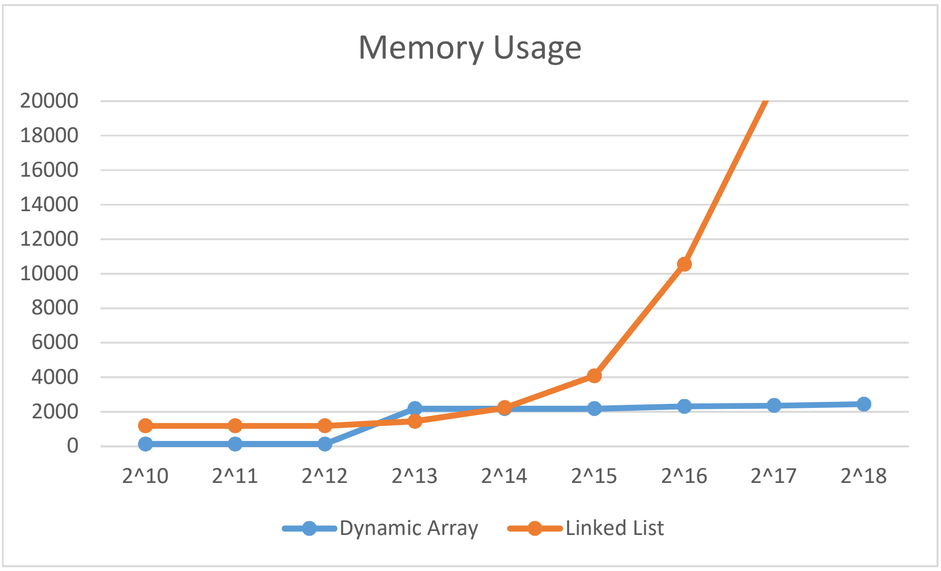
Peter Moldenhauer
1/26/17
CS 261 – Data Structures
Assignment 3 – Part 2

Linked List vs. Dynamic Array performance comparison

Memory usage – linked list vs. dynamic array for n from n = 2^10 to n = 2^18

Data collected for Dynamic Array	Data collected for Linked List
2^10 (1024) - 124 KB	2^10 (1024) - 1184 KB
2^11 (2048) - 124 KB	2^11 (2048) - 1184 KB
2^12 (4096) - 124 KB	2^12 (4096) - 1184 KB
2^13 (8192) - 2172 KB	2^13 (8192) - 1440 KB
2^14 (16384) - 2172 KB	2^14 (16384) - 2232 KB
2^15 (32768) - 2172 KB	2^15 (32768) - 4080 KB
2^16 (65536) - 2308 KB	2^16 (65536) - 10544 KB
2^17 (131072) - 2356 KB	2^17 (131072) - 20968 KB
2^18 (262144) - 2436 KB	2^18 (262144) - 41552 KB

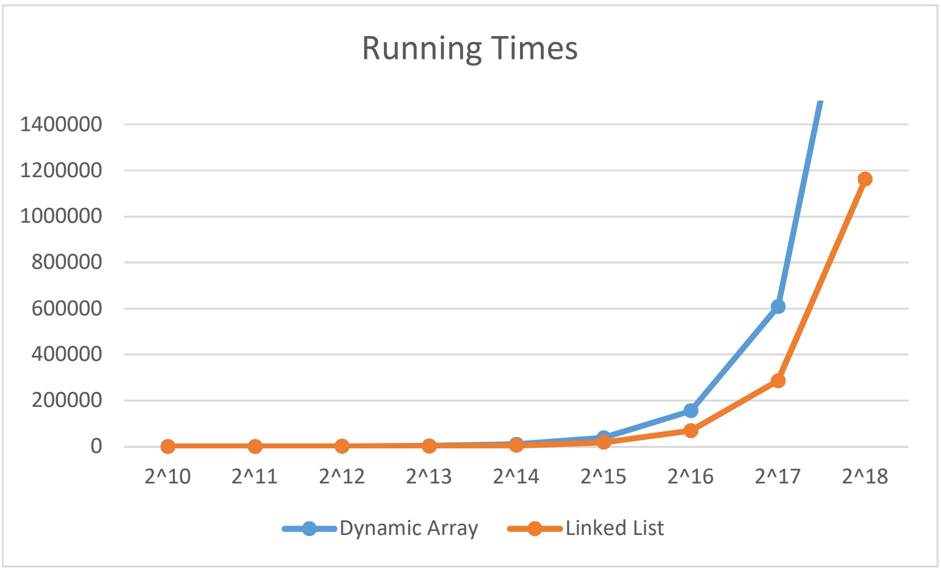


Element counts on X-axis
Memory in KB on Y-axis

Note: Due to the extreme range in numbers, I attempted to design the graph that included the majority of data points yet still shows the difference with the smaller numbers.

<u>Running time – linked list vs. dynamic array for n from n = 2^10 to n = 2^18</u>

Data collected for Dynamic Array	Data collected for Linked List
2^10 (1024) - 80 ms	2^10 (1024) - 40 ms
2^11 (2048) - 160 ms	2^11 (2048) - 140 ms
2^12 (4096) - 610 ms	2^12 (4096) - 330 ms
2^13 (8192) - 2400 ms	2^13 (8192) - 1260 ms
2^14 (16384) - 9660 ms	2^14 (16384) - 4370 ms
2^15 (32768) - 38040 ms	2^15 (32768) - 18270 ms
2^16 (65536) - 154940 ms	2^16 (65536) - 69030 ms
2^17 (131072) - 607960 ms	2^17 (131072) - 286010 ms
2^18 (262144) - 2.44426e+06 ms	2^18 (262144) - 1.1619e+06 ms



Element counts on X-axis
Time in milliseconds on Y-axis

Note: Due to the extreme range in numbers, it is hard to see the change of the lines with the small number of element counts but as the elements grow larger, the change in the graph becomes more clear.

1) Which of the implementations uses more memory? Explain why.

The linked list used more memory. This is because linked lists have poor locality. The memory that is used for the linked list is scattered around. The linked list has to allocate more memory for each piece of data and these memory locations are not contiguous addresses in memory (as they are with the dynamic array).

2) Which of the implementations is the fastest? Explain why.

The linked list is the fastest. This is because linked lists have fast O(1) insertion at any position in the list and the list does not have to resize as it does with the dynamic array. Insertion in the linked list is only breaking the list, inserting a new element, and repairing the list back together. Unlike the dynamic array which has slow O(n) performance because it has to resize and copy over all of the array elements into a larger array when the initial array size reaches capacity.

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

I would expect that the overall running times for both the dynamic array and linked list would be less (faster). This is because the contains() function has a speed of O(n) while the remove() function has a speed of O(1). Therefore, going from a O(n) operation to a O(1) operation would increase the speeds of both. It should be noted that even though the overall speeds of both dynamic array and linked list would be less, the linked list would still be faster than the dynamic array. This is due to the fact that the linked list still has O(1) insertion speed and does not have to resize and copy any elements into a larger container (O(n)) as the dynamic array does.