**Java BinarySearch:**

**public static <T> int binarySearch(List<? extends Comparable<? super T>> list,**

**T key)**

Searches the specified list for the specified object using the binary search algorithm. The list must be sorted into ascending order according to the natural ordering of its elements (as by the sort(List) method) prior to making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.

This method runs in log(n) time for a "random access" list (which provides near-constant-time positional access). If the specified list does not implement the RandomAccess interface and is large, this method will do an iterator-based binary search that performs O(n) link traversals and O(log n) element comparisons.

**Parameters:**

list - the list to be searched.

key - the key to be searched for.

Returns:

the index of the search key, if it is contained in the list; otherwise, (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or list.size() if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

**Throws:**

ClassCastException - if the list contains elements that are not mutually comparable (for example, strings and integers), or the search key is not mutually comparable with the elements of the list.

**public static <T> int binarySearch(List<? extends T> list,**

**T key,**

**Comparator<? super T> c)**

Searches the specified list for the specified object using the binary search algorithm. The list must be sorted into ascending order according to the specified comparator (as by the sort(List, Comparator) method), prior to making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.

This method runs in log(n) time for a "random access" list (which provides near-constant-time positional access). If the specified list does not implement the RandomAccess interface and is large, this method will do an iterator-based binary search that performs O(n) link traversals and O(log n) element comparisons.

**Parameters:**

list - the list to be searched.

key - the key to be searched for.

c - the comparator by which the list is ordered. A null value indicates that the elements' natural ordering should be used.

**Returns:**

the index of the search key, if it is contained in the list; otherwise, (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or list.size() if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

**Throws:**

ClassCastException - if the list contains elements that are not mutually comparable using the specified comparator, or the search key is not mutually comparable with the elements of the list using this comparator.

**First Example:**

// Java program to demonstrate working of Collections.

// binarySearch() in an array sorted in descending order.

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

public class GFG

{

public static void main(String[] args)

{

List al = new ArrayList();

al.add(2);

al.add(10);

al.add(30);

al.add(50);

al.add(100);

// The last parameter specifies the comparator method

// used for sorting.

int index = Collections.binarySearch(al, 50);

System.out.println("Found at index " + index);

}

}

**Another Example: (Finding An Element When the Array is reversely sorted)**

// Java program to demonstrate working of Collections.

// binarySearch() in an array sorted in descending order.

import java.util.List;

import java.util.ArrayList;

import java.util.Collections;

public class GFG

{

public static void main(String[] args)

{

List al = new ArrayList();

al.add(100);

al.add(50);

al.add(30);

al.add(10);

al.add(2);

// The last parameter specifies the comparator method

// used for sorting.

int index = Collections.binarySearch(al, 50,

Collections.reverseOrder());

System.out.println("Found at index " + index);

}

}

**Here, Collections.reverseOrder() is the Comparator**

**It is used to find a key which is arranged in reverseOrder.**