# System.Collections.Generic.IComparer<-T> Interface

```
[ILAsm]
.class public interface abstract IComparer<-T>

[C#]
public interface IComparer<in T>
```

**Assembly Info:**

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 4.0.0.0
- *Attributes:*
  - CLSCompliantAttribute(true)

**Summary**

Defines a method that a type implements to compare two objects.

**Library:** BCL

**Description**

This interface is used with the `System.Collections.Generic.List`1<T>.Sort` and `System.Collections.Generic.List`1<T>.BinarySearch` methods. It provides a way to customize the sort order of a collection. Classes that implement this interface include the `System.Collections.Generic.SortedDictionary`2<T1,T2>` and `System.Collections.Generic.SortedList`2<T1,T2>` generic classes.

The default implementation of this interface is the `System.Collections.Generic.Comparer`1<T>` class. The `System.StringComparer` class implements this interface for type `System.String`.

This interface supports ordering comparisons. That is, when the `System.Collections.Generic.Comparer`1<T>.Compare` method returns 0, it means that two objects sort the same. Implementation of exact equality comparisons is provided by the `System.Collections.Generic.IEqualityComparer`1<T>` generic interface.

We recommend that you derive from the `System.Collections.Generic.Comparer`1<T>` class instead of implementing the `System.Collections.Generic.IComparer`1<T>` interface, because the `System.Collections.Generic.Comparer`1<T>` class provides an explicit interface implementation of the `System.Collections.Generic.Comparer`1<T>.System#Collections#IComparer#Compare` method and the `System.Collections.Generic.Comparer`1<T>.Default` property that gets the default comparer for the object.

# IComparer<-T>.Compare(T, T) Method

```
[ILAsm]
.method public hidebysig newslot abstract virtual instance int32
Compare(!0 x, !0 y) cil managed

[C#]
public int Compare (T x, T y)
```

## Summary

Compares two objects and returns a value indicating whether one is less than, equal to, or greater than the other.

## Parameters

| Parameter | Description |
|---|---|
| *x* | The first object to compare. |
| *y* | The second object to compare. |

## Return Value

A signed integer that indicates the relative values of *x* and *y*, as shown in the following table.

| Value | Meaning |
|---|---|
| Less than zero | *x* is less than *y*. |
| Zero | *x* equals *y*. |
| Greater than zero | *x* is greater than *y*. |

## Description

Implement this method to provide a customized sort order comparison for type *T*.

Comparing `null` with any reference type is allowed and does not generate an exception. A null reference is considered to be less than any reference that is not null.

## Behaviors

For any objects A, B and C, the following are required to be true:

Compare(A,A) is required to return zero.

If Compare(A,B) returns zero then Compare(B,A) is required to return zero.

If Compare(A,B) is zero, then Compare(B,C) and Compare(A,C) must have the same sign (negative, zero or positive).

If Compare(B,C) is zero, then Compare(A,B) and Compare(A,C) must have the same sign (negative, zero or positive).

If Compare(A,B) returns zero and Compare(B,C) returns zero then Compare(A,C) is required to return zero.

If Compare(A,B) returns a value other than zero then Compare(B,A) is required to return a value of the opposite sign.

If Compare(A,B) returns a value *x* not equal to zero, and Compare(B,C) returns a value *y* of the same sign as *x*, then Compare(A,C) is required to a value of the same sign as *x* and *y*.

The exact behavior of this method is unspecified. The intent of this method is to provide a mechanism that orders instances of a class in a manner that is consistent with the mathematical definitions of the relational operators (<, >, and ==), without regard for class-specific definitions of the operators.