Results for ArrayList:

Number of Elements	Add (ms)	Sort (ms)	Shuffle (ms)	Random Get (ms)	Sequential Get (ms)
1,000	2	2	2	23	1
5,000	8	13	5	166	5
10,000	18	24	12	340	10

Results for LinkedList:

Number of Elements	Add (ms)	Sort (ms)	Shuffle (ms)	Random Get (ms)	Sequential Get (ms)
1,000	2	8	2	224	9
5,000	16	93	24	1975	118

10,000	32	232	58	3942	254

In the tables above, we can see that adding elements to the ArrayList and LinkedList is very fast, even for 10,000 elements. Sorting the ArrayList is generally faster than sorting the LinkedList, although the difference is not very significant. Shuffling the ArrayList is generally faster than shuffling the LinkedList, although again the difference is not very significant. Getting a random element from the ArrayList is generally faster than getting a random element from the LinkedList, with the difference becoming more evident as the number of elements in the list increases. Getting elements sequentially from the ArrayList is much faster than getting elements sequentially from the LinkedList, especially as the number of elements in the list increases.

Based on the timing results, the ArrayList implementation appears to be better for adding elements, sorting the list, shuffling the list, and getting a random element from the list. On the other hand, the LinkedList implementation appears to be better for getting elements sequentially from the list.

These results are consistent with how the two list implementations are implemented. The ArrayList is implemented as an array that can be accessed directly by index, so adding elements to the end of the list is very fast. Sorting and shuffling the list can be done efficiently using algorithms that take advantage of the random access of the array. Getting a random element from the list is also very fast since we can calculate the index of the element directly.

The LinkedList, on the other hand, is implemented as a linked list of nodes that store the elements. Adding elements to the end of the list involves creating a new node and updating the next pointer of the previous node, so it is not as fast as adding elements to the end of the ArrayList. Sorting and shuffling the list requires iterating over the list and rearranging the nodes, which can be slower than sorting and shuffling the ArrayList. Getting elements sequentially from the list is faster than getting elements randomly, since we can simply iterate over the nodes of the list in order.