Asymptotics

The asymptotics of Guava's utilities are entirely predictable, but are listed here for completeness.

`List`

Implementation	`add`	`add(i, elem)`	`remove(i)`	`contains`	Iteration	`size`
`ArrayList` (JDK)	*O(1)*	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(1)*
`LinkedList` (JDK)	*O(1)*	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(1)*
`CopyOnWriteArrayList` (JDK)	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(1)*
`ImmutableList`	N/A	N/A	N/A	*O(n)*	*O(n)*	*O(1)*
`ImmutableSet.asList()`	N/A	N/A	N/A	*O(1)*	*O(n)*	*O(1)*

`Set`

Implementation	`add`	`remove`	`contains`	Iteration	`size`
`HashSet` (JDK)	*O(1)*	*O(1)*	*O(1)*	*O(max n)* `*`	*O(1)*
`LinkedHashSet` (JDK)	*O(1)*	*O(1)*	*O(1)*	*O(n)*	*O(1)*
`TreeSet` (JDK)	*O(log n)*	*O(log n)*	*O(log n)*	*O(n)*	*O(1)* `**`
`CopyOnWriteArraySet` (JDK)	*O(n)*	*O(n)*	*O(n)*	*O(n)*	*O(1)*
`ImmutableSet`	N/A	N/A	*O(1)*	*O(n)*	*O(1)*
`ImmutableSortedSet`	N/A	N/A	*O(log n)*	*O(n)*	*O(1)*

`*` `HashSet` iteration takes time proportional to the maximum number of elements the `HashSet` has ever had, not proportional to the current number of elements.

`**` `TreeSet.subSet(...).size()` takes time proportional to the size of the subset.

`Multiset`

Note: *n* is the number of **distinct** elements in the multiset.

Implementation	Performs like a…	size()	count(E)	add(E, int)	remove(E, int)	setCount(E, int)	Iterate through entrySet() or elementSet()
HashMultiset	HashMap<E, Integer>	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(max n)* *
LinkedHashMultiset	LinkedHashMap<E, Integer>	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(n)*
TreeMultiset	TreeMap<E, Integer>	*O(1)* **	*O(log n)*	*O(log n)*	*O(log n)*	*O(log n)*	*O(n)*
ConcurrentHashMultiset	ConcurrentHashMap<E, AtomicInteger>	*O(n)*	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(n)*
ImmutableMultiset	ImmutableMap<E, Integer>	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(n)*
ImmutableSortedMultiset	ImmutableSortedMap<E, Integer>	*O(1)*	*O(log n)*	*O(log n)*	*O(log n)*	*O(log n)*	*O(n)*

* Like `HashMap`, the iteration cost through the `entrySet` is linear in the maximum number of elements the `HashMultiset` has ever had, not the number it has now.

** `TreeMultiset.subMultiset().size()` takes time *O(log n)*.

`Multimap`

`k` is the number of distinct keys; `n` is the number of distinct entries; `#(key)` is the number of entries associated with `key`. Where not specified, the asymptotics are equivalent to the "obvious" implementation based on the "Performs like a…" column.

Implementation	Performs like a…	size()	get(K)	put(K, V)	containsEntry(K, V)	Iterate through entries()	Iterate through asMap().entrySet()
ArrayListMultimap	HashMap<K, ArrayList<V>>	*O(1)*	*O(1)*	*O(1)*	*O(#(key))*	*O(max k + n)*	*O(max k)*
LinkedListMultimap	LinkedHashMap<K, LinkedList<V>>	*O(1)*	*O(1)*	*O(1)*	*O(#(key))*	*O(n)*	*O(k)*
HashMultimap	HashMap<K, HashSet<V>>	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(max n)*	*O(max k)*
LinkedHashMultimap	LinkedHashMap<K, LinkedHashSet<V>>	*O(1)*	*O(1)*	*O(1)*	*O(1)*	*O(n)*	*O(k)*
TreeMultimap	TreeMap<K, TreeSet<V>>	*O(1)*	*O(log k)*	*O(log k + log #(key))*	*O(log k + log #(key))*	*O(n)*	*O(k)*
ImmutableListMultimap	ImmutableMap<K, ImmutableList<V>>	*O(1)*	*O(1)*	N/A	*O(#(key))*	*O(n)*	*O(k)*

Implementation	Performs like a...	`size()`	`get(K)`	`put(K, V)`	`containsEntry(K, V)`	Iterate through `entries()`	Iterate through `asMap().entrySet()`
`ImmutableSetMultimap`	`ImmutableMap<K, ImmutableSet<V>>`	*O(1)*	*O(1)*	N/A	*O(1)*	*O(n)*	*O(k)*