|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | Ordering | Synchronized? | Duplicates allowed? | Null allowed? | Complexity of lookup, insertion,deletion |
| ArrayList | Insertion order | Unsynchronized | Allowed | Any number nulls | O(1) get,add O(n) remove |
| LinkedList | Insertion order | Unsynchronized | Allowed | Any number nulls | O(n) get,remove  O(1) amortized |
| HashSet | None | Unsynchronized | No | Single null | O(1) add,remove |
| LinkedHashSet | Insertion order | Unsynchronized | No | Single null | O(1) add,remove |
| TreeSet | Sorting order | Unsynchronized | No | Not allowed | O(log(n)) add,remove |
| HashMap | Not insertion order | Unsynchronized | No | One null key, any number null values | O(1) get,put,search |
| LinkedHashMap | Insertion order | Unsynchronized | No | One null key, any number null values | O(1) get,put |
| TreeMap | Sorting order | Unsynchronized | No | Not allowed | O(log(n) get,put |

Note:TreeMap and TreeSet enforce a natural sorting order, and therefore cannot allow nulls because they don’t follow any sort of sorting order.

It is not very easy to sort hashmap with comparator or comparable as there is no method within hashmap or map to do so, like there is for lists and sets. We can get the data from the hashmap and dump it into a list and sort it, then insert it into another hashmap if we want but there is no real way to do so using solely a hashmap.

One advantage of Iterator.remove() is that you can remove elements while iterating over the last, which is not the case for ArrayList.remove/Collections.remove()