1. Array and Collection.

Sr. No.	Key	Arrays	Collection
1	Size	Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward.	The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration.
2	Memory Consumption	Arrays due to fast execution consumes more memory and has better performance.	Collections, on the other hand, consume less memory but also have low performance as compared to Arrays.
3	Data type	Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays.	Collection, on the other hand, can hold both homogeneous and heterogeneous elements.
4	Primitives storage	Arrays can hold both object and primitive type data.	On the other hand, collection can hold only object types but not the primitive type of data.
5	Performance	Arrays due to its storage and internal implementation better in performance.	Collection on the other hand with respect to performance is not recommended to use.

2. Array List and Linked List.

Sr. No.	Key	ArrayList	LinkedList
1	Internal Implementation	ArrayList internally uses a dynamic array to store its elements.	LinkedList uses Doubly Linked List to store its elements.
2	Manipulation	ArrayList is slow as array manipulation is slower.	LinkedList is faster being node based as not much bit shifting required.

3	Implementation	ArrayList implements only List.	LinkedList implements List as well as Queue. It can acts as a queue as well.
4	Access	ArrayList is faster in storing and accessing data.	LinkedList is faster in manipulation of data.

3. List and Set

Sr. No.	Key	List	Set
1	Positional Access	The list provides positional access of the elements in the collection.	Set doesn't provide positional access to the elements in the collection.
2	Implementation	Implementation of List are ArrayList,LinkedList,Vector ,Stack.	Implementation of a set interface is HashSet and LinkedHashSet.
3	Duplicate	We can store the duplicate elements in the list.	We can't store duplicate elements in Set.
4	Ordering	List maintains insertion order of elements in the collection.	Set doesn't maintain any order.
5	Null Element	The list can store multiple null elements.	Set can store only one null element.

4. Collection and Collections.

	Collection	Collections
Definition	It is an interface that forms the root of the Java Collections Framework.	This utility class provides static methods to perform operations on collections.
Usage	Act as an interface for other interfaces like List, Set, and Queue. Some of the common methods include **add()**, **remove()**, and **size()**.	This class offers utility functions that operate on or return collections. For example, the **Collections.sort()** method sorts elements in a list.
Implementation	It doesn't have any direct implementation as it is an interface. But you can implement the collection interface by using various Java classes, like ArrayList, HashSet, and PriorityQueue.	As it is a final class, meaning it can not be subclassed. It doesn't have any public constructors, so its methods are accessed statically.

Flexibility	Since it's an interface, developers can create custom implementations if required.	As a utility class, it provides ready-to-use methods that help the developer to save time while performing common operations on collections.
Purpose	The purpose is to provide a standard way to handle and manage a group of objects.	It offers a utility function that simplifies common tasks associated with collections.

5. Comparable and Comparator.

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.