|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| ArrayLists are not synchronized which meaning multiple threads can call methods on it at a time. | Vectors are synchronized meaning only one thread can call methods on a vector at a time. |
| ArrayLists increase its size by 50 percent. | Vectors default to doubling the size of its array. |
| ArrayLists use iterator interface to traverse the elements. | Vectors use Enumeration interface to traverse the elements but can use iterator too. |

|  |  |
| --- | --- |
| **HashSet** | **SortedSet** |
| Uses HashTable to store data. | Uses red-black tree to store data, which is a binary tree. |
| It is faster compared to SortedSet and has a time complexity of O(1). | It is slower compared to HashSet and has time complexity of O(logN). |
| Data is not sorted in any order. | Data is sorted in form of a tree. |

|  |  |
| --- | --- |
| **TreeSet** | **HashSet** |
| TreeSet is slower because of sorting operation it needs to perform on each insertion.It has a time complexity of O(LogN). | It is faster compared to TreeSet and has a time complexity of O(1). |
| TreeSet is backed up by NavigableMap in Java and by default it uses TreeMap. | Uses HashTable to store data. |
| Does not allow null value to be inserted and throws null pointer exception when it is inserted. | HashSet allows null value to be inserted. |

|  |  |
| --- | --- |
| **Array** | **List** |
| In an array, data is stored in contiguous blocks of memory. | In a List, data is stored in a random order. |
| Insertions or deletions usually mean moving all of the elements in the array along to make space for the new element or fill in a gap. | Cost of each operation on the list highly depends on the implementation of the list. |
| Arrays store the actual element. | Lists contain references to elements. |
| Time Complexity to access elements is Linear. | Time complexity may vary depending on the operations. |

|  |  |
| --- | --- |
| **List** | **Set** |
| Maintains the elements in insertion order | Elements are not maintained any order |
| List allow duplicate elements | Set doesn't allow duplicate elements |
| List permits any number of null values | Set permits only one null value in its collection |
| List can be inserted in in both forward direction and backward direction using Listiterator. | Set can be traversed only in forward direction with the help of iterator. |

|  |  |
| --- | --- |
| **NavigableSet** | **NavigableMap** |
| It behaves like a SortedSet. | Uses map data structure. |
| Stores distinct values (No duplicates). | It stores keys and elements can be obtained by them. |
| Has limited functions compared to navigablemap and can retrieve various elements throughout the set. | It has many functions which allow the Map to be displayed in different forms. |