| Operation | Element Count | ArrayList  (MilliSeconds) | LinkedList  (MilliSeconds) |
| --- | --- | --- | --- |
| Add | 10 | 1.26 | 0.15 |
| Add | 1,000 | 6.96 | 0.90 |
| Add | 5,000 | 6.90 | 4.24 |
| Add | 10,000 | 15.02 | 5.63 |
| Sort | 10 | 0.42 | 0.14 |
| Sort | 1,000 | 1.43 | 1.22 |
| Sort | 5,000 | 10.01 | 5.51 |
| Sort | 10,000 | 9.81 | 4.82 |
| Shuffle | 10 | 0.03 | 0.02 |
| Shuffle | 1,000 | 0.67 | 0.30 |
| Shuffle | 5,000 | 1.55 | 4.02 |
| Shuffle | 10,000 | 2.71 | 3.29 |
| Random Get | 100 | 0.10 | 0.09 |
| Random Get | 10,000 | 1.78 | 15.75 |
| Random Get | 100,000 | 14.23 | 2,902.59 |
| Random Get | 1,000,000 | 133.81 | 19,023.76 |
| Sequential Get | 1,000 | 0.03 | 0.65 |
| Sequential Get | 5,000 | 0.25 | 61.96 |
| Sequential Get | 10,000 | 0.34 | 243.12 |
| Sequential Get | 100,000 | 1.99 | 603.01 |

* I noticed that when it comes to adding and sorting, it is more efficient to use LinkedLists, as it generates results at a faster basis.
* For Shuffle, LinkedLists is better for smaller datasets, but once the datasets get larger, performance improves with ArrayLists.
* For Random Gets & Sequential Gets, ArrayLists proved to be the dominant force, with performance comparison not even being a question.
* The results are consistent with how the lists are implemented because if you need to add or sort, a LinkedList can easily be manipulated, whereas an arrayList needs to generate a whole new container for the new data.
* Since an arrayList has a set index, it is extremely easy to get access to certain data, and it is faster to ‘get’ to that data.