



1. Runs an outer loop from 1000 up to 128,000, doubling the loop variable,n, at each iteration

2. In each iteration, it performs n calls to add() followed by n calls to contains()

3. Prints the time taken by these contains() calls in milliseconds

1. The dynamic array implementation uses more memory because each time it hits capacity and performs another push it must allocate a fresh array of twice the current array’s size, which causes some memory inefficiency because it is not required to fill in the entire array. The linked list implementation allocates a single node for each new piece of information and thus does not waste memory.

2. The linked list implementation appears to be slightly quicker as n grows arbitrarily large. This is likely because of the time cost of the dynamic array having to copy over an array to a larger array when size is equal to capacity and another element is added. This added cost to the add() function renders the dynamic array more inefficient. Since the contains() function searches through the entire array in both cases, the dynamic array’s advantage of immediate element access is rendered unimportant.

3. After changing from contains() to remove(), let’s compare some graphs:



It appears the linked list structure offers a slower remove() approach as n gets larger (the differences when n is between 0 and 60000 is negligible). Both are O(n) which makes intuitive sense considering they are both erasing from the first element onwards. I assume the dynamic array’s advantage as n grows extremely large is its immediate access to indexed elements vs. the linked list’s mandatory traversal.



This one’s a weird one. The linked list’s graph is rather erratic, increasing and decreasing sharply, and then providing the same memory efficiency as the dynamic array as n grows larger. My theory is that since the dynamic array’s contiguously allocated chunk of memory does not change over the course of the program, its graph is a bit boring (note: there is no new allocation using the remove function of the dynamic array, rather each element is simply shifted backwards). I suppose the linked list’s behavior is due to the linking up of an element’s previous and next items before it is removed as well as the memory processes involved in traversal. The big difference between these two structures is that the linked list is cleaning up after itself by deleting nodes but the dynamic array’s contiguous chunk does not change- while there is no value-possessed memory at certain elements, those elements are still technically allocated to be part of the array.