```csharp
// Decompiled with JetBrains decompiler
// Type: System.Linq.Enumerable
// Assembly: System.Linq, Version=4.2.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
// MVID: 6BC5D5C3-B137-47AC-970B-82F3552008E0
// Assembly location: C:\Program Files\dotnet\sdk\NuGetFallbackFolder\microsoft.netcore.app\2.0.0\ref\netcoreapp2.0\System.Linq.dll

using System.Collections;
using System.Collections.Generic;

namespace System.Linq
{
  /// <summary>Provides a set of static (Shared in Visual Basic) methods for querying objects that implement <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
  public static class Enumerable
  {
    /// <summary>Applies an accumulator function over a sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate over.</param>
    /// <param name="func">An accumulator function to be invoked on each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The final accumulator value.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="func">func</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static TSource Aggregate<TSource>(this IEnumerable<TSource> source, Func<TSource, TSource, TSource> func);
    /// <summary>Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate over.</param>
    /// <param name="seed">The initial accumulator value.</param>
    /// <param name="func">An accumulator function to be invoked on each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TAccumulate">The type of the accumulator value.</typeparam>
    /// <returns>The final accumulator value.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="func">func</paramref> is null.</exception>
    public static TAccumulate Aggregate<TSource, TAccumulate>(this IEnumerable<TSource> source, TAccumulate seed, Func<TAccumulate, TSource,
        TAccumulate> func);
    /// <summary>Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value, and the specified function is used to select the result value.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate over.</param>
    /// <param name="seed">The initial accumulator value.</param>
    /// <param name="func">An accumulator function to be invoked on each element.</param>
    /// <param name="resultSelector">A function to transform the final accumulator value into the result value.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TAccumulate">The type of the accumulator value.</typeparam>
    /// <typeparam name="TResult">The type of the resulting value.</typeparam>
    /// <returns>The transformed final accumulator value.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="func">func</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
    public static TResult Aggregate<TSource, TAccumulate, TResult>(this IEnumerable<TSource> source, TAccumulate seed, Func<TAccumulate, TSource, TAccumulate> func, Func<TAccumulate, TResult> resultSelector);
    /// <summary>Determines whether all elements of a sequence satisfy a condition.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements to apply the predicate to.</param>
    /// <param name="predicate">A function to test each element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>true if every element of the source sequence passes the test in the specified predicate, or if the sequence is empty; otherwise, false.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static bool All<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Determines whether a sequence contains any elements.</summary>
    /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to check for emptiness.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>true if the source sequence contains any elements; otherwise, false.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
```

```
        public static bool Any<TSource>(this IEnumerable<TSource> source);
        /// <summary>Determines whether any element of a sequence satisfies a condition.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to apply the predicate to.</param>
        /// <param name="predicate">A function to test each element for a condition.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>true if any elements in the source sequence pass the test in the specified predicate; otherwise, false.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        public static bool Any<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
        /// <param name="source"></param>
        /// <param name="element"></param>
        /// <typeparam name="TSource"></typeparam>
        /// <returns></returns>
        public static IEnumerable<TSource> Append<TSource>(this IEnumerable<TSource> source, TSource element);
        /// <summary>Returns the input typed as <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
        /// <param name="source">The sequence to type as <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The input sequence typed as <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</returns>
        public static IEnumerable<TSource> AsEnumerable<TSource>(this IEnumerable<TSource> source);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Decimal"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static decimal Average(this IEnumerable<decimal> source);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Double"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
```

```csharp
        public static double Average(this IEnumerable<double> source);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int32"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Average(this IEnumerable<int> source);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Int64"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int64"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Average(this IEnumerable<long> source);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Decimal"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
        public static decimal? Average(this IEnumerable<decimal?> source);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Double"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static double? Average(this IEnumerable<double?> source);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></see> values to calculate the average of.</param>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</
        exception>
    public static double? Average(this IEnumerable<int?> source);
    /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Int64"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></see> values to calculate the average of.</param>
    /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
    public static double? Average(this IEnumerable<long?> source);
    /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Single"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see cref="T:System.Single"></see> values to calculate the average of.</param>
    /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static float? Average(this IEnumerable<float?> source);
    /// <summary>Computes the average of a sequence of <see cref="T:System.Single"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Single"></see> values to calculate the average of.</param>
    /// <returns>The average of the sequence of values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static float Average(this IEnumerable<float> source);
    /// <summary>Computes the average of a sequence of <see cref="T:System.Decimal"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate an average.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The average of the sequence of values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
    public static decimal Average<TSource>(this IEnumerable<TSource> source,
            Func<TSource, decimal> selector);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Double"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Average<TSource>(this IEnumerable<TSource> source, Func<TSource, double> selector);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Int32"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static double Average<TSource>(this IEnumerable<TSource> source, Func<TSource, int> selector);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Int64"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static double Average<TSource>(this IEnumerable<TSource> source, Func<TSource, long> selector);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Decimal"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
        public static decimal? Average<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal?> selector);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Double"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static double? Average<TSource>(this IEnumerable<TSource> source, Func<TSource, double?> selector);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Int32"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum of the elements in the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static double? Average<TSource>(this IEnumerable<TSource> source, Func<TSource, int?> selector);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Int64"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        public static double? Average<TSource>(this IEnumerable<TSource> source, Func<TSource, long?> selector);
        /// <summary>Computes the average of a sequence of nullable <see cref="T:System.Single"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values, or null if the source sequence is empty or contains only values that are null.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static float? Average<TSource>(this IEnumerable<TSource> source, Func<TSource, float?> selector);
        /// <summary>Computes the average of a sequence of <see cref="T:System.Single"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values to calculate the average of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The average of the sequence of values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
```

```
    public static float Average<TSource>(this IEnumerable<TSource> source, Func<TSource, float> selector);
    /// <summary>Casts the elements of an <see cref="T:System.Collections.IEnumerable"></see> to the specified type.</summary>
    /// <param name="source">The <see cref="T:System.Collections.IEnumerable"></see> that contains the elements to be cast to type TResult.</param>
    /// <typeparam name="TResult">The type to cast the elements of source to.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains each element of the source sequence cast to the specified type.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidCastException">An element in the sequence cannot be cast to type <paramref name="TResult">TResult</paramref>.</exception>
    public static IEnumerable<TResult> Cast<TResult>(this IEnumerable source);
    /// <summary>Concatenates two sequences.</summary>
    /// <param name="first">The first sequence to concatenate.</param>
    /// <param name="second">The sequence to concatenate to the first sequence.</param>
    /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the concatenated elements of the two input sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
    public static IEnumerable<TSource> Concat<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second);
    /// <summary>Determines whether a sequence contains a specified element by using the default equality comparer.</summary>
    /// <param name="source">A sequence in which to locate a value.</param>
    /// <param name="value">The value to locate in the sequence.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>true if the source sequence contains an element that has the specified value; otherwise, false.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static bool Contains<TSource>(this IEnumerable<TSource> source, TSource value);
    /// <summary>Determines whether a sequence contains a specified element by using a specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</summary>
    /// <param name="source">A sequence in which to locate a value.</param>
    /// <param name="value">The value to locate in the sequence.</param>
    /// <param name="comparer">An equality comparer to compare values.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
```

```
        /// <returns>true if the source sequence contains an element that has the specified value; otherwise, false.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static bool Contains<TSource>(this IEnumerable<TSource> source, TSource value, IEqualityComparer<TSource> comparer);
        /// <summary>Returns the number of elements in a sequence.</summary>
        /// <param name="source">A sequence that contains elements to be counted.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The number of elements in the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The number of elements in <paramref name="source">source</paramref> is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
        public static int Count<TSource>(this IEnumerable<TSource> source);
        /// <summary>Returns a number that represents how many elements in the specified sequence satisfy a condition.</summary>
        /// <param name="source">A sequence that contains elements to be tested and counted.</param>
        /// <param name="predicate">A function to test each element for a condition.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>A number that represents how many elements in the sequence satisfy the condition in the predicate function.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The number of elements in <paramref name="source">source</paramref> is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
        public static int Count<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
        /// <summary>Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty.</summary>
        /// <param name="source">The sequence to return a default value for if it is empty.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> object that contains the default value for the <paramref name="TSource">TSource</paramref> type if <paramref name="source">source</paramref> is empty; otherwise, <paramref name="source">source</paramref>.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this IEnumerable<TSource> source);
```

```
    /// <summary>Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.</summary>
    /// <param name="source">The sequence to return the specified value for if it is empty.</param>
    /// <param name="defaultValue">The value to return if the sequence is empty.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains <paramref name="defaultValue">defaultValue</paramref> if <paramref name="source">source</paramref> is empty; otherwise, <paramref name="source">source</paramref>.</returns>
    public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this IEnumerable<TSource> source, TSource defaultValue);
    /// <summary>Returns distinct elements from a sequence by using the default equality comparer to compare values.</summary>
    /// <param name="source">The sequence to remove duplicate elements from.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains distinct elements from the source sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static IEnumerable<TSource> Distinct<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns distinct elements from a sequence by using a specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare values.</summary>
    /// <param name="source">The sequence to remove duplicate elements from.</param>
    /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare values.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains distinct elements from the source sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static IEnumerable<TSource> Distinct<TSource>(this IEnumerable<TSource> source, IEqualityComparer<TSource> comparer);
    /// <summary>Returns the element at a specified index in a sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an element from.</param>
    /// <param name="index">The zero-based index of the element to retrieve.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The element at the specified position in the source sequence.</
```

```
returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.ArgumentOutOfRangeException"><paramref name="index">index</paramref> is less than 0 or greater than or equal to the number of elements in <paramref name="source">source</paramref>.</exception>
        public static TSource ElementAt<TSource>(this IEnumerable<TSource> source, int index);
        /// <summary>Returns the element at a specified index in a sequence or a default value if the index is out of range.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an element from.</param>
        /// <param name="index">The zero-based index of the element to retrieve.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>default(<paramref name="TSource">TSource</paramref>) if the index is outside the bounds of the source sequence; otherwise, the element at the specified position in the source sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static TSource ElementAtOrDefault<TSource>(this IEnumerable<TSource> source, int index);
        /// <summary>Returns an empty <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that has the specified type argument.</summary>
        /// <typeparam name="TResult">The type to assign to the type parameter of the returned generic <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</typeparam>
        /// <returns>An empty <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose type argument is <paramref name="TResult">TResult</paramref>.</returns>
        public static IEnumerable<TResult> Empty<TResult>();
        /// <summary>Produces the set difference of two sequences by using the default equality comparer to compare values.</summary>
        /// <param name="first">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements that are not also in second will be returned.</param>
        /// <param name="second">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements that also occur in the first sequence will cause those elements to be removed from the returned sequence.</param>
        /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
        /// <returns>A sequence that contains the set difference of the elements of two sequences.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
        public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource>
         first, IEnumerable<TSource> second);
        /// <summary>Produces the set difference of two sequences by using the 
          specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></
          see> to compare values.</summary>
        /// <param name="first">An <see 
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements 
          that are not also in second will be returned.</param>
        /// <param name="second">An <see 
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements 
          that also occur in the first sequence will cause those elements to be 
          removed from the returned sequence.</param>
        /// <param name="comparer">An <see 
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare 
          values.</param>
        /// <typeparam name="TSource">The type of the elements of the input 
          sequences.</typeparam>
        /// <returns>A sequence that contains the set difference of the elements of 
          two sequences.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref 
          name="first">first</paramref> or <paramref name="second">second</paramref> 
           is null.</exception>
        public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource> 
           first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
        /// <summary>Returns the first element of a sequence.</summary>
        /// <param name="source">The <see 
          cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the 
          first element of.</param>
        /// <typeparam name="TSource">The type of the elements of source.</
          typeparam>
        /// <returns>The first element in the specified sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref 
          name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException">The source sequence 
           is empty.</exception>
        public static TSource First<TSource>(this IEnumerable<TSource> source);
        /// <summary>Returns the first element in a sequence that satisfies a 
          specified condition.</summary>
        /// <param name="source">An <see 
          cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an 
          element from.</param>
        /// <param name="predicate">A function to test each element for a 
          condition.</param>
        /// <typeparam name="TSource">The type of the elements of source.</
          typeparam>
        /// <returns>The first element in the sequence that passes the test in the 
          specified predicate function.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref 
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException">No element 
          satisfies the condition in <paramref name="predicate">predicate</
          paramref>.   -or-   The source sequence is empty.</exception>
```

```
    public static TSource First<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Returns the first element of a sequence, or a default value if the sequence contains no elements.</summary>
    /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the first element of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>default(<paramref name="TSource">TSource</paramref>) if <paramref name="source">source</paramref> is empty; otherwise, the first element in <paramref name="source">source</paramref>.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static TSource FirstOrDefault<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an element from.</param>
    /// <param name="predicate">A function to test each element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>default(<paramref name="TSource">TSource</paramref>) if <paramref name="source">source</paramref> is empty or if no element passes the test specified by <paramref name="predicate">predicate</paramref>; otherwise, the first element in <paramref name="source">source</paramref> that passes the test specified by <paramref name="predicate">predicate</paramref>.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static TSource FirstOrDefault<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Groups the elements of a sequence according to a specified key selector function.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
    /// <param name="keySelector">A function to extract the key for each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
    /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1" sourceendlinenumber="1">An <code>IEnumerable&lt;&gt;<_tkey2c_tsource="">&gt;</_tkey2c_></code> in C# or <code>IEnumerable(Of IGrouping(Of TKey, TSource))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> object contains a sequence of
```

```
objects and a key.</p>
        /// </returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1" sourceendlinenumber="1">An <code>IEnumerable&lt;&gt;<_tkey2c_tsource="">&gt;</_tkey2c_></code> in C# or <code>IEnumerable(Of IGrouping(Of TKey, TSource))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> object contains a collection of objects and a key.</p>
        /// </returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IEqualityComparer<TKey> comparer);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="elementSelector">A function to map each source element to an element in the <see cref="T:System.Linq.IGrouping`2"></see>.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the elements in the <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
        /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1"
```

```
sourceendlinenumber="1">An <code>IEnumerable&lt;&gt;<_tkey2c_telement="">&gt;</_tkey2c_></code> in C# or <code>IEnumerable(Of IGrouping(Of TKey, TElement))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> object contains a collection of objects of type <code data-dev-comment-type="paramref">TElement</code> and a key.</p>
        /// </returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.</exception>
        public static IEnumerable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector);
        /// <summary>Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the elements in the <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
        /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1" sourceendlinenumber="1">An <code>IEnumerable&lt;&gt;<_tkey2c_telement="">&gt;</_tkey2c_></code> in C# or <code>IEnumerable(Of IGrouping(Of TKey, TElement))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> object contains a collection of objects of type <code data-dev-comment-type="paramref">TElement</code> and a key.</p>
        /// </returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.</exception>
        public static IEnumerable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector, IEqualityComparer<TKey> comparer);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="resultSelector">A function to create a result value from each group.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
        /// <returns>A collection of elements of type <paramref name="TResult">TResult</paramref> where each element represents a projection over a group and its key.</returns>
        public static IEnumerable<TResult> GroupBy<TSource, TKey, TResult>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TKey, IEnumerable<TSource>, TResult> resultSelector);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="resultSelector">A function to create a result value from each group.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys with.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
        /// <returns>A collection of elements of type <paramref name="TResult">TResult</paramref> where each element represents a projection over a group and its key.</returns>
        public static IEnumerable<TResult> GroupBy<TSource, TKey, TResult>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TKey, IEnumerable<TSource>, TResult> resultSelector, IEqualityComparer<TKey> comparer);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
        /// <param name="resultSelector">A function to create a result value from each group.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the elements in each <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
        /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
        /// <returns>A collection of elements of type <paramref name="TResult">TResult</paramref> where each element represents a projection over a group and its key.</returns>
        public static IEnumerable<TResult> GroupBy<TSource, TKey, TElement, TResult>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector, Func<TKey, IEnumerable<TElement>, TResult> resultSelector);
        /// <summary>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
        /// <param name="keySelector">A function to extract the key for each element.</param>
        /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
        /// <param name="resultSelector">A function to create a result value from each group.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys with.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the elements in each <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
        /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
        /// <returns>A collection of elements of type <paramref name="TResult">TResult</paramref> where each element represents a projection over a group and its key.</returns>
        public static IEnumerable<TResult> GroupBy<TSource, TKey, TElement, TResult>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector, Func<TKey, IEnumerable<TElement>, TResult> resultSelector, IEqualityComparer<TKey> comparer);
```

```csharp
        /// <summary>Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.</summary>
        /// <param name="outer">The first sequence to join.</param>
        /// <param name="inner">The sequence to join to the first sequence.</param>
        /// <param name="outerKeySelector">A function to extract the join key from each element of the first sequence.</param>
        /// <param name="innerKeySelector">A function to extract the join key from each element of the second sequence.</param>
        /// <param name="resultSelector">A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.</param>
        /// <typeparam name="TOuter">The type of the elements of the first sequence.</typeparam>
        /// <typeparam name="TInner">The type of the elements of the second sequence.</typeparam>
        /// <typeparam name="TKey">The type of the keys returned by the key selector functions.</typeparam>
        /// <typeparam name="TResult">The type of the result elements.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains elements of type <paramref name="TResult">TResult</paramref> that are obtained by performing a grouped join on two sequences.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref name="innerKeySelector">innerKeySelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
        public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, IEnumerable<TInner>, TResult> resultSelector);
        /// <summary>Correlates the elements of two sequences based on key equality and groups the results. A specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> is used to compare keys.</summary>
        /// <param name="outer">The first sequence to join.</param>
        /// <param name="inner">The sequence to join to the first sequence.</param>
        /// <param name="outerKeySelector">A function to extract the join key from each element of the first sequence.</param>
        /// <param name="innerKeySelector">A function to extract the join key from each element of the second sequence.</param>
        /// <param name="resultSelector">A function to create a result element from an element from the first sequence and a collection of matching elements from the second sequence.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to hash and compare keys.</param>
        /// <typeparam name="TOuter">The type of the elements of the first sequence.</typeparam>
        /// <typeparam name="TInner">The type of the elements of the second sequence.</typeparam>
```

```
    /// <typeparam name="TKey">The type of the keys returned by the key selector functions.</typeparam>
    /// <typeparam name="TResult">The type of the result elements.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains elements of type <paramref name="TResult">TResult</paramref> that are obtained by performing a grouped join on two sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref name="innerKeySelector">innerKeySelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
    public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, IEnumerable<TInner>, TResult> resultSelector, IEqualityComparer<TKey> comparer);
    /// <summary>Produces the set intersection of two sequences by using the default equality comparer to compare values.</summary>
    /// <param name="first">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct elements that also appear in second will be returned.</param>
    /// <param name="second">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct elements that also appear in the first sequence will be returned.</param>
    /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
    /// <returns>A sequence that contains the elements that form the set intersection of two sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
    public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second);
    /// <summary>Produces the set intersection of two sequences by using the specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare values.</summary>
    /// <param name="first">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct elements that also appear in second will be returned.</param>
    /// <param name="second">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct elements that also appear in the first sequence will be returned.</param>
    /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare values.</param>
    /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
    /// <returns>A sequence that contains the elements that form the set intersection of two sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref>
```

```
      is null.</exception>
    public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
    /// <summary>Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.</summary>
    /// <param name="outer">The first sequence to join.</param>
    /// <param name="inner">The sequence to join to the first sequence.</param>
    /// <param name="outerKeySelector">A function to extract the join key from each element of the first sequence.</param>
    /// <param name="innerKeySelector">A function to extract the join key from each element of the second sequence.</param>
    /// <param name="resultSelector">A function to create a result element from two matching elements.</param>
    /// <typeparam name="TOuter">The type of the elements of the first sequence.</typeparam>
    /// <typeparam name="TInner">The type of the elements of the second sequence.</typeparam>
    /// <typeparam name="TKey">The type of the keys returned by the key selector functions.</typeparam>
    /// <typeparam name="TResult">The type of the result elements.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that has elements of type <paramref name="TResult">TResult</paramref> that are obtained by performing an inner join on two sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref name="innerKeySelector">innerKeySelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
    public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, TInner, TResult> resultSelector);
    /// <summary>Correlates the elements of two sequences based on matching keys. A specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> is used to compare keys.</summary>
    /// <param name="outer">The first sequence to join.</param>
    /// <param name="inner">The sequence to join to the first sequence.</param>
    /// <param name="outerKeySelector">A function to extract the join key from each element of the first sequence.</param>
    /// <param name="innerKeySelector">A function to extract the join key from each element of the second sequence.</param>
    /// <param name="resultSelector">A function to create a result element from two matching elements.</param>
    /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to hash and compare keys.</param>
    /// <typeparam name="TOuter">The type of the elements of the first sequence.</typeparam>
    /// <typeparam name="TInner">The type of the elements of the second sequence.</typeparam>
    /// <typeparam name="TKey">The type of the keys returned by the key selector functions.</typeparam>
    /// <typeparam name="TResult">The type of the result elements.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that has elements of type <paramref name="TResult">TResult</paramref> that are obtained by performing an inner join on two sequences.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref name="innerKeySelector">innerKeySelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
    public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, TInner, TResult> resultSelector, IEqualityComparer<TKey> comparer);
    /// <summary>Returns the last element of a sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the last element of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value at the last position in the source sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException">The source sequence is empty.</exception>
    public static TSource Last<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns the last element of a sequence that satisfies a specified condition.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an element from.</param>
    /// <param name="predicate">A function to test each element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The last element in the sequence that passes the test in the specified predicate function.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException">No element satisfies the condition in <paramref name="predicate">predicate</paramref>.   -or-   The source sequence is empty.</exception>
    public static TSource Last<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Returns the last element of a sequence, or a default value if the sequence contains no elements.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the last element of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>default(<paramref name="TSource">TSource</paramref>) if the source sequence is empty; otherwise, the last element in the <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static TSource LastOrDefault<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an element from.</param>
    /// <param name="predicate">A function to test each element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>default(<paramref name="TSource">TSource</paramref>) if the sequence is empty or if no elements pass the test in the predicate function; otherwise, the last element that passes the test in the predicate function.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static TSource LastOrDefault<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Returns an <see cref="T:System.Int64"></see> that represents the total number of elements in a sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements to be counted.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The number of elements in the source sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The number of elements exceeds <see cref="F:System.Int64.MaxValue"></see>.</exception>
    public static long LongCount<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns an <see cref="T:System.Int64"></see> that represents how many elements in a sequence satisfy a condition.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements to be counted.</param>
    /// <param name="predicate">A function to test each element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>A number that represents how many elements in the sequence satisfy the condition in the predicate function.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
```

```
        name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The number of matching elements exceeds <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static long LongCount<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
        /// <summary>Returns the maximum value in a sequence of <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Decimal"></see> values to determine the maximum value of.</param>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static decimal Max(this IEnumerable<decimal> source);
        /// <summary>Returns the maximum value in a sequence of <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Double"></see> values to determine the maximum value of.</param>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Max(this IEnumerable<double> source);
        /// <summary>Returns the maximum value in a sequence of <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int32"></see> values to determine the maximum value of.</param>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static int Max(this IEnumerable<int> source);
        /// <summary>Returns the maximum value in a sequence of <see cref="T:System.Int64"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int64"></see> values to determine the maximum value of.</param>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static long Max(this IEnumerable<long> source);
        /// <summary>Returns the maximum value in a sequence of nullable <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Decimal"></see> values to determine the maximum value of.</param>
        /// <returns>A value of type Nullable in C# or Nullable(Of Decimal) in
          Visual Basic that corresponds to the maximum value in the sequence.</returns>
      /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static decimal? Max(this IEnumerable<decimal?> source);
        /// <summary>Returns the maximum value in a sequence of nullable <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Double"></see> values to determine the maximum value of.</param>
        /// <returns>A value of type Nullable in C# or Nullable(Of Double) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static double? Max(this IEnumerable<double?> source);
        /// <summary>Returns the maximum value in a sequence of nullable <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></see> values to determine the maximum value of.</param>
        /// <returns>A value of type Nullable in C# or Nullable(Of Int32) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static int? Max(this IEnumerable<int?> source);
        /// <summary>Returns the maximum value in a sequence of nullable <see cref="T:System.Int64"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></see> values to determine the maximum value of.</param>
        /// <returns>A value of type Nullable in C# or Nullable(Of Int64) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static long? Max(this IEnumerable<long?> source);
        /// <summary>Returns the maximum value in a sequence of nullable <see cref="T:System.Single"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Single"></see> values to determine the maximum value of.</param>
        /// <returns>A value of type Nullable in C# or Nullable(Of Single) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static float? Max(this IEnumerable<float?> source);
        /// <summary>Returns the maximum value in a sequence of <see cref="T:System.Single"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Single"></see> values to determine the maximum value of.</param>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static float Max(this IEnumerable<float> source);
        /// <summary>Returns the maximum value in a generic sequence.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static TSource Max<TSource>(this IEnumerable<TSource> source);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum <see cref="T:System.Decimal"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static decimal Max<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum <see cref="T:System.Double"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Max<TSource>(this IEnumerable<TSource> source, Func<TSource, double> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum <see cref="T:System.Int32"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static int Max<TSource>(this IEnumerable<TSource> source, Func<TSource, int> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum <see cref="T:System.Int64"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static long Max<TSource>(this IEnumerable<TSource> source, Func<TSource, long> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum nullable <see cref="T:System.Decimal"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The value of type Nullable in C# or Nullable(Of Decimal) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static decimal? Max<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal?> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum nullable <see cref="T:System.Double"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The value of type Nullable in C# or Nullable(Of Double) in Visual Basic that corresponds to the maximum value in the sequence.</
```

```
returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static double? Max<TSource>(this IEnumerable<TSource> source, Func<TSource, double?> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum nullable <see cref="T:System.Int32"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The value of type Nullable in C# or Nullable(Of Int32) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static int? Max<TSource>(this IEnumerable<TSource> source, Func<TSource, int?> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum nullable <see cref="T:System.Int64"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The value of type Nullable in C# or Nullable(Of Int64) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static long? Max<TSource>(this IEnumerable<TSource> source, Func<TSource, long?> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the maximum nullable <see cref="T:System.Single"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the maximum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The value of type Nullable in C# or Nullable(Of Single) in Visual Basic that corresponds to the maximum value in the sequence.</returns>
```

```
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static float? Max<TSource>(this IEnumerable<TSource> source, Func<TSource, float?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the maximum <see cref="T:System.Single"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the maximum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The maximum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static float Max<TSource>(this IEnumerable<TSource> source, Func<TSource, float> selector);
    /// <summary>Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.</summary>
    /// <param name="source">A sequence of values to determine the maximum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TResult">The type of the value returned by selector.</typeparam>
    /// <returns>The maximum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static TResult Max<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, TResult> selector);
    /// <summary>Returns the minimum value in a sequence of <see cref="T:System.Decimal"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Decimal"></see> values to determine the minimum value of.</param>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static decimal Min(this IEnumerable<decimal> source);
    /// <summary>Returns the minimum value in a sequence of <see cref="T:System.Double"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Double"></see> values to determine the minimum value of.</param>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static double Min(this IEnumerable<double> source);
    /// <summary>Returns the minimum value in a sequence of <see cref="T:System.Int32"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Int32"></see> values to determine the minimum value of.</param>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static int Min(this IEnumerable<int> source);
    /// <summary>Returns the minimum value in a sequence of <see cref="T:System.Int64"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Int64"></see> values to determine the minimum value of.</param>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static long Min(this IEnumerable<long> source);
    /// <summary>Returns the minimum value in a sequence of nullable <see cref="T:System.Decimal"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see cref="T:System.Decimal"></see> values to determine the minimum value of.</param>
    /// <returns>A value of type Nullable in C# or Nullable(Of Decimal) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static decimal? Min(this IEnumerable<decimal?> source);
    /// <summary>Returns the minimum value in a sequence of nullable <see cref="T:System.Double"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see cref="T:System.Double"></see> values to determine the minimum value of.</param>
    /// <returns>A value of type Nullable in C# or Nullable(Of Double) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static double? Min(this IEnumerable<double?> source);
    /// <summary>Returns the minimum value in a sequence of nullable <see cref="T:System.Int32"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></see> values to determine the minimum value of.</param>
    /// <returns>A value of type Nullable in C# or Nullable(Of Int32) in Visual
```

```
        Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    public static int? Min(this IEnumerable<int?> source);
    /// <summary>Returns the minimum value in a sequence of nullable <see
      cref="T:System.Int64"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see
      cref="T:System.Int64"></see> values to determine the minimum value of.</
      param>
    /// <returns>A value of type Nullable in C# or Nullable(Of Int64) in Visual
      Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    public static long? Min(this IEnumerable<long?> source);
    /// <summary>Returns the minimum value in a sequence of nullable <see
      cref="T:System.Single"></see> values.</summary>
    /// <param name="source">A sequence of nullable <see
      cref="T:System.Single"></see> values to determine the minimum value of.</
      param>
    /// <returns>A value of type Nullable in C# or Nullable(Of Single) in Visual
       Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    public static float? Min(this IEnumerable<float?> source);
    /// <summary>Returns the minimum value in a sequence of <see
      cref="T:System.Single"></see> values.</summary>
    /// <param name="source">A sequence of <see cref="T:System.Single"></see>
      values to determine the minimum value of.</param>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref
      name="source">source</paramref> contains no elements.</exception>
    public static float Min(this IEnumerable<float> source);
    /// <summary>Returns the minimum value in a generic sequence.</summary>
    /// <param name="source">A sequence of values to determine the minimum value
       of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    public static TSource Min<TSource>(this IEnumerable<TSource> source);
    /// <summary>Invokes a transform function on each element of a sequence and
      returns the minimum <see cref="T:System.Decimal"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value
       of.</param>
    /// <param name="selector">A transform function to apply to each element.</
      param>
    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
    /// <returns>The minimum value in the sequence.</returns>
```

```
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static decimal Min<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the minimum <see cref="T:System.Double"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the minimum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The minimum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static double Min<TSource>(this IEnumerable<TSource> source, Func<TSource, double> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the minimum <see cref="T:System.Int32"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the minimum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The minimum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
        public static int Min<TSource>(this IEnumerable<TSource> source, Func<TSource, int> selector);
        /// <summary>Invokes a transform function on each element of a sequence and returns the minimum <see cref="T:System.Int64"></see> value.</summary>
        /// <param name="source">A sequence of values to determine the minimum value of.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The minimum value in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.InvalidOperationException"><paramref
```

```
name="source">source</paramref> contains no elements.</exception>
    public static long Min<TSource>(this IEnumerable<TSource> source, Func<TSource, long> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum nullable <see cref="T:System.Decimal"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value of type Nullable in C# or Nullable(Of Decimal) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static decimal? Min<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum nullable <see cref="T:System.Double"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value of type Nullable in C# or Nullable(Of Double) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static double? Min<TSource>(this IEnumerable<TSource> source, Func<TSource, double?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum nullable <see cref="T:System.Int32"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value of type Nullable in C# or Nullable(Of Int32) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static int? Min<TSource>(this IEnumerable<TSource> source, Func<TSource, int?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum nullable <see cref="T:System.Int64"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value of type Nullable in C# or Nullable(Of Int64) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static long? Min<TSource>(this IEnumerable<TSource> source, Func<TSource, long?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum nullable <see cref="T:System.Single"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The value of type Nullable in C# or Nullable(Of Single) in Visual Basic that corresponds to the minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static float? Min<TSource>(this IEnumerable<TSource> source, Func<TSource, float?> selector);
    /// <summary>Invokes a transform function on each element of a sequence and returns the minimum <see cref="T:System.Single"></see> value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException"><paramref name="source">source</paramref> contains no elements.</exception>
    public static float Min<TSource>(this IEnumerable<TSource> source, Func<TSource, float> selector);
    /// <summary>Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.</summary>
    /// <param name="source">A sequence of values to determine the minimum value of.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TResult">The type of the value returned by selector.</typeparam>
    /// <returns>The minimum value in the sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static TResult Min<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, TResult> selector);
    /// <summary>Filters the elements of an <see cref="T:System.Collections.IEnumerable"></see> based on a specified type.</summary>
    /// <param name="source">The <see cref="T:System.Collections.IEnumerable"></see> whose elements to filter.</param>
    /// <typeparam name="TResult">The type to filter the elements of the sequence on.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains elements from the input sequence of type <paramref name="TResult">TResult</paramref>.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static IEnumerable<TResult> OfType<TResult>(this IEnumerable source);
    /// <summary>Sorts the elements of a sequence in ascending order according to a key.</summary>
    /// <param name="source">A sequence of values to order.</param>
    /// <param name="keySelector">A function to extract a key from an element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted according to a key.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
    public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
    /// <summary>Sorts the elements of a sequence in ascending order by using a specified comparer.</summary>
    /// <param name="source">A sequence of values to order.</param>
    /// <param name="keySelector">A function to extract a key from an element.</param>
    /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
```

```
        param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IComparer<TKey> comparer);
        /// <summary>Sorts the elements of a sequence in descending order according to a key.</summary>
        /// <param name="source">A sequence of values to order.</param>
        /// <param name="keySelector">A function to extract a key from an element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted in descending order according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> OrderByDescending<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
        /// <summary>Sorts the elements of a sequence in descending order by using a specified comparer.</summary>
        /// <param name="source">A sequence of values to order.</param>
        /// <param name="keySelector">A function to extract a key from an element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted in descending order according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> OrderByDescending<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IComparer<TKey> comparer);
        /// <param name="source"></param>
        /// <param name="element"></param>
        /// <typeparam name="TSource"></typeparam>
```

```
    /// <returns></returns>
    public static IEnumerable<TSource> Prepend<TSource>(this IEnumerable<TSource> source, TSource element);
    /// <summary>Generates a sequence of integral numbers within a specified range.</summary>
    /// <param name="start">The value of the first integer in the sequence.</param>
    /// <param name="count">The number of sequential integers to generate.</param>
    /// <returns>An IEnumerable in C# or IEnumerable(Of Int32) in Visual Basic that contains a range of sequential integral numbers.</returns>
    /// <exception cref="T:System.ArgumentOutOfRangeException"><paramref name="count">count</paramref> is less than 0.   -or-  <paramref name="start">start</paramref> + <paramref name="count">count</paramref> -1 is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
    public static IEnumerable<int> Range(int start, int count);
    /// <summary>Generates a sequence that contains one repeated value.</summary>
    /// <param name="element">The value to be repeated.</param>
    /// <param name="count">The number of times to repeat the value in the generated sequence.</param>
    /// <typeparam name="TResult">The type of the value to be repeated in the result sequence.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains a repeated value.</returns>
    /// <exception cref="T:System.ArgumentOutOfRangeException"><paramref name="count">count</paramref> is less than 0.</exception>
    public static IEnumerable<TResult> Repeat<TResult>(TResult element, int count);
    /// <summary>Inverts the order of the elements in a sequence.</summary>
    /// <param name="source">A sequence of values to reverse.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>A sequence whose elements correspond to those of the input sequence in reverse order.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static IEnumerable<TSource> Reverse<TSource>(this IEnumerable<TSource> source);
    /// <summary>Projects each element of a sequence into a new form.</summary>
    /// <param name="source">A sequence of values to invoke a transform function on.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TResult">The type of the value returned by selector.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the transform function on each element of <paramref name="source">source</paramref>.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
```

```
name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static IEnumerable<TResult> Select<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, TResult> selector);
        /// <summary>Projects each element of a sequence into a new form by incorporating the element's index.</summary>
        /// <param name="source">A sequence of values to invoke a transform function on.</param>
        /// <param name="selector">A transform function to apply to each source element; the second parameter of the function represents the index of the source element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TResult">The type of the value returned by selector.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the transform function on each element of <paramref name="source">source</paramref>.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static IEnumerable<TResult> Select<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, int, TResult> selector);
        /// <summary>Projects each element of a sequence to an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> and flattens the resulting sequences into one sequence.</summary>
        /// <param name="source">A sequence of values to project.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TResult">The type of the elements of the sequence returned by selector.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the one-to-many transform function on each element of the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static IEnumerable<TResult> SelectMany<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, IEnumerable<TResult>> selector);
        /// <summary>Projects each element of a sequence to an <see cref="T:System.Collections.Generic.IEnumerable`1"></see>, and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.</summary>
        /// <param name="source">A sequence of values to project.</param>
        /// <param name="selector">A transform function to apply to each source element; the second parameter of the function represents the index of the source element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TResult">The type of the elements of the sequence returned by selector.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the one-to-many transform function on each element of an input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        public static IEnumerable<TResult> SelectMany<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, int, IEnumerable<TResult>> selector);
        /// <summary>Projects each element of a sequence to an <see cref="T:System.Collections.Generic.IEnumerable`1"></see>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein.</summary>
        /// <param name="source">A sequence of values to project.</param>
        /// <param name="collectionSelector">A transform function to apply to each element of the input sequence.</param>
        /// <param name="resultSelector">A transform function to apply to each element of the intermediate sequence.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TCollection">The type of the intermediate elements collected by collectionSelector.</typeparam>
        /// <typeparam name="TResult">The type of the elements of the resulting sequence.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the one-to-many transform function <paramref name="collectionSelector">collectionSelector</paramref> on each element of <paramref name="source">source</paramref> and then mapping each of those sequence elements and their corresponding source element to a result element.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="collectionSelector">collectionSelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
        public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(this IEnumerable<TSource> source, Func<TSource, IEnumerable<TCollection>> collectionSelector, Func<TSource, TCollection, TResult> resultSelector);
        /// <summary>Projects each element of a sequence to an <see cref="T:System.Collections.Generic.IEnumerable`1"></see>, flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element.</summary>
        /// <param name="source">A sequence of values to project.</param>
        /// <param name="collectionSelector">A transform function to apply to each source element; the second parameter of the function represents the index of the source element.</param>
        /// <param name="resultSelector">A transform function to apply to each element of the intermediate sequence.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
```

```csharp
        /// <typeparam name="TCollection">The type of the intermediate elements collected by collectionSelector.</typeparam>
        /// <typeparam name="TResult">The type of the elements of the resulting sequence.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements are the result of invoking the one-to-many transform function <paramref name="collectionSelector">collectionSelector</paramref> on each element of <paramref name="source">source</paramref> and then mapping each of those sequence elements and their corresponding source element to a result element.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="collectionSelector">collectionSelector</paramref> or <paramref name="resultSelector">resultSelector</paramref> is null.</exception>
        public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(this IEnumerable<TSource> source, Func<TSource, int, IEnumerable<TCollection>> collectionSelector, Func<TSource, TCollection, TResult> resultSelector);
        /// <summary>Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.</summary>
        /// <param name="first">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to second.</param>
        /// <param name="second">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to the first sequence.</param>
        /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
        /// <returns>true if the two source sequences are of equal length and their corresponding elements are equal according to the default equality comparer for their type; otherwise, false.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
        public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second);
        /// <summary>Determines whether two sequences are equal by comparing their elements by using a specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</summary>
        /// <param name="first">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to second.</param>
        /// <param name="second">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to the first sequence.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to use to compare elements.</param>
        /// <typeparam name="TSource">The type of the elements of the input sequences.</typeparam>
        /// <returns>true if the two source sequences are of equal length and their corresponding elements compare equal according to <paramref
```

```
name="comparer">comparer</paramref>; otherwise, false.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
    public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
    /// <summary>Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the single element of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The single element of the input sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException">The input sequence contains more than one element.   -or-   The input sequence is empty.</exception>
    public static TSource Single<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return a single element from.</param>
    /// <param name="predicate">A function to test an element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The single element of the input sequence that satisfies a condition.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException">No element satisfies the condition in <paramref name="predicate">predicate</paramref>.   -or-   More than one element satisfies the condition in <paramref name="predicate">predicate</paramref>.   -or-   The source sequence is empty.</exception>
    public static TSource Single<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the single element of.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The single element of the input sequence, or default(<paramref name="TSource">TSource</paramref>) if the sequence contains no elements.</
```

```
returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    /// <exception cref="T:System.InvalidOperationException">The input sequence contains more than one element.</exception>
    public static TSource SingleOrDefault<TSource>(this IEnumerable<TSource> source);
    /// <summary>Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return a single element from.</param>
    /// <param name="predicate">A function to test an element for a condition.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The single element of the input sequence that satisfies the condition, or default(<paramref name="TSource">TSource</paramref>) if no such element is found.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static TSource SingleOrDefault<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Bypasses a specified number of elements in a sequence and then returns the remaining elements.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return elements from.</param>
    /// <param name="count">The number of elements to skip before returning the remaining elements.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements that occur after the specified index in the input sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
    public static IEnumerable<TSource> Skip<TSource>(this IEnumerable<TSource> source, int count);
    /// <param name="source"></param>
    /// <param name="count"></param>
    /// <typeparam name="TSource"></typeparam>
    /// <returns></returns>
    public static IEnumerable<TSource> SkipLast<TSource>(this IEnumerable<TSource> source, int count);
    /// <summary>Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.</summary>
    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return
```

```
          elements from.</param>
        /// <param name="predicate">A function to test each element for a condition.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by <paramref name="predicate">predicate</paramref>.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        public static IEnumerable<TSource> SkipWhile<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
        /// <summary>Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to return elements from.</param>
        /// <param name="predicate">A function to test each source element for a condition; the second parameter of the function represents the index of the source element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements from the input sequence starting at the first element in the linear series that does not pass the test specified by <paramref name="predicate">predicate</paramref>.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        public static IEnumerable<TSource> SkipWhile<TSource>(this IEnumerable<TSource> source, Func<TSource, int, bool> predicate);
        /// <summary>Computes the sum of a sequence of <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Decimal"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
        public static decimal Sum(this IEnumerable<decimal> source);
        /// <summary>Computes the sum of a sequence of <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Double"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static double Sum(this IEnumerable<double> source);
        /// <summary>Computes the sum of a sequence of <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int32"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
        public static int Sum(this IEnumerable<int> source);
        /// <summary>Computes the sum of a sequence of <see cref="T:System.Int64"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Int64"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static long Sum(this IEnumerable<long> source);
        /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Decimal"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Decimal"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
        public static decimal? Sum(this IEnumerable<decimal?> source);
        /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Double"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Double"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static double? Sum(this IEnumerable<double?> source);
        /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Int32"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
        public static int? Sum(this IEnumerable<int?> source);
        /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Int64"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
        public static long? Sum(this IEnumerable<long?> source);
        /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Single"></see> values.</summary>
        /// <param name="source">A sequence of nullable <see cref="T:System.Single"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static float? Sum(this IEnumerable<float?> source);
        /// <summary>Computes the sum of a sequence of <see cref="T:System.Single"></see> values.</summary>
        /// <param name="source">A sequence of <see cref="T:System.Single"></see> values to calculate the sum of.</param>
        /// <returns>The sum of the values in the sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static float Sum(this IEnumerable<float> source);
        /// <summary>Computes the sum of the sequence of <see cref="T:System.Decimal"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values that are used to calculate a sum.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The sum of the projected values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
        /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
        public static decimal Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, decimal> selector);
        /// <summary>Computes the sum of the sequence of <see cref="T:System.Double"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
        /// <param name="source">A sequence of values that are used to calculate a sum.</param>
        /// <param name="selector">A transform function to apply to each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>The sum of the projected values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
```

```
    public static double Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, double> selector);
    /// <summary>Computes the sum of the sequence of <see cref="T:System.Int32"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
    public static int Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, int> selector);
    /// <summary>Computes the sum of the sequence of <see cref="T:System.Int64"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
    public static long Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, long> selector);
    /// <summary>Computes the sum of the sequence of nullable <see cref="T:System.Decimal"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
    public static decimal? Sum<TSource>(this IEnumerable<TSource> source,
      Func<TSource, decimal?> selector);
    /// <summary>Computes the sum of the sequence of nullable <see cref="T:System.Double"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    public static double? Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, double?> selector);
    /// <summary>Computes the sum of the sequence of nullable <see cref="T:System.Int32"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
    public static int? Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, int?> selector);
    /// <summary>Computes the sum of the sequence of nullable <see cref="T:System.Int64"></see> values that are obtained by invoking a transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a sum.</param>
    /// <param name="selector">A transform function to apply to each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
    public static long? Sum<TSource>(this IEnumerable<TSource> source, Func<TSource, long?> selector);
    /// <summary>Computes the sum of the sequence of nullable <see cref="T:System.Single"></see> values that are obtained by invoking a
```

```
      transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a
      sum.</param>
    /// <param name="selector">A transform function to apply to each element.</
      param>
    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> or <paramref name="selector">selector</
      paramref> is null.</exception>
    public static float? Sum<TSource>(this IEnumerable<TSource> source,
      Func<TSource, float?> selector);
    /// <summary>Computes the sum of the sequence of <see
      cref="T:System.Single"></see> values that are obtained by invoking a
      transform function on each element of the input sequence.</summary>
    /// <param name="source">A sequence of values that are used to calculate a
      sum.</param>
    /// <param name="selector">A transform function to apply to each element.</
      param>
    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
    /// <returns>The sum of the projected values.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> or <paramref name="selector">selector</
      paramref> is null.</exception>
    public static float Sum<TSource>(this IEnumerable<TSource> source,
      Func<TSource, float> selector);
    /// <summary>Returns a specified number of contiguous elements from the
      start of a sequence.</summary>
    /// <param name="source">The sequence to return elements from.</param>
    /// <param name="count">The number of elements to return.</param>
    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
      see> that contains the specified number of elements from the start of the
      input sequence.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
    public static IEnumerable<TSource> Take<TSource>(this IEnumerable<TSource>
      source, int count);
    /// <param name="source"></param>
    /// <param name="count"></param>
    /// <typeparam name="TSource"></typeparam>
    /// <returns></returns>
    public static IEnumerable<TSource> TakeLast<TSource>(this
      IEnumerable<TSource> source, int count);
    /// <summary>Returns elements from a sequence as long as a specified
      condition is true.</summary>
    /// <param name="source">A sequence to return elements from.</param>
    /// <param name="predicate">A function to test each element for a
      condition.</param>
```

```csharp
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the elements from the input sequence that occur before the element at which the test no longer passes.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static IEnumerable<TSource> TakeWhile<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
    /// <summary>Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.</summary>
    /// <param name="source">The sequence to return elements from.</param>
    /// <param name="predicate">A function to test each source element for a condition; the second parameter of the function represents the index of the source element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains elements from the input sequence that occur before the element at which the test no longer passes.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
    public static IEnumerable<TSource> TakeWhile<TSource>(this IEnumerable<TSource> source, Func<TSource, int, bool> predicate);
    /// <summary>Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.</summary>
    /// <param name="source">An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to sort.</param>
    /// <param name="keySelector">A function to extract a key from each element.</param>
    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
    /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted according to a key.</returns>
    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
    public static IOrderedEnumerable<TSource> ThenBy<TSource, TKey>(this IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector);
    /// <summary>Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.</summary>
    /// <param name="source">An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to sort.</param>
    /// <param name="keySelector">A function to extract a key from each element.</param>
```

```
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> ThenBy<TSource, TKey>(this IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector, IComparer<TKey> comparer);
        /// <summary>Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.</summary>
        /// <param name="source">An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to sort.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted in descending order according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> ThenByDescending<TSource, TKey>(this IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector);
        /// <summary>Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.</summary>
        /// <param name="source">An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to sort.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted in descending order according to a key.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref
```

```
name="keySelector">keySelector</paramref> is null.</exception>
        public static IOrderedEnumerable<TSource> ThenByDescending<TSource, TKey>(this IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector, IComparer<TKey> comparer);
        /// <summary>Creates an array from a <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create an array from.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>An array that contains the elements from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static TSource[] ToArray<TSource>(this IEnumerable<TSource> source);
        /// <summary>Creates a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see> that contains keys and values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.   -or-   <paramref name="keySelector">keySelector</paramref> produces a key that is null.</exception>
        /// <exception cref="T:System.ArgumentException"><paramref name="keySelector">keySelector</paramref> produces duplicate keys for two elements.</exception>
        public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
        /// <summary>Creates a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function and key comparer.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
```

```
keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the keys returned by keySelector.</typeparam>
        /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see> that contains keys and values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.   -or-  <paramref name="keySelector">keySelector</paramref> produces a key that is null.</exception>
        /// <exception cref="T:System.ArgumentException"><paramref name="keySelector">keySelector</paramref> produces duplicate keys for two elements.</exception>
        public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IEqualityComparer<TKey> comparer);
        /// <summary>Creates a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to specified key selector and element selector functions.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="elementSelector">A transform function to produce a result element value from each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the value returned by elementSelector.</typeparam>
        /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see> that contains values of type <paramref name="TElement">TElement</paramref> selected from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.   -or-  <paramref name="keySelector">keySelector</paramref> produces a key that is null.</exception>
        /// <exception cref="T:System.ArgumentException"><paramref name="keySelector">keySelector</paramref> produces duplicate keys for two elements.</exception>
        public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey, TElement>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector);
        /// <summary>Creates a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from an <see
```

```
cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function, a comparer, and an element selector function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Collections.Generic.Dictionary`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="elementSelector">A transform function to produce a result element value from each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the value returned by elementSelector.</typeparam>
        /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see> that contains values of type <paramref name="TElement">TElement</paramref> selected from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.   -or-  <paramref name="keySelector">keySelector</paramref> produces a key that is null.</exception>
        /// <exception cref="T:System.ArgumentException"><paramref name="keySelector">keySelector</paramref> produces duplicate keys for two elements.</exception>
        public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey, TElement>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector, IEqualityComparer<TKey> comparer);
        /// <param name="source"></param>
        /// <typeparam name="TSource"></typeparam>
        /// <returns></returns>
        public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource> source);
        /// <param name="source"></param>
        /// <param name="comparer"></param>
        /// <typeparam name="TSource"></typeparam>
        /// <returns></returns>
        public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource> source, IEqualityComparer<TSource> comparer);
        /// <summary>Creates a <see cref="T:System.Collections.Generic.List`1"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
        /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Collections.Generic.List`1"></see> from.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>A <see cref="T:System.Collections.Generic.List`1"></see> that contains elements from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
        public static List<TSource> ToList<TSource>(this IEnumerable<TSource> source);
        /// <summary>Creates a <see cref="T:System.Linq.Lookup`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function.</summary>
        /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Linq.Lookup`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains keys and values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
        /// <summary>Creates a <see cref="T:System.Linq.Lookup`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function and key comparer.</summary>
        /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Linq.Lookup`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains keys and values.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
        public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IEqualityComparer<TKey> comparer);
        /// <summary>Creates a <see cref="T:System.Linq.Lookup`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to
```

```
 specified key selector and element selector functions.</summary>
        /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Linq.Lookup`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="elementSelector">A transform function to produce a result element value from each element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the value returned by elementSelector.</typeparam>
        /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains values of type <paramref name="TElement">TElement</paramref> selected from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.</exception>
        public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TSource, TElement> elementSelector);
        /// <summary>Creates a <see cref="T:System.Linq.Lookup`2"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to a specified key selector function, a comparer and an element selector function.</summary>
        /// <param name="source">The <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see cref="T:System.Linq.Lookup`2"></see> from.</param>
        /// <param name="keySelector">A function to extract a key from each element.</param>
        /// <param name="elementSelector">A transform function to produce a result element value from each element.</param>
        /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
        /// <typeparam name="TElement">The type of the value returned by elementSelector.</typeparam>
        /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains values of type <paramref name="TElement">TElement</paramref> selected from the input sequence.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.</exception>
        public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>(this
             IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
            Func<TSource, TElement> elementSelector, IEqualityComparer<TKey>
            comparer);
        /// <summary>Produces the set union of two sequences by using the default
            equality comparer.</summary>
        /// <param name="first">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
            elements form the first set for the union.</param>
        /// <param name="second">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
            elements form the second set for the union.</param>
        /// <typeparam name="TSource">The type of the elements of the input
            sequences.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
            see> that contains the elements from both input sequences, excluding
            duplicates.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref
            name="first">first</paramref> or <paramref name="second">second</paramref>
             is null.</exception>
        public static IEnumerable<TSource> Union<TSource>(this IEnumerable<TSource>
            first, IEnumerable<TSource> second);
        /// <summary>Produces the set union of two sequences by using a specified
            <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</
            summary>
        /// <param name="first">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
            elements form the first set for the union.</param>
        /// <param name="second">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
            elements form the second set for the union.</param>
        /// <param name="comparer">The <see
            cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
            values.</param>
        /// <typeparam name="TSource">The type of the elements of the input
            sequences.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
            see> that contains the elements from both input sequences, excluding
            duplicates.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref
            name="first">first</paramref> or <paramref name="second">second</paramref>
             is null.</exception>
        public static IEnumerable<TSource> Union<TSource>(this IEnumerable<TSource>
            first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
        /// <summary>Filters a sequence of values based on a predicate.</summary>
        /// <param name="source">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> to filter.</
            param>
        /// <param name="predicate">A function to test each element for a
            condition.</param>
        /// <typeparam name="TSource">The type of the elements of source.</
            typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
```

```
see> that contains elements from the input sequence that satisfy the condition.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);
        /// <summary>Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.</summary>
        /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> to filter.</param>
        /// <param name="predicate">A function to test each source element for a condition; the second parameter of the function represents the index of the source element.</param>
        /// <typeparam name="TSource">The type of the elements of source.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains elements from the input sequence that satisfy the condition.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
        public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, int, bool> predicate);
        /// <summary>Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.</summary>
        /// <param name="first">The first sequence to merge.</param>
        /// <param name="second">The second sequence to merge.</param>
        /// <param name="resultSelector">A function that specifies how to merge the elements from the two sequences.</param>
        /// <typeparam name="TFirst">The type of the elements of the first input sequence.</typeparam>
        /// <typeparam name="TSecond">The type of the elements of the second input sequence.</typeparam>
        /// <typeparam name="TResult">The type of the elements of the result sequence.</typeparam>
        /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains merged elements of two input sequences.</returns>
        /// <exception cref="T:System.ArgumentNullException"><paramref name="first">first</paramref> or <paramref name="second">second</paramref> is null.</exception>
        public static IEnumerable<TResult> Zip<TFirst, TSecond, TResult>(this IEnumerable<TFirst> first, IEnumerable<TSecond> second, Func<TFirst, TSecond, TResult> resultSelector);
    }
}
```