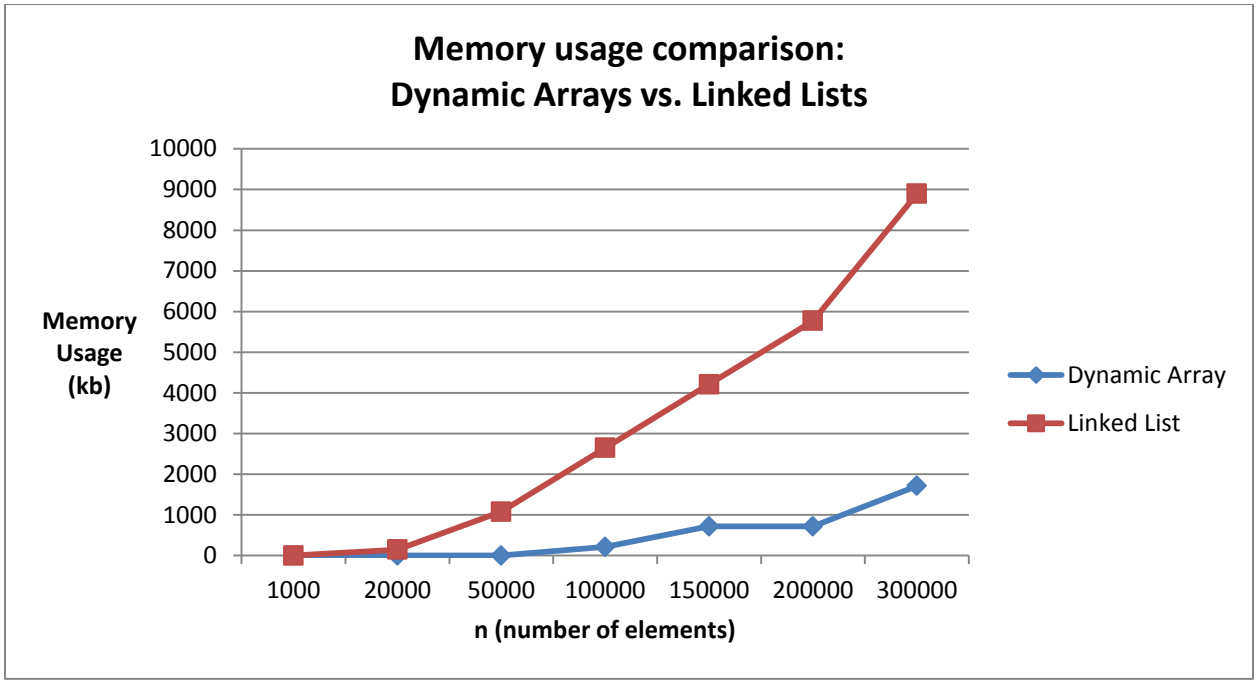
Number of elements (n)	Dynamic Array	Linked List
1000	0	0
20000	950	790
50000	5960	5060
100000	23870	20380
150000	53800	47320
200000	95460	103030
300000	215320	339170



Kamal Chaya, Cam Tu Vo
CS261-Assignment3 Problem2
10.23.2013

Memory Usage (kb)

n (number of elements)	Dynamic Array	Linked List
1000	0	0
20000	0	144
50000	0	1080
100000	212	2648
150000	716	4208
200000	716	5772
300000	1712	8896



Kamal Chaya, Cam Tu Vo
CS261-Assignment3 Problem2
10.23.2013

Which of the implementations uses more memory? Explain why.

The linked list implementation will use more memory, because each node of a linked list must store a value, in addition to a pointer to the next and previous node (assuming it is doubly linked like in this implementation), while only the value must be stored in a dynamic array. Of course, there is the argument that since the dynamic array in this implementation doubles its capacity every time it is full, that this unused memory may mean that it uses more than the linked list. The fact that each node in a linked list has to store three pieces of information (as opposed to the one piece in the dynamic array) equates to the linked list using up more memory.

Which of the implementations is the fastest? Explain why.

The dynamic array implementation is faster. This is because searching a dynamic array is a constant time operation (O(1)), while searching a linked list is a linear time operation (O(n)).

Would you expect anything to change if the loop performed remove() instead of contains()? If so, what?

I believe that the linked list would be a little faster than the dynamic array. This is because to remove elements from a dynamic array, after finding the specified element, it must then shift all of the other elements from that index up to n backwards by one index. In the worst case scenario, this would be a O(n) operation. For a linked list, to remove a node all we have to do is rearrange the pointers accordingly, and then deallocate the node that is not used anymore, which is a constant time operation O(1) in addition to the O(n) run time it takes to find the value to remove. However, the net effect is that both run times would really just be O(n), and the linked list would be slightly faster if you measured the times very precisely.