Collection	Insertion Order	Synchronized	Duplicate Elements	Allowing of Null	Extra Stuff
ArrayList<T>	Maintained. Inserts at the end of list or specified index	unsynchronized	Yes	Yes	Contain O(n)
LinkedList<T>	Maintained. Insert at end, beginning, or specified index	unsynchronized	Yes	yes	-Doubly linked list Get, set O(n) Can be FIFO And LIFO
HashSet<E>	Unordered	Unsynchronized	No	Yes only 1	Avoid too high capacity(c) or low load factor Remove, add, contains O(1) Iterate O(c)
LinkedHashSet <E>	Maintained	Unsynchronized	No	Yes only 1	Remove, add, contains O(1) Iterate O(n)
Tree Set<E>	Ordered by natural or comparator ordering	Unsynchronized	No	No	Remove, add, contains O(logn)
HashMap<K ,V>	Unordered	unsynchronized	Yes but not duplicate keys	Yes	Loadfactor (0.75) Remove, add, contains O(1) Iterate O(c)
LinkedHashMap <K, V>	ordered	unsynchronized	Yes but not duplicate keys	Yes	Remove, add, contains O(1) Iterate O(n)
Tree Map<K, V>	Ordered by natural or comparator ordering	unsynchronized	Yes but not duplicate keys	Null elements but not keys	Remove, add, contains O(logn)

1. We can't sort a Hashmap directly using comparable or comparator because hashmap is a key, value pair <k, v> organized as a hash table and comparable and Comparator use a type parameter <T>. To sort a map, use tree map which uses comparator or comparable.
2. The remove method in iterator is to safely remove elements in a collection while iterating through it. If you do that without iterator, it will throw a concurrent modification exception.
3. A HashSet is backed by a hash table (hashmap instance) which allows add, remove, contains, and size operations to be constant time.