List BinarySearch() Method in C#

geeksforgeeks.org/list-binarysearch-method-in-c-sharp

January 10, 2019

List<T>.BinarySearch() Method uses a binary search algorithm to locate a specific element in the sorted List<T> or a portion of it. There are 3 methods in the overload list of this method as follows:

- **BinarySearch(T)**
- **BinarySearch(T, IComparer<T>)**
- **BinarySearch(Int32, Int32, T, IComparer<T>)**

BinarySearch(T) Method

This method searches for an element in the entire sorted **List<T>** using the *default comparer* and returns the zero-based index of the searched element.

Syntax:

```
public int BinarySearch (T item);
```

Here, item is the object which is to be locate and the value of *item* can be ***null*** or ***reference*** type.

Return Type: If the *item* is found, then this method returns the *zero-based index* of the element to be searched for and if not found, then a negative number that is the *bitwise* complement of the index of the next element will be return and the complement is larger than that item. If there is no larger element, the *bitwise* complement of *Count* will be return.

Exception: This method will give ***InvalidOperationException*** if the default comparer *Default* cannot find an implementation of the IComparable<T> generic interface or the IComparable interface for type T.

Below programs illustrate the use of above-discussed method:

Example 1:

filter_none
edit

play_arrow

brightness_4

```
using System;
using System.Collections.Generic;
class GFG {
public static void Main()
{
List< string > Geek = new List< string >();
Geek.Add( "ABCD" ):
Geek.Add( "ORST" );
Geek.Add( "XYZ" ):
Geek.Add( "IIKL" ):
Console.WriteLine( "The Original List is:" );
foreach ( string g in Geek)
{
Console.WriteLine(g);
}
Console.WriteLine( "\nThe List in Sorted form" );
Geek.Sort():
Console.WriteLine();
foreach ( string g in Geek)
{
Console.WriteLine(g);
}
Console.WriteLine( "\nInsert EFGH :" );
int index = Geek.BinarySearch( "EFGH" );
if (index < 0)
{
Geek.Insert(~index, "EFGH" );
}
Console.WriteLine();
foreach ( string g in Geek)
{
Console.WriteLine(g);
}
}
}
```

Output:

```
The Original List is:
ABCD
QRST
XYZ
IJKL

The List in Sorted form

ABCD
IJKL
QRST
XYZ

Insert EFGH :

ABCD
EFGH
IJKL
QRST
XYZ
```

Example 2: In this example, the List is created with some integer values and to insert a new integer using *BinarySearch(T)* method in the List by using a user define function.

filter_none
edit

play_arrow

brightness_4

```
using System;
using System.Collections.Generic;
class GFG {
public void binarySearch(List< int > Geek)
{
Console.WriteLine( "\nInsert 3 :" );
int index = Geek.BinarySearch(3);
if (index < 0)
{
Geek.Insert(~index, 3);
}
foreach ( int g in Geek)
{
Console.WriteLine(g);
}
}
}
public class search {
public static void Main()
{
GFG gg = new GFG();
List< int > Geek = new List< int >() {
5, 6, 1, 9};
Console.WriteLine( "Original List" );
foreach ( int g in Geek)
{
Console.WriteLine(g);
}
Console.WriteLine( "\nList in Sorted form" );
Geek.Sort();
foreach ( int g in Geek)
{
Console.WriteLine(g);
}
gg.binarySearch(Geek);
}
}
```

Output:

```
Original List
5
6
1
9

List in Sorted form
1
5
6
9

Insert 3 :
1
3
5
6
9
```

BinarySearch(T) Method

This method searches for an element in the entire sorted List using the specified comparer and returns the zero-based index of the searched element.

Syntax:

> public int BinarySearch (T item, System.Collections.Generic.IComparer<T> comparer);

Parameters:

- **item** : It is the item to locate and the value of the item can be *null* for reference types.
- **comparer** : It is the IComparer<T> implementation to use when comparing elements.

Return Value: If the item founds, then this method returns the zero-based index of the element to be searched for and if not found, then a negative number that is the bitwise complement of the index of the next element that is larger than item or, if there is no larger element, the bitwise complement of Count.

Exception: This method will give *InvalidOperationException* if the comparer is null, and the default comparer Default cannot find an implementation of the IComparable<T> generic interface or the IComparable interface for type *T*.

Below programs illustrate the use of the above-discussed method:

Example 1:

filter_none
edit

play_arrow

brightness_4

```
using  System;
using  System.Collections.Generic;
class  GFG : IComparer<  string  > {
public  int  Compare(  string  x,  string  y)
{
if  (x ==  null  || y ==  null  )
{
return  0;
}
return  x.CompareTo(y);
}
}
class  geek {
public  static  void  Main()
{
List<  string  > list1 =  new  List<  string  >();
list1.Add(  "B"  );
list1.Add(  "C"  );
list1.Add(  "E"  );
list1.Add(  "A"  );
Console.WriteLine(  "Original string"  );
foreach  (  string  g  in  list1)
{
Console.WriteLine(g);
}
GFG gg =  new  GFG();
list1.Sort(gg);
Console.WriteLine(  "\nList in sorted form"  );
foreach  (  string  g  in  list1)
{
Console.WriteLine(g);
}
int  index = list1.BinarySearch(  "D"  , gg);
if  (index < 0)
{
list1.Insert(~index,  "D"  );
}
Console.WriteLine(  "\nAfetr inserting \"D\" in the List"  );
foreach  (  string  g  in  list1)
{
Console.WriteLine(g);
}
}
}
```

Output:

```
Original string
B
C
E
A

List in sorted form
A
B
C
E

Afetr inserting "D" in the List
A
B
C
D
E
```

Example 2: In this example, the List is created with some integer values and to insert a new integer using BinarySearch(T, Comparer <T>) method in the List by using a user define function.

filter_none
edit

play_arrow

brightness_4

```
using System;
using System.Collections.Generic;
class GFG : IComparer< int > {
public int Compare( int x, int y)
{
if (x == 0 || y == 0)
{
return 0;
}
return x.CompareTo(y);
}
}
class geek {
public static void Main()
{
List< int > list1 = new List< int >() {
5, 6, 1, 9};
Console.WriteLine( "Original string" );
foreach ( int g in list1)
{
Console.WriteLine(g);
}
GFG gg = new GFG();
list1.Sort(gg);
Console.WriteLine( "\nList in sorted form" );
foreach ( int g in list1)
{
Console.WriteLine(g);
}
bSearch b = new bSearch();
b.binarySearch(list1);
}
}
class bSearch {
public void binarySearch(List< int > list1)
{
GFG gg = new GFG();
int index = list1.BinarySearch(3, gg);
if (index < 0)
{
list1.Insert(~index, 3);
}
Console.WriteLine( "\nAfter inserting \"3\" in the List" );
foreach ( int g in list1)
{
Console.WriteLine(g);
}
}
}
```

Output:

```
Original string
5
6
1
9

List in sorted form
1
5
6
9

After inserting "3" in the List
1
3
5
6
9
```

BinarySearch(Int32, Int32, T, IComparer<T>)

This method is used to search a range of elements in the sorted List<T> for an element using the specified comparer and returns the zero-based index of the element.

Syntax:

> public int BinarySearch (int index, int count, T item, System.Collections.Generic.IComparer<T> comparer);

Parameters:

> **index**: It is the zero-based starting index of the range to search.
>
> **count**: It is the length of the range to search.
>
> **item**: It is the object to locate. The value can be null for the reference type.
>
> **comparer**: It is the IComparer implementation to use when comparing elements, or null to use the default comparer Default.

Return Value: It returns the zero-based index of item in the sorted List<T>, if the item is found; otherwise, a negative number that is the bitwise complement of the index of the next element that is larger than item or, if there is no larger element, the bitwise complement of Count.

Exceptions:

- **ArgumentOutOfRangeException**: If the index is less than 0 or count is less than 0.
- **ArgumentException**: If the index and count do not represent a valid range.
- **InvalidOperationException**: If the comparer is null.

Example:

filter_none
edit

play_arrow

brightness_4

```
using System;
using System.Collections.Generic;
class GFG : IComparer< int >
{
public int Compare( int x, int y)
{
if (x == 0 || y == 0)
{
return 0;
}
return x.CompareTo(y);
}
}
class search {
public void binarySearch(List< int > list1,
int i)
{
Console.WriteLine( "\nBinarySearch a " +
"range and Insert 3" );
GFG gg = new GFG();
int index = list1.BinarySearch(0, i,
3, gg);
if (index < 0)
{
list1.Insert(~index, 3);
i++;
}
Display(list1);
}
public void Display(List< int > list)
{
foreach ( int g in list )
{
Console.WriteLine(g);
}
}
}
class geek
{
public static void Main()
{
List< int > list1 = new List< int >()
{
```

```
15,4,2,9,5,7,6,8,10
};
int  i = 7;
Console.WriteLine(  "Original List"  );
search d =   new  search();
d.Display(list1);
GFG gg =   new  GFG();
Console.WriteLine(  "\nSort a range with "  +
"the alternate comparer"  );
list1.Sort(1, i, gg);
d.Display(list1);
d.binarySearch(list1,i);
}
}
```

Output:

```
Original List
15
4
2
9
5
7
6
8
10

Sort a range with the alternate comparer
15
2
4
5
6
7
8
9
10

BinarySearch a range and Insert 3
15
2
3
4
5
6
7
8
9
10
```

Note:

- If the List<T> contains more than one element with the same value, the method returns only one of the occurrences, and it might return any one of the occurrences, not necessarily the first one.
- The List<T> must already be sorted according to the comparer implementation; otherwise, the result is incorrect.
- This method is an O(log n) operation, where n is the number of elements in the range.

Reference:

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.binarysearch?view=netframework-4.7.2

SoumikMondal
3rd year student of Information Technology JADAVPUR UNIVERSITY

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.