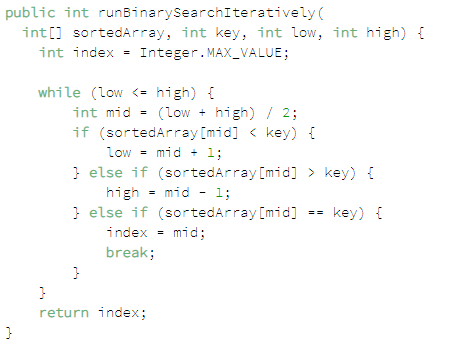
<https://www.baeldung.com/java-binary-search>

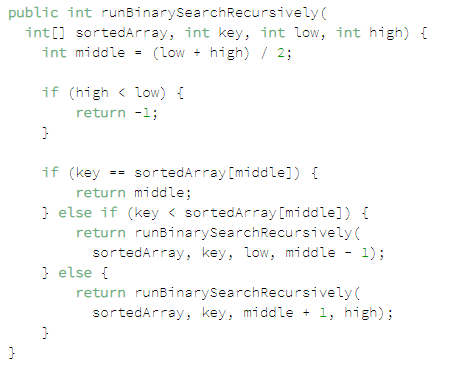
**Binary Search**

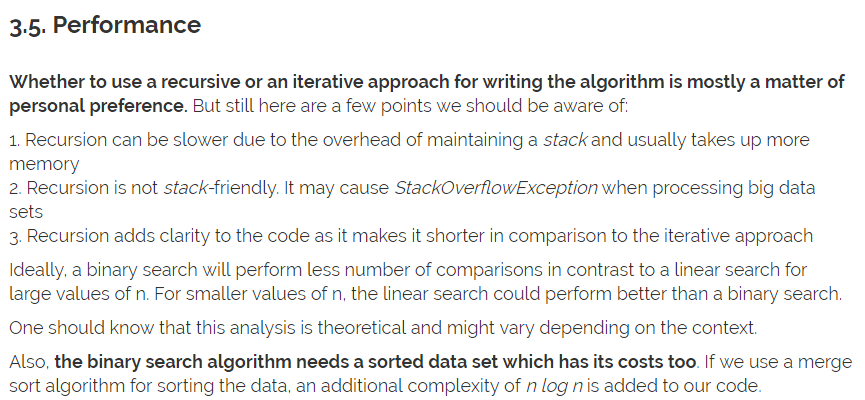
If we start saving items in sorted order and search for items using the binary search, we can achieve a complexity of O(log n).

**With binary search, the time taken by the search results naturally increases with the size of the dataset, but not proportionately.**



**Recursive Impl**





[*LinkedList*](https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) is a doubly-linked list implementation of the [*List*](https://docs.oracle.com/javase/8/docs/api/java/util/List.html) and *[Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html)* interfaces. It implements all optional list operations and permits all elements (including null).

[**https://www.baeldung.com/java-linkedlist**](https://www.baeldung.com/java-linkedlist) **AL and LL differences**

LinkedList will be a better fit, such as preferences for constant insertion/deletion time (e.g., frequent insertions/deletions/updates), over constant access time and effective memory usage.

[**https://www.baeldung.com/java-arraylist**](https://www.baeldung.com/java-arraylist)

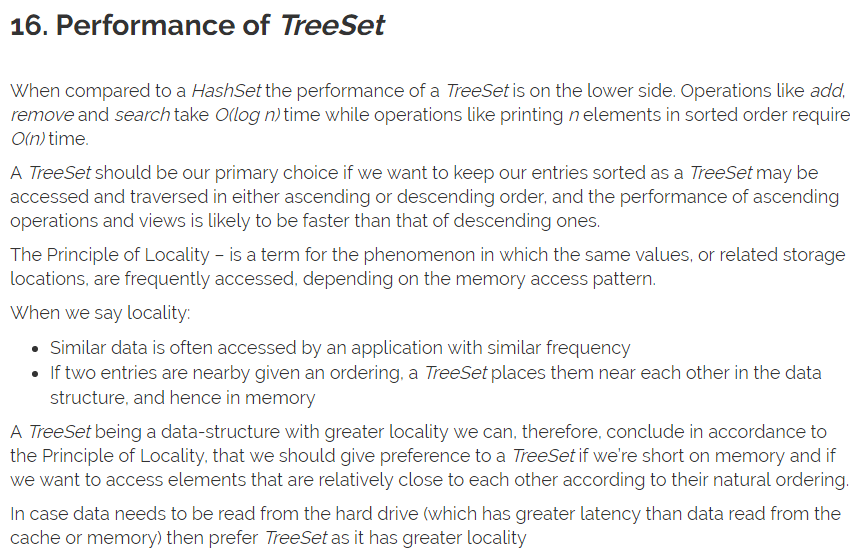
K provides a nice way to get an unmodifiable collection out of an existing one:

|  |  |
| --- | --- |
| 1 | Collections.unmodifiableList(list); |

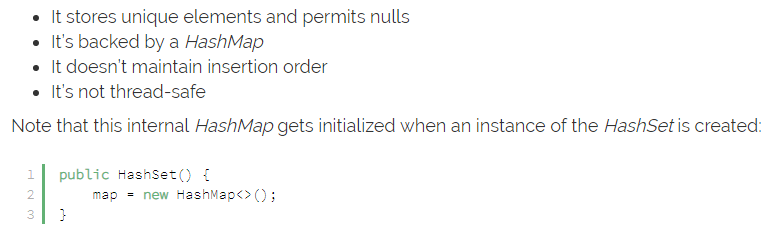
[*CopyOnWriteArrayList*](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) from the java.util.concurrentpackage.

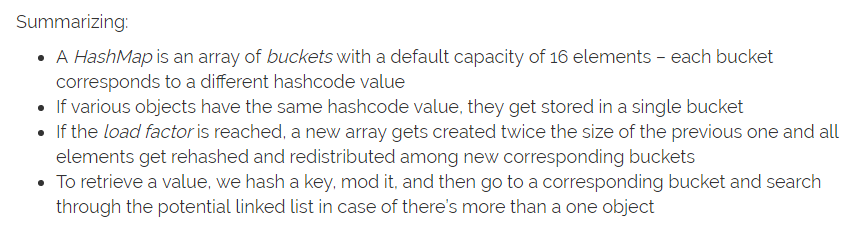
This is a very useful construct in the multi-threaded programs – when we want to iterate over a list in a thread-safe way without an explicit synchronization.

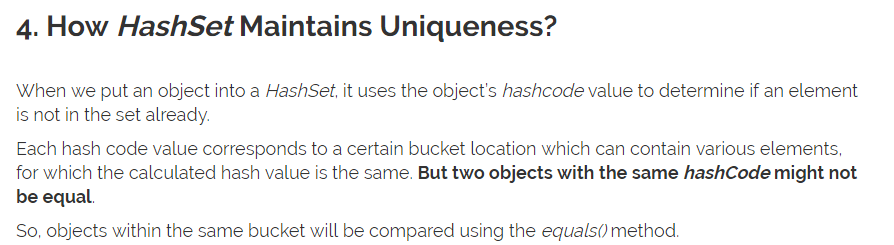
[**https://www.baeldung.com/java-tree-set**](https://www.baeldung.com/java-tree-set)

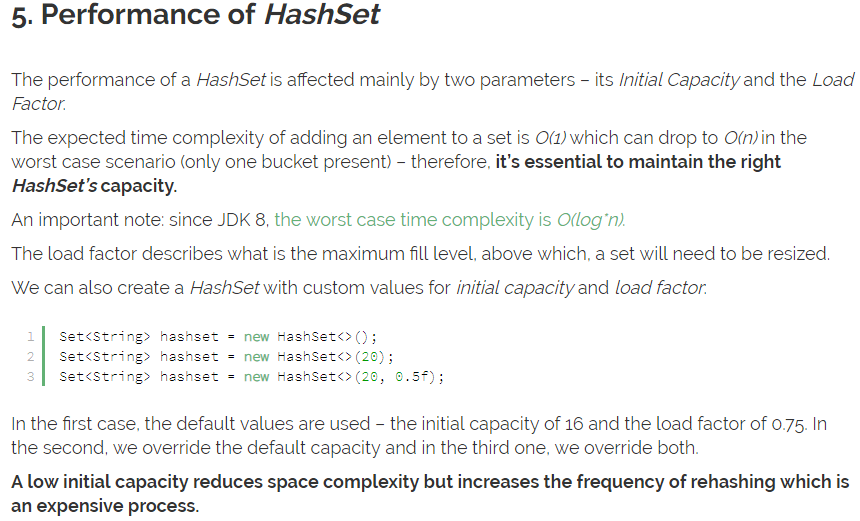


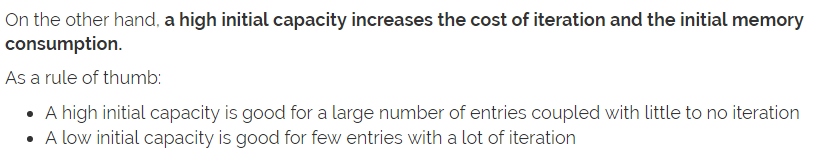
[**https://www.baeldung.com/java-hashset**](https://www.baeldung.com/java-hashset)











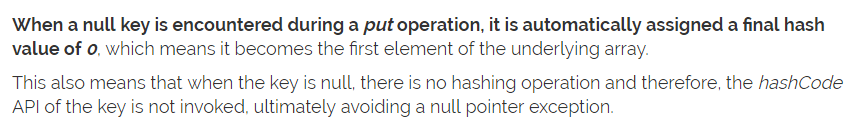
[**https://www.baeldung.com/java-hashmap**](https://www.baeldung.com/java-hashmap)

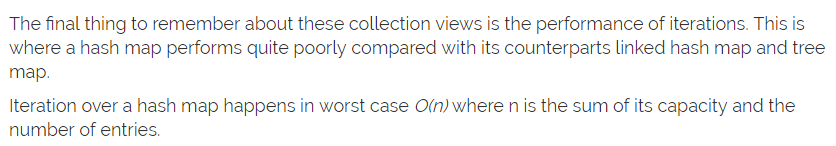


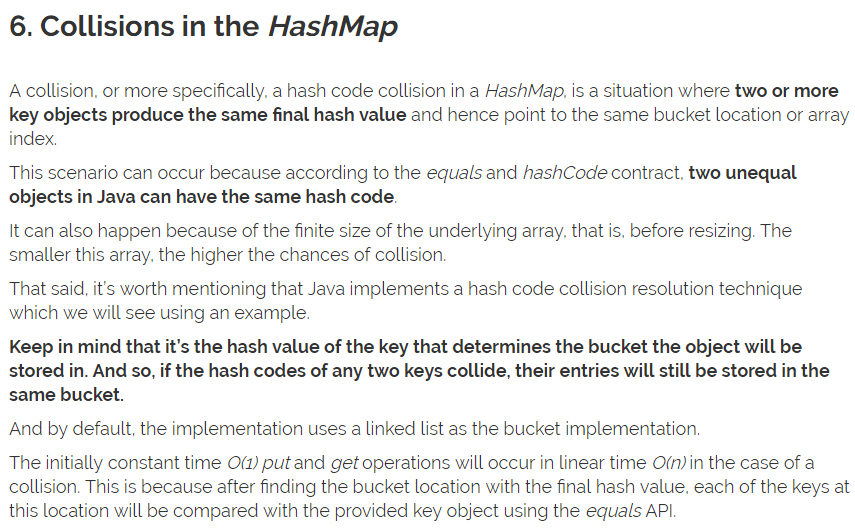








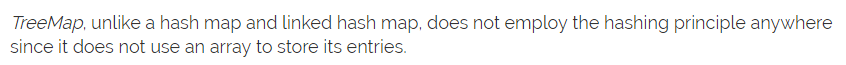


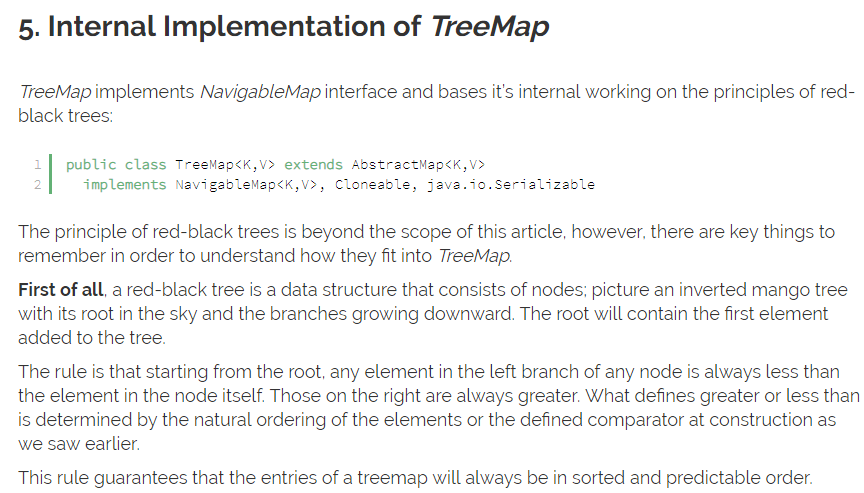


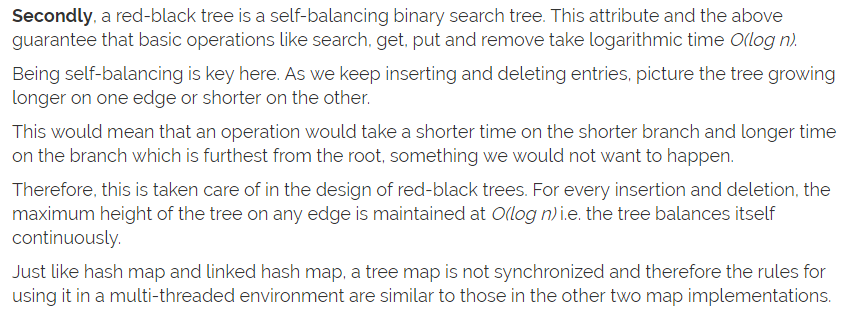
Java 8, the linked lists are dynamically replaced with balanced binary search trees in collision resolution after the number of collisions in a given bucket location exceed a certain threshold.

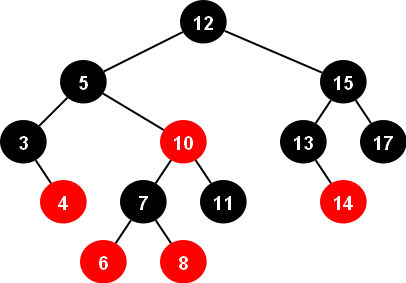
[**https://www.baeldung.com/java-treemap**](https://www.baeldung.com/java-treemap)

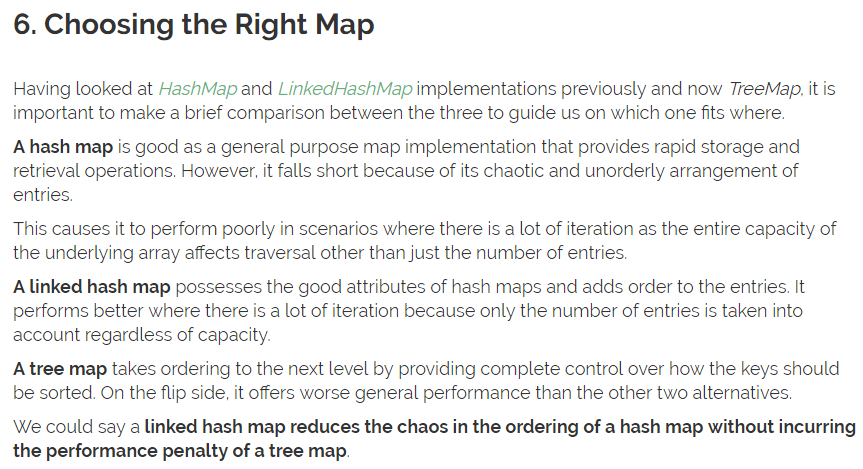




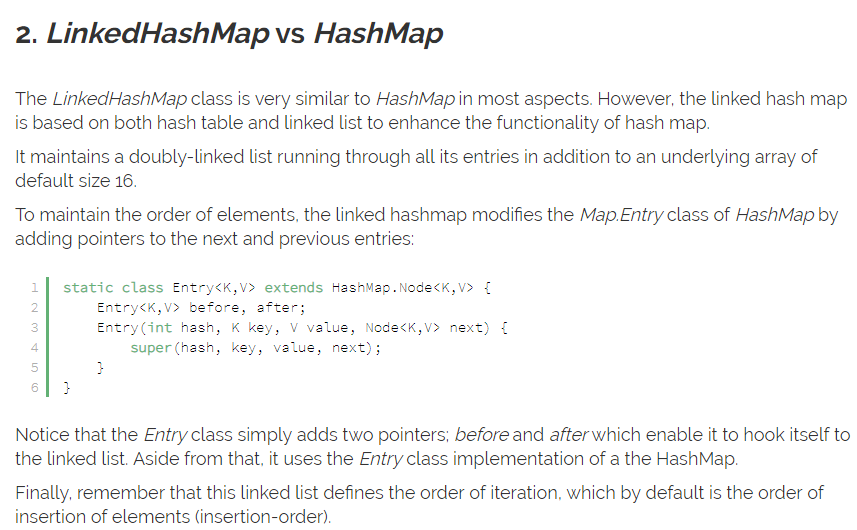








[**https://www.baeldung.com/java-linked-hashmap**](https://www.baeldung.com/java-linked-hashmap)



Map m = Collections.synchronizedMap(new LinkedHashMap());

However, this **constant-time performance of LinkedHashMap is likely to be a little worse than the constant-time of HashMap** due to the added overhead of maintaining a doubly-linked list.

Iteration over collection views of LinkedHashMap also takes linear time O(n) similar to that of HashMap. On the flip side,**LinkedHashMap‘s linear time performance during iteration is better than HashMap‘s linear time**.

This is because, for LinkedHashMap, n in O(n) is only the number of entries in the map regardless of the capacity. Whereas, for HashMap, n is capacity and the size summed up, O(size+capacity).

[**https://www.baeldung.com/java-treemap-vs-hashmap**](https://www.baeldung.com/java-treemap-vs-hashmap)

