| linkedList | | | | dynamic Array | | |
|---|---|---|---|---|---|---|
| N | Memory Usage | Time | | N | Memory Usage | Time |
| 1000 | 0 KB | 0 ms | | 1000 | 0 KB | 0 ms |
| 2000 | 0 KB | 0 ms | | 2000 | 0 KB | 0 ms |
| 4000 | 0 KB | 30 ms | | 4000 | 0 KB | 0 ms |
| 8000 | 0 KB | 120 ms | | 8000 | 0 KB | 0 ms |
| 16000 | 20 KB | 510 ms | | 16000 | 0 KB | 0 ms |
| 32000 | 520 KB | 2040 ms | | 32000 | 0 KB | 0 ms |
| 64000 | 1520 KB | 8210 ms | | 64000 | 0 KB | 0 ms |
| 128000 | 3516 KB | 33340 ms | | 128000 | 0 KB | 0 ms |
| 256000 | 7516 KB | 233050 ms | | 256000 | 0 KB | 0 ms |

Clearly the linked list implementation used the most memory of the two. This is because in addition to the structure holding data, it also holds a pointer to the next link in the list, thereby increasing the amount to memory needed. Similarly, the dynamic array is the faster of the two structures. I'd guess the reason for this is that there is only need to use pointer math to access a given part of the array, where with the linked list, each data element has its own pointer. If we were to use contains() instead of remove(), I'd expect the dynamic array's time to increase as you need to copy the array each time you remove an element from the array. The linked list you simply have to change the assigned pointers to the previous and next link.