

```
1 // Decompiled with JetBrains decompiler
2 // Type: System.Linq.Enumerable
3 // Assembly: System.Linq, Version=4.2.0.0, Culture=neutral,
4 // MVID: 6BC5D5C3-B137-47AC-970B-82F3552008E0
5 // Assembly location: C:\Program Files\dotnet\sdk\NuGetFallbackFolder
6 // \microsoft.netcore.app\2.0.0\ref\netcoreapp2.0\System.Linq.dll
7 using System.Collections;
8 using System.Collections.Generic;
9
10 namespace System.Linq
11 {
12     /// <summary>Provides a set of static (Shared in Visual Basic) methods for
13     /// querying objects that implement <see
14     /// cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
15     public static class Enumerable
16     {
17         /// <summary>Applies an accumulator function over a sequence.</summary>
18         /// <param name="source">An <see
19         /// cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate
20         /// over.</param>
21         /// <param name="func">An accumulator function to be invoked on each
22         /// element.</param>
23         /// <typeparam name="TSource">The type of the elements of source.</
24         /// typeparam>
25         /// <returns>The final accumulator value.</returns>
26         /// <exception cref="T:System.ArgumentNullException"><paramref
27         /// name="source">source</paramref> or <paramref name="func">func</paramref>
28         /// is null.</exception>
29         /// <exception cref="T:System.InvalidOperationException"><paramref
30         /// name="source">source</paramref> contains no elements.</exception>
31         public static TSource Aggregate<TSource>(this IEnumerable<TSource> source,
32         Func<TSource, TSource, TSource> func);
33         /// <summary>Applies an accumulator function over a sequence. The specified
34         /// seed value is used as the initial accumulator value.</summary>
35         /// <param name="source">An <see
36         /// cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate
37         /// over.</param>
38         /// <param name="seed">The initial accumulator value.</param>
39         /// <param name="func">An accumulator function to be invoked on each
40         /// element.</param>
41         /// <typeparam name="TSource">The type of the elements of source.</
42         /// typeparam>
43         /// <typeparam name="TAccumulate">The type of the accumulator value.</
44         /// typeparam>
45         /// <returns>The final accumulator value.</returns>
46         /// <exception cref="T:System.ArgumentNullException"><paramref
47         /// name="source">source</paramref> or <paramref name="func">func</paramref>
48         /// is null.</exception>
49         public static TAccumulate Aggregate<TSource, TAccumulate>(this
50         IEnumerable<TSource> source, TAccumulate seed, Func<TAccumulate, TSource,
```

```

    TAccumulate> func);
32    /// <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>
33    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to aggregate
    over.</param>
34    /// <param name="seed">The initial accumulator value.</param>
35    /// <param name="func">An accumulator function to be invoked on each
    element.</param>
36    /// <param name="resultSelector">A function to transform the final
    accumulator value into the result value.</param>
37    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
38    /// <typeparam name="TAccumulate">The type of the accumulator value.</
    typeparam>
39    /// <typeparam name="TResult">The type of the resulting value.</typeparam>
40    /// <returns>The transformed final accumulator value.</returns>
41    /// <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>
42    public static TResult Aggregate<TSource, TAccumulate, TResult>(this
    IEnumerable<TSource> source, TAccumulate seed, Func<TAccumulate, TSource,
    TAccumulate> func, Func<TAccumulate, TResult> resultSelector);
43    /// <summary>Determines whether all elements of a sequence satisfy a
    condition.</summary>
44    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> that contains the
    elements to apply the predicate to.</param>
45    /// <param name="predicate">A function to test each element for a
    condition.</param>
46    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
47    /// <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>
48    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
49    public static bool All<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
50    /// <summary>Determines whether a sequence contains any elements.</summary>
51    /// <param name="source">The <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to check for
    emptiness.</param>
52    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
53    /// <returns>true if the source sequence contains any elements; otherwise,
    false.</returns>
54    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>

```

```

55 public static bool Any<TSource>(this IEnumerable<TSource> source);
56 /// <summary>Determines whether any element of a sequence satisfies a
    condition.</summary>
57 /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
    apply the predicate to.</param>
58 /// <param name="predicate">A function to test each element for a
    condition.</param>
59 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
60 /// <returns>true if any elements in the source sequence pass the test in
    the specified predicate; otherwise, false.</returns>
61 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
62 public static bool Any<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
63 /// <param name="source"></param>
64 /// <param name="element"></param>
65 /// <typeparam name="TSource"></typeparam>
66 /// <returns></returns>
67 public static IEnumerable<TSource> Append<TSource>(this IEnumerable<TSource>
    source, TSource element);
68 /// <summary>Returns the input typed as <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
69 /// <param name="source">The sequence to type as <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see>.</param>
70 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
71 /// <returns>The input sequence typed as <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see>.</returns>
72 public static IEnumerable<TSource> AsEnumerable<TSource>(this
    IEnumerable<TSource> source);
73 /// <summary>Computes the average of a sequence of <see
    cref="T:System.Decimal"></see> values.</summary>
74 /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>
    values to calculate the average of.</param>
75 /// <returns>The average of the sequence of values.</returns>
76 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
77 /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
78 public static decimal Average(this IEnumerable<decimal> source);
79 /// <summary>Computes the average of a sequence of <see
    cref="T:System.Double"></see> values.</summary>
80 /// <param name="source">A sequence of <see cref="T:System.Double"></see>
    values to calculate the average of.</param>
81 /// <returns>The average of the sequence of values.</returns>
82 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
83 /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>

```

```
84 public static double Average(this IEnumerable<double> source);
85 /// <summary>Computes the average of a sequence of <see
   cref="T:System.Int32"></see> values.</summary>
86 /// <param name="source">A sequence of <see cref="T:System.Int32"></see>
   values to calculate the average of.</param>
87 /// <returns>The average of the sequence of values.</returns>
88 /// <exception cref="T:System.ArgumentNullException"><paramref
   name="source">source</paramref> is null.</exception>
89 /// <exception cref="T:System.InvalidOperationException"><paramref
   name="source">source</paramref> contains no elements.</exception>
90 public static double Average(this IEnumerable<int> source);
91 /// <summary>Computes the average of a sequence of <see
   cref="T:System.Int64"></see> values.</summary>
92 /// <param name="source">A sequence of <see cref="T:System.Int64"></see>
   values to calculate the average of.</param>
93 /// <returns>The average of the sequence of values.</returns>
94 /// <exception cref="T:System.ArgumentNullException"><paramref
   name="source">source</paramref> is null.</exception>
95 /// <exception cref="T:System.InvalidOperationException"><paramref
   name="source">source</paramref> contains no elements.</exception>
96 public static double Average(this IEnumerable<long> source);
97 /// <summary>Computes the average of a sequence of nullable <see
   cref="T:System.Decimal"></see> values.</summary>
98 /// <param name="source">A sequence of nullable <see
   cref="T:System.Decimal"></see> values to calculate the average of.</param>
99 /// <returns>The average of the sequence of values, or null if the source
   sequence is empty or contains only values that are null.</returns>
100 /// <exception cref="T:System.ArgumentNullException"><paramref
   name="source">source</paramref> is null.</exception>
101 /// <exception cref="T:System.OverflowException">The sum of the elements in
   the sequence is larger than <see cref="F:System.Decimal.MaxValue"></
   see>.</exception>
102 public static decimal? Average(this IEnumerable<decimal?> source);
103 /// <summary>Computes the average of a sequence of nullable <see
   cref="T:System.Double"></see> values.</summary>
104 /// <param name="source">A sequence of nullable <see
   cref="T:System.Double"></see> values to calculate the average of.</param>
105 /// <returns>The average of the sequence of values, or null if the source
   sequence is empty or contains only values that are null.</returns>
106 /// <exception cref="T:System.ArgumentNullException"><paramref
   name="source">source</paramref> is null.</exception>
107 public static double? Average(this IEnumerable<double?> source);
108 /// <summary>Computes the average of a sequence of nullable <see
   cref="T:System.Int32"></see> values.</summary>
109 /// <param name="source">A sequence of nullable <see
   cref="T:System.Int32"></see> values to calculate the average of.</param>
110 /// <returns>The average of the sequence of values, or null if the source
   sequence is empty or contains only values that are null.</returns>
111 /// <exception cref="T:System.ArgumentNullException"><paramref
   name="source">source</paramref> is null.</exception>
112 /// <exception cref="T:System.OverflowException">The sum of the elements in
   the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</
```

```

        exception>
113     public static double? Average(this IEnumerable<int?> source);
114     /// <summary>Computes the average of a sequence of nullable <see
        cref="T:System.Int64"></see> values.</summary>
115     /// <param name="source">A sequence of nullable <see
        cref="T:System.Int64"></see> values to calculate the average of.</param>
116     /// <returns>The average of the sequence of values, or null if the source
        sequence is empty or contains only values that are null.</returns>
117     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
118     /// <exception cref="T:System.OverflowException">The sum of the elements in
        the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</
        exception>
119     public static double? Average(this IEnumerable<long?> source);
120     /// <summary>Computes the average of a sequence of nullable <see
        cref="T:System.Single"></see> values.</summary>
121     /// <param name="source">A sequence of nullable <see
        cref="T:System.Single"></see> values to calculate the average of.</param>
122     /// <returns>The average of the sequence of values, or null if the source
        sequence is empty or contains only values that are null.</returns>
123     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
124     public static float? Average(this IEnumerable<float?> source);
125     /// <summary>Computes the average of a sequence of <see
        cref="T:System.Single"></see> values.</summary>
126     /// <param name="source">A sequence of <see cref="T:System.Single"></see>
        values to calculate the average of.</param>
127     /// <returns>The average of the sequence of values.</returns>
128     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
129     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
130     public static float Average(this IEnumerable<float> source);
131     /// <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>
132     /// <param name="source">A sequence of values that are used to calculate an
        average.</param>
133     /// <param name="selector">A transform function to apply to each element.</
        param>
134     /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
135     /// <returns>The average of the sequence of values.</returns>
136     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
137     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
138     /// <exception cref="T:System.OverflowException">The sum of the elements in
        the sequence is larger than <see cref="F:System.Decimal.MaxValue"></
        see>.</exception>
139     public static decimal Average<TSource>(this IEnumerable<TSource> source,

```

```

    Func<TSource, decimal> selector);
140    /// <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>
141    /// <param name="source">A sequence of values to calculate the average of.</
    param>
142    /// <param name="selector">A transform function to apply to each element.</
    param>
143    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
144    /// <returns>The average of the sequence of values.</returns>
145    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
146    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
147    public static double Average<TSource>(this IEnumerable<TSource> source,
    Func<TSource, double> selector);
148    /// <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>
149    /// <param name="source">A sequence of values to calculate the average of.</
    param>
150    /// <param name="selector">A transform function to apply to each element.</
    param>
151    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
152    /// <returns>The average of the sequence of values.</returns>
153    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
154    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
155    /// <exception cref="T:System.OverflowException">The sum of the elements in
    the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</
    exception>
156    public static double Average<TSource>(this IEnumerable<TSource> source,
    Func<TSource, int> selector);
157    /// <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>
158    /// <param name="source">A sequence of values to calculate the average of.</
    param>
159    /// <param name="selector">A transform function to apply to each element.</
    param>
160    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
161    /// <returns>The average of the sequence of values.</returns>
162    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
163    /// <exception cref="T:System.InvalidOperationException"><paramref

```



```
        name="source">source</paramref> contains no elements.</exception>
164    /// <exception cref="T:System.OverflowException">The sum of the elements in
    the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</
    exception>
165    public static double Average<TSource>(this IEnumerable<TSource> source,
    Func<TSource, long> selector);
166    /// <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>
167    /// <param name="source">A sequence of values to calculate the average of.</
    param>
168    /// <param name="selector">A transform function to apply to each element.</
    param>
169    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
170    /// <returns>The average of the sequence of values, or null if the source
    sequence is empty or contains only values that are null.</returns>
171    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
172    /// <exception cref="T:System.OverflowException">The sum of the elements in
    the sequence is larger than <see cref="F:System.Decimal.MaxValue"></
    see>.</exception>
173    public static decimal? Average<TSource>(this IEnumerable<TSource> source,
    Func<TSource, decimal?> selector);
174    /// <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>
175    /// <param name="source">A sequence of values to calculate the average of.</
    param>
176    /// <param name="selector">A transform function to apply to each element.</
    param>
177    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
178    /// <returns>The average of the sequence of values, or null if the source
    sequence is empty or contains only values that are null.</returns>
179    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
180    public static double? Average<TSource>(this IEnumerable<TSource> source,
    Func<TSource, double?> selector);
181    /// <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>
182    /// <param name="source">A sequence of values to calculate the average of.</
    param>
183    /// <param name="selector">A transform function to apply to each element.</
    param>
184    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
185    /// <returns>The average of the sequence of values, or null if the source
    sequence is empty or contains only values that are null.</returns>
```

```
186    /// <exception cref="T:System.ArgumentNullException"><paramref  
    name="source">source</paramref> or <paramref name="selector">selector</  
    paramref> is null.</exception> 7  
187    /// <exception cref="T:System.OverflowException">The sum of the elements in  
    the sequence is larger than <see cref="F:System.Int64.MaxValue"></see>.</  
    exception> 7  
188    public static double? Average<TSource>(this IEnumerable<TSource> source,  
    Func<TSource, int?> selector); 7  
189    /// <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> 7  
190    /// <param name="source">A sequence of values to calculate the average of.</  
    param> 7  
191    /// <param name="selector">A transform function to apply to each element.</  
    param> 7  
192    /// <typeparam name="TSource">The type of the elements of source.</  
    typeparam> 7  
193    /// <returns>The average of the sequence of values, or null if the source  
    sequence is empty or contains only values that are null.</returns> 7  
194    public static double? Average<TSource>(this IEnumerable<TSource> source,  
    Func<TSource, long?> selector); 7  
195    /// <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> 7  
196    /// <param name="source">A sequence of values to calculate the average of.</  
    param> 7  
197    /// <param name="selector">A transform function to apply to each element.</  
    param> 7  
198    /// <typeparam name="TSource">The type of the elements of source.</  
    typeparam> 7  
199    /// <returns>The average of the sequence of values, or null if the source  
    sequence is empty or contains only values that are null.</returns> 7  
200    /// <exception cref="T:System.ArgumentNullException"><paramref  
    name="source">source</paramref> or <paramref name="selector">selector</  
    paramref> is null.</exception> 7  
201    public static float? Average<TSource>(this IEnumerable<TSource> source,  
    Func<TSource, float?> selector); 7  
202    /// <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> 7  
203    /// <param name="source">A sequence of values to calculate the average of.</  
    param> 7  
204    /// <param name="selector">A transform function to apply to each element.</  
    param> 7  
205    /// <typeparam name="TSource">The type of the elements of source.</  
    typeparam> 7  
206    /// <returns>The average of the sequence of values.</returns> 7  
207    /// <exception cref="T:System.ArgumentNullException"><paramref  
    name="source">source</paramref> or <paramref name="selector">selector</  
    paramref> is null.</exception> 7  
208    /// <exception cref="T:System.InvalidOperationException"><paramref  
    name="source">source</paramref> contains no elements.</exception> 7
```



```

209     public static float Average<TSource>(this IEnumerable<TSource> source,
210         Func<TSource, float> selector);
211     /// <summary>Casts the elements of an <see
212         cref="T:System.Collections.IEnumerable"></see> to the specified type.</
213         summary>
214     /// <param name="source">The <see cref="T:System.Collections.IEnumerable"></
215         see> that contains the elements to be cast to type TResult.</param>
216     /// <typeparam name="TResult">The type to cast the elements of source to.</
217         typeparam>
218     /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
219         see> that contains each element of the source sequence cast to the
220         specified type.</returns>
221     /// <exception cref="T:System.ArgumentNullException"><paramref
222         name="source">source</paramref> is null.</exception>
223     /// <exception cref="T:System.InvalidCastException">An element in the
224         sequence cannot be cast to type <paramref name="TResult">TResult</
225         paramref>.</exception>
226     public static IEnumerable<TResult> Cast<TResult>(this IEnumerable source);
227     /// <summary>Concatenates two sequences.</summary>
228     /// <param name="first">The first sequence to concatenate.</param>
229     /// <param name="second">The sequence to concatenate to the first
230         sequence.</param>
231     /// <typeparam name="TSource">The type of the elements of the input
232         sequences.</typeparam>
233     /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
234         see> that contains the concatenated elements of the two input sequences.</
235         returns>
236     /// <exception cref="T:System.ArgumentNullException"><paramref
237         name="first">first</paramref> or <paramref name="second">second</paramref>
238         is null.</exception>
239     public static IEnumerable<TSource> Concat<TSource>(this IEnumerable<TSource>
240         first, IEnumerable<TSource> second);
241     /// <summary>Determines whether a sequence contains a specified element by
242         using the default equality comparer.</summary>
243     /// <param name="source">A sequence in which to locate a value.</param>
244     /// <param name="value">The value to locate in the sequence.</param>
245     /// <typeparam name="TSource">The type of the elements of source.</
246         typeparam>
247     /// <returns>true if the source sequence contains an element that has the
248         specified value; otherwise, false.</returns>
249     /// <exception cref="T:System.ArgumentNullException"><paramref
250         name="source">source</paramref> is null.</exception>
251     public static bool Contains<TSource>(this IEnumerable<TSource> source,
252         TSource value);
253     /// <summary>Determines whether a sequence contains a specified element by
254         using a specified <see
255         cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</summary>
256     /// <param name="source">A sequence in which to locate a value.</param>
257     /// <param name="value">The value to locate in the sequence.</param>
258     /// <param name="comparer">An equality comparer to compare values.</param>
259     /// <typeparam name="TSource">The type of the elements of source.</
260         typeparam>

```

```

236    /// <returns>true if the source sequence contains an element that has the
    specified value; otherwise, false.</returns>
237    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
238    public static bool Contains<TSource>(this IEnumerable<TSource> source,
    TSource value, IEqualityComparer<TSource> comparer);
239    /// <summary>Returns the number of elements in a sequence.</summary>
240    /// <param name="source">A sequence that contains elements to be counted.</
    param>
241    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
242    /// <returns>The number of elements in the input sequence.</returns>
243    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
244    /// <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>
245    public static int Count<TSource>(this IEnumerable<TSource> source);
246    /// <summary>Returns a number that represents how many elements in the
    specified sequence satisfy a condition.</summary>
247    /// <param name="source">A sequence that contains elements to be tested and
    counted.</param>
248    /// <param name="predicate">A function to test each element for a
    condition.</param>
249    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
250    /// <returns>A number that represents how many elements in the sequence
    satisfy the condition in the predicate function.</returns>
251    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
252    /// <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>
253    public static int Count<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
254    /// <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>
255    /// <param name="source">The sequence to return a default value for if it is
    empty.</param>
256    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
257    /// <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>
258    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
259    public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this
    IEnumerable<TSource> source);

```

```

260    /// <summary>Returns the elements of the specified sequence or the specified
        value in a singleton collection if the sequence is empty.</summary>
261    /// <param name="source">The sequence to return the specified value for if
        it is empty.</param>
262    /// <param name="defaultValue">The value to return if the sequence is
        empty.</param>
263    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
264    /// <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>
265    public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this
        IEnumerable<TSource> source, TSource defaultValue);
266    /// <summary>Returns distinct elements from a sequence by using the default
        equality comparer to compare values.</summary>
267    /// <param name="source">The sequence to remove duplicate elements from.</
        param>
268    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
269    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
        see> that contains distinct elements from the source sequence.</returns>
270    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
271    public static IEnumerable<TSource> Distinct<TSource>(this
        IEnumerable<TSource> source);
272    /// <summary>Returns distinct elements from a sequence by using a specified
        <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to
        compare values.</summary>
273    /// <param name="source">The sequence to remove duplicate elements from.</
        param>
274    /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
        values.</param>
275    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
276    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
        see> that contains distinct elements from the source sequence.</returns>
277    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
278    public static IEnumerable<TSource> Distinct<TSource>(this
        IEnumerable<TSource> source, IEqualityComparer<TSource> comparer);
279    /// <summary>Returns the element at a specified index in a sequence.</
        summary>
280    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an
        element from.</param>
281    /// <param name="index">The zero-based index of the element to retrieve.</
        param>
282    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
283    /// <returns>The element at the specified position in the source sequence.</

```

```

        returns>
284    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
285    /// <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>
286    public static TSource ElementAt<TSource>(this IEnumerable<TSource> source,
        int index);
287    /// <summary>Returns the element at a specified index in a sequence or a
        default value if the index is out of range.</summary>
288    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an
        element from.</param>
289    /// <param name="index">The zero-based index of the element to retrieve.</
        param>
290    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
291    /// <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>
292    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
293    public static TSource ElementAtOrDefault<TSource>(this IEnumerable<TSource>
        source, int index);
294    /// <summary>Returns an empty <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> that has the
        specified type argument.</summary>
295    /// <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>
296    /// <returns>An empty <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> whose type
        argument is <paramref name="TResult">TResult</paramref>.</returns>
297    public static IEnumerable<TResult> Empty<TResult>();
298    /// <summary>Produces the set difference of two sequences by using the
        default equality comparer to compare values.</summary>
299    /// <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>
300    /// <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>
301    /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
302    /// <returns>A sequence that contains the set difference of the elements of
        two sequences.</returns>
303    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="first">first</paramref> or <paramref name="second">second</paramref>
        is null.</exception>
304    public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource>

```

```

        first, IEnumerable<TSource> second);
305    /// <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>
306    /// <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>
307    /// <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>
308    /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
        values.</param>
309    /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
310    /// <returns>A sequence that contains the set difference of the elements of
        two sequences.</returns>
311    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="first">first</paramref> or <paramref name="second">second</paramref>
        is null.</exception>
312    public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource>
        first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
313    /// <summary>Returns the first element of a sequence.</summary>
314    /// <param name="source">The <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the
        first element of.</param>
315    /// <typeparam name="TSource">The type of the elements of source.</
       typeparam>
316    /// <returns>The first element in the specified sequence.</returns>
317    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
318    /// <exception cref="T:System.InvalidOperationException">The source sequence
        is empty.</exception>
319    public static TSource First<TSource>(this IEnumerable<TSource> source);
320    /// <summary>Returns the first element in a sequence that satisfies a
        specified condition.</summary>
321    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an
        element from.</param>
322    /// <param name="predicate">A function to test each element for a
        condition.</param>
323    /// <typeparam name="TSource">The type of the elements of source.</
       typeparam>
324    /// <returns>The first element in the sequence that passes the test in the
        specified predicate function.</returns>
325    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="predicate">predicate</
        paramref> is null.</exception>
326    /// <exception cref="T:System.InvalidOperationException">No element
        satisfies the condition in <paramref name="predicate">predicate</
        paramref>. -or- The source sequence is empty.</exception>

```

```

327 public static TSource First<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
328 /// <summary>Returns the first element of a sequence, or a default value if
    the sequence contains no elements.</summary>
329 /// <param name="source">The <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the
    first element of.</param>
330 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
331 /// <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>
332 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
333 public static TSource FirstOrDefault<TSource>(this IEnumerable<TSource>
    source);
334 /// <summary>Returns the first element of the sequence that satisfies a
    condition or a default value if no such element is found.</summary>
335 /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an
    element from.</param>
336 /// <param name="predicate">A function to test each element for a
    condition.</param>
337 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
338 /// <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>
339 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
340 public static TSource FirstOrDefault<TSource>(this IEnumerable<TSource>
    source, Func<TSource, bool> predicate);
341 /// <summary>Groups the elements of a sequence according to a specified key
    selector function.</summary>
342 /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
    group.</param>
343 /// <param name="keySelector">A function to extract the key for each
    element.</param>
344 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
345 /// <typeparam name="TKey">The type of the key returned by keySelector.</
   typeparam>
346 /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1"
    sourceendlinenumber="1">An <code>IEnumerable<TSource></code> in C# or <code>IEnumerable(Of TSource)</code> in Visual Basic where each <xref
    href="System.Linq.IGrouping`2"></xref> object contains a sequence of

```



```

    objects and a key.</p>
347    /// </returns>
348    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
349    public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>
    (this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
350    /// <summary>Groups the elements of a sequence according to a specified key
    selector function and compares the keys by using a specified comparer.</
    summary>
351    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
    group.</param>
352    /// <param name="keySelector">A function to extract the key for each
    element.</param>
353    /// <param name="comparer">An <see
    cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
    keys.</param>
354    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
355    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
356    /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1"
    sourceendlinenumber="1">An <code>IEnumerable<TSource></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>
357    /// </returns>
358    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
359    public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>
    (this IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    IEqualityComparer<TKey> comparer);
360    /// <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>
361    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
    group.</param>
362    /// <param name="keySelector">A function to extract the key for each
    element.</param>
363    /// <param name="elementSelector">A function to map each source element to
    an element in the <see cref="T:System.Linq.IGrouping`2"></see>.</param>
364    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
365    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
366    /// <typeparam name="TElement">The type of the elements in the <see
    cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
367    /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1"

```

```

sourceendlinenumber="1">An IEnumerable<TElement> in C# or IEnumerable(Of TElement) in Visual Basic where each object contains a collection of objects of type TElement and a key.</p>
368 /// </returns>
369 /// <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>
370 public static IEnumerable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey,
    TElement>(this IEnumerable<TSource> source, Func<TSource, TKey>
    keySelector, Func<TSource, TElement> elementSelector);
371 /// <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>
372 /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
    group.</param>
373 /// <param name="keySelector">A function to extract the key for each
    element.</param>
374 /// <param name="elementSelector">A function to map each source element to
    an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
375 /// <param name="comparer">An <see
    cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
    keys.</param>
376 /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
377 /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
378 /// <typeparam name="TElement">The type of the elements in the <see
    cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
379 /// <returns><p sourcefile="System.Linq.yml" sourcestartlinenumber="1"
    sourceendlinenumber="1">An IEnumerable<TElement> in C# or IEnumerable(Of TElement) in Visual Basic where each object contains a collection of objects of type TElement and a key.</p>
380 /// </returns>
381 /// <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>
382 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);
383 /// <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>

```

```

384    /// <param name="source">An <see
      cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
      group.</param>
385    /// <param name="keySelector">A function to extract the key for each
      element.</param>
386    /// <param name="resultSelector">A function to create a result value from
      each group.</param>
387    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
388    /// <typeparam name="TKey">The type of the key returned by keySelector.</
      typeparam>
389    /// <typeparam name="TResult">The type of the result value returned by
      resultSelector.</typeparam>
390    /// <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>
391    public static IEnumerable<TResult> GroupBy<TSource, TKey, TResult>(this
      IEnumerable<TSource> source, Func<TSource, TKey> keySelector, Func<TKey,
      IEnumerable<TSource>, TResult> resultSelector);
392    /// <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>
393    /// <param name="source">An <see
      cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
      group.</param>
394    /// <param name="keySelector">A function to extract the key for each
      element.</param>
395    /// <param name="resultSelector">A function to create a result value from
      each group.</param>
396    /// <param name="comparer">An <see
      cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
      keys with.</param>
397    /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
398    /// <typeparam name="TKey">The type of the key returned by keySelector.</
      typeparam>
399    /// <typeparam name="TResult">The type of the result value returned by
      resultSelector.</typeparam>
400    /// <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>
401    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);
402    /// <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>
403    /// <param name="source">An <see
      cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
      group.</param>

```

```

404    /// <param name="keySelector">A function to extract the key for each element.</param>
405    /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
406    /// <param name="resultSelector">A function to create a result value from each group.</param>
407    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
408    /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
409    /// <typeparam name="TElement">The type of the elements in each <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
410    /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
411    /// <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>
412    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);
413    /// <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>
414    /// <param name="source">An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to group.</param>
415    /// <param name="keySelector">A function to extract the key for each element.</param>
416    /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
417    /// <param name="resultSelector">A function to create a result value from each group.</param>
418    /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys with.</param>
419    /// <typeparam name="TSource">The type of the elements of source.</typeparam>
420    /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
421    /// <typeparam name="TElement">The type of the elements in each <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
422    /// <typeparam name="TResult">The type of the result value returned by resultSelector.</typeparam>
423    /// <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>
424    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);

```

```

425     /// <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>
426     /// <param name="outer">The first sequence to join.</param>
427     /// <param name="inner">The sequence to join to the first sequence.</param>
428     /// <param name="outerKeySelector">A function to extract the join key from
        each element of the first sequence.</param>
429     /// <param name="innerKeySelector">A function to extract the join key from
        each element of the second sequence.</param>
430     /// <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>
431     /// <typeparam name="TOuter">The type of the elements of the first
        sequence.</typeparam>
432     /// <typeparam name="TInner">The type of the elements of the second
        sequence.</typeparam>
433     /// <typeparam name="TKey">The type of the keys returned by the key selector
        functions.</typeparam>
434     /// <typeparam name="TResult">The type of the result elements.</typeparam>
435     /// <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>
436     /// <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>
437     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);
438     /// <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>
439     /// <param name="outer">The first sequence to join.</param>
440     /// <param name="inner">The sequence to join to the first sequence.</param>
441     /// <param name="outerKeySelector">A function to extract the join key from
        each element of the first sequence.</param>
442     /// <param name="innerKeySelector">A function to extract the join key from
        each element of the second sequence.</param>
443     /// <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>
444     /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to hash and
        compare keys.</param>
445     /// <typeparam name="TOuter">The type of the elements of the first
        sequence.</typeparam>
446     /// <typeparam name="TInner">The type of the elements of the second
        sequence.</typeparam>

```

```

447     /// <typeparam name="TKey">The type of the keys returned by the key selector ↗
        functions.</typeparam>
448     /// <typeparam name="TResult">The type of the result elements.</typeparam>
449     /// <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>
450     /// <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>
451     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);
452     /// <summary>Produces the set intersection of two sequences by using the ↗
        default equality comparer to compare values.</summary>
453     /// <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>
454     /// <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>
455     /// <typeparam name="TSource">The type of the elements of the input ↗
        sequences.</typeparam>
456     /// <returns>A sequence that contains the elements that form the set ↗
        intersection of two sequences.</returns>
457     /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="first">first</paramref> or <paramref name="second">second</paramref> ↗
        is null.</exception>
458     public static IEnumerable<TSource> Intersect<TSource>(this ↗
        IEnumerable<TSource> first, IEnumerable<TSource> second);
459     /// <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>
460     /// <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>
461     /// <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>
462     /// <param name="comparer">An <see ↗
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare ↗
        values.</param>
463     /// <typeparam name="TSource">The type of the elements of the input ↗
        sequences.</typeparam>
464     /// <returns>A sequence that contains the elements that form the set ↗
        intersection of two sequences.</returns>
465     /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="first">first</paramref> or <paramref name="second">second</paramref> ↗

```



```

        is null.</exception>
466     public static IEnumerable<TSource> Intersect<TSource>(this           ↗
        IEnumerable<TSource> first, IEnumerable<TSource> second,           ↗
        IEqualityComparer<TSource> comparer);
467     /// <summary>Correlates the elements of two sequences based on matching ↗
        keys. The default equality comparer is used to compare keys.</summary>
468     /// <param name="outer">The first sequence to join.</param>
469     /// <param name="inner">The sequence to join to the first sequence.</param>
470     /// <param name="outerKeySelector">A function to extract the join key from ↗
        each element of the first sequence.</param>
471     /// <param name="innerKeySelector">A function to extract the join key from ↗
        each element of the second sequence.</param>
472     /// <param name="resultSelector">A function to create a result element from ↗
        two matching elements.</param>
473     /// <typeparam name="TOuter">The type of the elements of the first ↗
        sequence.</typeparam>
474     /// <typeparam name="TInner">The type of the elements of the second ↗
        sequence.</typeparam>
475     /// <typeparam name="TKey">The type of the keys returned by the key selector ↗
        functions.</typeparam>
476     /// <typeparam name="TResult">The type of the result elements.</typeparam>
477     /// <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>
478     /// <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>
479     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);
480     /// <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>
481     /// <param name="outer">The first sequence to join.</param>
482     /// <param name="inner">The sequence to join to the first sequence.</param>
483     /// <param name="outerKeySelector">A function to extract the join key from ↗
        each element of the first sequence.</param>
484     /// <param name="innerKeySelector">A function to extract the join key from ↗
        each element of the second sequence.</param>
485     /// <param name="resultSelector">A function to create a result element from ↗
        two matching elements.</param>
486     /// <param name="comparer">An <see ↗
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to hash and ↗
        compare keys.</param>
487     /// <typeparam name="TOuter">The type of the elements of the first ↗
        sequence.</typeparam>
488     /// <typeparam name="TInner">The type of the elements of the second ↗
        sequence.</typeparam>

```

```

489    /// <typeparam name="TKey">The type of the keys returned by the key selector ↗
        functions.</typeparam>
490    /// <typeparam name="TResult">The type of the result elements.</typeparam>
491    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></see> ↗
        see> that has elements of type <paramref name="TResult">TResult</paramref> ↗
        that are obtained by performing an inner join on two sequences.</returns>
492    /// <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>
493    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);
494    /// <summary>Returns the last element of a sequence.</summary>
495    /// <param name="source">An <see ↗
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the ↗
        last element of.</param>
496    /// <typeparam name="TSource">The type of the elements of source.</ ↗
        typeparam>
497    /// <returns>The value at the last position in the source sequence.</ ↗
        returns>
498    /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="source">source</paramref> is null.</exception>
499    /// <exception cref="T:System.InvalidOperationException">The source sequence ↗
        is empty.</exception>
500    public static TSource Last<TSource>(this IEnumerable<TSource> source);
501    /// <summary>Returns the last element of a sequence that satisfies a ↗
        specified condition.</summary>
502    /// <param name="source">An <see ↗
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return an ↗
        element from.</param>
503    /// <param name="predicate">A function to test each element for a ↗
        condition.</param>
504    /// <typeparam name="TSource">The type of the elements of source.</ ↗
        typeparam>
505    /// <returns>The last element in the sequence that passes the test in the ↗
        specified predicate function.</returns>
506    /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="source">source</paramref> or <paramref name="predicate">predicate</ ↗
        paramref> is null.</exception>
507    /// <exception cref="T:System.InvalidOperationException">No element ↗
        satisfies the condition in <paramref name="predicate">predicate</ ↗
        paramref>. -or- The source sequence is empty.</exception>
508    public static TSource Last<TSource>(this IEnumerable<TSource> source, ↗
        Func<TSource, bool> predicate);
509    /// <summary>Returns the last element of a sequence, or a default value if ↗
        the sequence contains no elements.</summary>
510    /// <param name="source">An <see ↗
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the ↗
        last element of.</param>

```

```

511    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
512    /// <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>
513    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
514    public static

```

```

    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
535    /// <exception cref="T:System.OverflowException">The number of matching
    elements exceeds <see cref="F:System.Int64.MaxValue"></see>.</exception>
536    public static long LongCount<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
537    /// <summary>Returns the maximum value in a sequence of <see
    cref="T:System.Decimal"></see> values.</summary>
538    /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>
    values to determine the maximum value of.</param>
539    /// <returns>The maximum value in the sequence.</returns>
540    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
541    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
542    public static decimal Max(this IEnumerable<decimal> source);
543    /// <summary>Returns the maximum value in a sequence of <see
    cref="T:System.Double"></see> values.</summary>
544    /// <param name="source">A sequence of <see cref="T:System.Double"></see>
    values to determine the maximum value of.</param>
545    /// <returns>The maximum value in the sequence.</returns>
546    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
547    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
548    public static double Max(this IEnumerable<double> source);
549    /// <summary>Returns the maximum value in a sequence of <see
    cref="T:System.Int32"></see> values.</summary>
550    /// <param name="source">A sequence of <see cref="T:System.Int32"></see>
    values to determine the maximum value of.</param>
551    /// <returns>The maximum value in the sequence.</returns>
552    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
553    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
554    public static int Max(this IEnumerable<int> source);
555    /// <summary>Returns the maximum value in a sequence of <see
    cref="T:System.Int64"></see> values.</summary>
556    /// <param name="source">A sequence of <see cref="T:System.Int64"></see>
    values to determine the maximum value of.</param>
557    /// <returns>The maximum value in the sequence.</returns>
558    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
559    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
560    public static long Max(this IEnumerable<long> source);
561    /// <summary>Returns the maximum value in a sequence of nullable <see
    cref="T:System.Decimal"></see> values.</summary>
562    /// <param name="source">A sequence of nullable <see
    cref="T:System.Decimal"></see> values to determine the maximum value of.</
    param>
563    /// <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>
564    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
565    public static decimal? Max(this IEnumerable<decimal?> source);
566    /// <summary>Returns the maximum value in a sequence of nullable <see
    cref="T:System.Double"></see> values.</summary>
567    /// <param name="source">A sequence of nullable <see
    cref="T:System.Double"></see> values to determine the maximum value of.</
    param>
568    /// <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>
569    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
570    public static double? Max(this IEnumerable<double?> source);
571    /// <summary>Returns the maximum value in a sequence of nullable <see
    cref="T:System.Int32"></see> values.</summary>
572    /// <param name="source">A sequence of nullable <see
    cref="T:System.Int32"></see> values to determine the maximum value of.</
    param>
573    /// <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>
574    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
575    public static int? Max(this IEnumerable<int?> source);
576    /// <summary>Returns the maximum value in a sequence of nullable <see
    cref="T:System.Int64"></see> values.</summary>
577    /// <param name="source">A sequence of nullable <see
    cref="T:System.Int64"></see> values to determine the maximum value of.</
    param>
578    /// <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>
579    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
580    public static long? Max(this IEnumerable<long?> source);
581    /// <summary>Returns the maximum value in a sequence of nullable <see
    cref="T:System.Single"></see> values.</summary>
582    /// <param name="source">A sequence of nullable <see
    cref="T:System.Single"></see> values to determine the maximum value of.</
    param>
583    /// <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>
584    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
585    public static float? Max(this IEnumerable<float?> source);
586    /// <summary>Returns the maximum value in a sequence of <see
    cref="T:System.Single"></see> values.</summary>
587    /// <param name="source">A sequence of <see cref="T:System.Single"></see>
    values to determine the maximum value of.</param>
588    /// <returns>The maximum value in the sequence.</returns>
589    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>

```

```

590     /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
591     public static float Max(this IEnumerable<float> source);
592     /// <summary>Returns the maximum value in a generic sequence.</summary>
593     /// <param name="source">A sequence of values to determine the maximum value
    of.</param>
594     /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
595     /// <returns>The maximum value in the sequence.</returns>
596     /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
597     public static TSource Max<TSource>(this IEnumerable<TSource> source);
598     /// <summary>Invokes a transform function on each element of a sequence and
    returns the maximum <see cref="T:System.Decimal"></see> value.</summary>
599     /// <param name="source">A sequence of values to determine the maximum value
    of.</param>
600     /// <param name="selector">A transform function to apply to each element.</
    param>
601     /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
602     /// <returns>The maximum value in the sequence.</returns>
603     /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
604     /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
605     public static decimal Max<TSource>(this IEnumerable<TSource> source,
    Func<TSource, decimal> selector);
606     /// <summary>Invokes a transform function on each element of a sequence and
    returns the maximum <see cref="T:System.Double"></see> value.</summary>
607     /// <param name="source">A sequence of values to determine the maximum value
    of.</param>
608     /// <param name="selector">A transform function to apply to each element.</
    param>
609     /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
610     /// <returns>The maximum value in the sequence.</returns>
611     /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
612     /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
613     public static double Max<TSource>(this IEnumerable<TSource> source,
    Func<TSource, double> selector);
614     /// <summary>Invokes a transform function on each element of a sequence and
    returns the maximum <see cref="T:System.Int32"></see> value.</summary>
615     /// <param name="source">A sequence of values to determine the maximum value
    of.</param>
616     /// <param name="selector">A transform function to apply to each element.</
    param>
617     /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>

```



```

618     /// <returns>The maximum value in the sequence.</returns>
619     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
620     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
621     public static int Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, int> selector);
622     /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum <see cref="T:System.Int64"></see> value.</summary>
623     /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
624     /// <param name="selector">A transform function to apply to each element.</
        param>
625     /// <typeparam name="TSource">The type of the elements of source.</
       typeparam>
626     /// <returns>The maximum value in the sequence.</returns>
627     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
628     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
629     public static long Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, long> selector);
630     /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum nullable <see cref="T:System.Decimal"></see> value.</
        summary>
631     /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
632     /// <param name="selector">A transform function to apply to each element.</
        param>
633     /// <typeparam name="TSource">The type of the elements of source.</
       typeparam>
634     /// <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>
635     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
636     public static decimal? Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, decimal?> selector);
637     /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum nullable <see cref="T:System.Double"></see> value.</
        summary>
638     /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
639     /// <param name="selector">A transform function to apply to each element.</
        param>
640     /// <typeparam name="TSource">The type of the elements of source.</
       typeparam>
641     /// <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>
642    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
643    public static double? Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, double?> selector);
644    /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum nullable <see cref="T:System.Int32"></see> value.</
        summary>
645    /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
646    /// <param name="selector">A transform function to apply to each element.</
        param>
647    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
648    /// <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>
649    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
650    public static int? Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, int?> selector);
651    /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum nullable <see cref="T:System.Int64"></see> value.</
        summary>
652    /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
653    /// <param name="selector">A transform function to apply to each element.</
        param>
654    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
655    /// <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>
656    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
657    public static long? Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, long?> selector);
658    /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum nullable <see cref="T:System.Single"></see> value.</
        summary>
659    /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
660    /// <param name="selector">A transform function to apply to each element.</
        param>
661    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
662    /// <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>

```

```

663     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
664     public static float? Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, float?> selector);
665     /// <summary>Invokes a transform function on each element of a sequence and
        returns the maximum <see cref="T:System.Single"></see> value.</summary>
666     /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
667     /// <param name="selector">A transform function to apply to each element.</
        param>
668     /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
669     /// <returns>The maximum value in the sequence.</returns>
670     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
671     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
672     public static float Max<TSource>(this IEnumerable<TSource> source,
        Func<TSource, float> selector);
673     /// <summary>Invokes a transform function on each element of a generic
        sequence and returns the maximum resulting value.</summary>
674     /// <param name="source">A sequence of values to determine the maximum value
        of.</param>
675     /// <param name="selector">A transform function to apply to each element.</
        param>
676     /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
677     /// <typeparam name="TResult">The type of the value returned by selector.</
        typeparam>
678     /// <returns>The maximum value in the sequence.</returns>
679     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
680     public static TResult Max<TSource, TResult>(this IEnumerable<TSource>
        source, Func<TSource, TResult> selector);
681     /// <summary>Returns the minimum value in a sequence of <see
        cref="T:System.Decimal"></see> values.</summary>
682     /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>
        values to determine the minimum value of.</param>
683     /// <returns>The minimum value in the sequence.</returns>
684     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
685     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
686     public static decimal Min(this IEnumerable<decimal> source);
687     /// <summary>Returns the minimum value in a sequence of <see
        cref="T:System.Double"></see> values.</summary>
688     /// <param name="source">A sequence of <see cref="T:System.Double"></see>
        values to determine the minimum value of.</param>
689     /// <returns>The minimum value in the sequence.</returns>

```

```

690    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
691    /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
692    public static double Min(this IEnumerable<double> source);
693    /// <summary>Returns the minimum value in a sequence of <see
        cref="T:System.Int32"></see> values.</summary>
694    /// <param name="source">A sequence of <see cref="T:System.Int32"></see>
        values to determine the minimum value of.</param>
695    /// <returns>The minimum value in the sequence.</returns>
696    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
697    /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
698    public static int Min(this IEnumerable<int> source);
699    /// <summary>Returns the minimum value in a sequence of <see
        cref="T:System.Int64"></see> values.</summary>
700    /// <param name="source">A sequence of <see cref="T:System.Int64"></see>
        values to determine the minimum value of.</param>
701    /// <returns>The minimum value in the sequence.</returns>
702    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
703    /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
704    public static long Min(this IEnumerable<long> source);
705    /// <summary>Returns the minimum value in a sequence of nullable <see
        cref="T:System.Decimal"></see> values.</summary>
706    /// <param name="source">A sequence of nullable <see
        cref="T:System.Decimal"></see> values to determine the minimum value of.</
        param>
707    /// <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>
708    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
709    public static decimal? Min(this IEnumerable<decimal?> source);
710    /// <summary>Returns the minimum value in a sequence of nullable <see
        cref="T:System.Double"></see> values.</summary>
711    /// <param name="source">A sequence of nullable <see
        cref="T:System.Double"></see> values to determine the minimum value of.</
        param>
712    /// <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>
713    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
714    public static double? Min(this IEnumerable<double?> source);
715    /// <summary>Returns the minimum value in a sequence of nullable <see
        cref="T:System.Int32"></see> values.</summary>
716    /// <param name="source">A sequence of nullable <see
        cref="T:System.Int32"></see> values to determine the minimum value of.</
        param>
717    /// <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>
718    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
719    public static int? Min(this IEnumerable<int?> source);
720    /// <summary>Returns the minimum value in a sequence of nullable <see
    cref="T:System.Int64"></see> values.</summary>
721    /// <param name="source">A sequence of nullable <see
    cref="T:System.Int64"></see> values to determine the minimum value of.</
    param>
722    /// <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>
723    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
724    public static long? Min(this IEnumerable<long?> source);
725    /// <summary>Returns the minimum value in a sequence of nullable <see
    cref="T:System.Single"></see> values.</summary>
726    /// <param name="source">A sequence of nullable <see
    cref="T:System.Single"></see> values to determine the minimum value of.</
    param>
727    /// <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>
728    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
729    public static float? Min(this IEnumerable<float?> source);
730    /// <summary>Returns the minimum value in a sequence of <see
    cref="T:System.Single"></see> values.</summary>
731    /// <param name="source">A sequence of <see cref="T:System.Single"></see>
    values to determine the minimum value of.</param>
732    /// <returns>The minimum value in the sequence.</returns>
733    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
734    /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
735    public static float Min(this IEnumerable<float> source);
736    /// <summary>Returns the minimum value in a generic sequence.</summary>
737    /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
738    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
739    /// <returns>The minimum value in the sequence.</returns>
740    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
741    public static TSource Min<TSource>(this IEnumerable<TSource> source);
742    /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum <see cref="T:System.Decimal"></see> value.</summary>
743    /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
744    /// <param name="selector">A transform function to apply to each element.</
    param>
745    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
746    /// <returns>The minimum value in the sequence.</returns>

```

```

747  /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
748  /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
749  public static decimal Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, decimal> selector);
750  /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum <see cref="T:System.Double"></see> value.</summary>
751  /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
752  /// <param name="selector">A transform function to apply to each element.</
    param>
753  /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
754  /// <returns>The minimum value in the sequence.</returns>
755  /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
756  /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
757  public static double Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, double> selector);
758  /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum <see cref="T:System.Int32"></see> value.</summary>
759  /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
760  /// <param name="selector">A transform function to apply to each element.</
    param>
761  /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
762  /// <returns>The minimum value in the sequence.</returns>
763  /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
764  /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
765  public static int Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, int> selector);
766  /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum <see cref="T:System.Int64"></see> value.</summary>
767  /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
768  /// <param name="selector">A transform function to apply to each element.</
    param>
769  /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
770  /// <returns>The minimum value in the sequence.</returns>
771  /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
772  /// <exception cref="T:System.InvalidOperationException"><paramref

```



```

    name="source">source</paramref> contains no elements.</exception>
773     public static long Min<TSource>(this IEnumerable<TSource> source,           ↗
        Func<TSource, long> selector);
774     /// <summary>Invokes a transform function on each element of a sequence and ↗
        returns the minimum nullable <see cref="T:System.Decimal"></see> value.</ ↗
        summary>
775     /// <param name="source">A sequence of values to determine the minimum value ↗
        of.</param>
776     /// <param name="selector">A transform function to apply to each element.</ ↗
        param>
777     /// <typeparam name="TSource">The type of the elements of source.</ ↗
        typeparam>
778     /// <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>
779     /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="source">source</paramref> or <paramref name="selector">selector</ ↗
        paramref> is null.</exception>
780     public static decimal? Min<TSource>(this IEnumerable<TSource> source,       ↗
        Func<TSource, decimal?> selector);
781     /// <summary>Invokes a transform function on each element of a sequence and ↗
        returns the minimum nullable <see cref="T:System.Double"></see> value.</ ↗
        summary>
782     /// <param name="source">A sequence of values to determine the minimum value ↗
        of.</param>
783     /// <param name="selector">A transform function to apply to each element.</ ↗
        param>
784     /// <typeparam name="TSource">The type of the elements of source.</ ↗
        typeparam>
785     /// <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>
786     /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="source">source</paramref> or <paramref name="selector">selector</ ↗
        paramref> is null.</exception>
787     public static double? Min<TSource>(this IEnumerable<TSource> source,       ↗
        Func<TSource, double?> selector);
788     /// <summary>Invokes a transform function on each element of a sequence and ↗
        returns the minimum nullable <see cref="T:System.Int32"></see> value.</ ↗
        summary>
789     /// <param name="source">A sequence of values to determine the minimum value ↗
        of.</param>
790     /// <param name="selector">A transform function to apply to each element.</ ↗
        param>
791     /// <typeparam name="TSource">The type of the elements of source.</ ↗
        typeparam>
792     /// <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>
793     /// <exception cref="T:System.ArgumentNullException"><paramref ↗
        name="source">source</paramref> or <paramref name="selector">selector</ ↗
        paramref> is null.</exception>

```

```
794 public static int? Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, int?> selector);
795 /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum nullable <see cref="T:System.Int64"></see> value.</
    summary>
796 /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
797 /// <param name="selector">A transform function to apply to each element.</
    param>
798 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
799 /// <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>
800 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
801 public static long? Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, long?> selector);
802 /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum nullable <see cref="T:System.Single"></see> value.</
    summary>
803 /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
804 /// <param name="selector">A transform function to apply to each element.</
    param>
805 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
806 /// <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>
807 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
808 public static float? Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, float?> selector);
809 /// <summary>Invokes a transform function on each element of a sequence and
    returns the minimum <see cref="T:System.Single"></see> value.</summary>
810 /// <param name="source">A sequence of values to determine the minimum value
    of.</param>
811 /// <param name="selector">A transform function to apply to each element.</
    param>
812 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
813 /// <returns>The minimum value in the sequence.</returns>
814 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
815 /// <exception cref="T:System.InvalidOperationException"><paramref
    name="source">source</paramref> contains no elements.</exception>
816 public static float Min<TSource>(this IEnumerable<TSource> source,
    Func<TSource, float> selector);
```

```

817     /// <summary>Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.</summary>
818     /// <param name="source">A sequence of values to determine the minimum value of.</param>
819     /// <param name="selector">A transform function to apply to each element.</param>
820     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
821     /// <typeparam name="TResult">The type of the value returned by selector.</typeparam>
822     /// <returns>The minimum value in the sequence.</returns>
823     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
824     public static TResult Min<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, TResult> selector);
825     /// <summary>Filters the elements of an <see cref="T:System.Collections.IEnumerable"></see> based on a specified type.</summary>
826     /// <param name="source">The <see cref="T:System.Collections.IEnumerable"></see> whose elements to filter.</param>
827     /// <typeparam name="TResult">The type to filter the elements of the sequence on.</typeparam>
828     /// <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>
829     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
830     public static IEnumerable<TResult> OfType<TResult>(this IEnumerable source);
831     /// <summary>Sorts the elements of a sequence in ascending order according to a key.</summary>
832     /// <param name="source">A sequence of values to order.</param>
833     /// <param name="keySelector">A function to extract a key from an element.</param>
834     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
835     /// <typeparam name="TKey">The type of the key returned by keySelector.</typeparam>
836     /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose elements are sorted according to a key.</returns>
837     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
838     public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
839     /// <summary>Sorts the elements of a sequence in ascending order by using a specified comparer.</summary>
840     /// <param name="source">A sequence of values to order.</param>
841     /// <param name="keySelector">A function to extract a key from an element.</param>
842     /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</param>

```

```

    param>
843    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
844    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
845    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
    elements are sorted according to a key.</returns>
846    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
847    public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this
    IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer);
848    /// <summary>Sorts the elements of a sequence in descending order according
    to a key.</summary>
849    /// <param name="source">A sequence of values to order.</param>
850    /// <param name="keySelector">A function to extract a key from an element.</
    param>
851    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
852    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
853    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
    elements are sorted in descending order according to a key.</returns>
854    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
855    public static IOrderedEnumerable<TSource> OrderByDescending<TSource, TKey>
    (this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
856    /// <summary>Sorts the elements of a sequence in descending order by using a
    specified comparer.</summary>
857    /// <param name="source">A sequence of values to order.</param>
858    /// <param name="keySelector">A function to extract a key from an element.</
    param>
859    /// <param name="comparer">An <see
    cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
    param>
860    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
861    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
862    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
    elements are sorted in descending order according to a key.</returns>
863    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
864    public static IOrderedEnumerable<TSource> OrderByDescending<TSource, TKey>
    (this IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    IComparer<TKey> comparer);
865    /// <param name="source"></param>
866    /// <param name="element"></param>
867    /// <typeparam name="TSource"></typeparam>

```

```

868     /// <returns></returns>
869     public static IEnumerable<TSource> Prepend<TSource>(this
      IEnumerable<TSource> source, TSource element);
870     /// <summary>Generates a sequence of integral numbers within a specified
      range.</summary>
871     /// <param name="start">The value of the first integer in the sequence.</
      param>
872     /// <param name="count">The number of sequential integers to generate.</
      param>
873     /// <returns>An IEnumerable in C# or IEnumerable(Of Int32) in Visual Basic
      that contains a range of sequential integral numbers.</returns>
874     /// <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>
875     public static IEnumerable<int> Range(int start, int count);
876     /// <summary>Generates a sequence that contains one repeated value.</
      summary>
877     /// <param name="element">The value to be repeated.</param>
878     /// <param name="count">The number of times to repeat the value in the
      generated sequence.</param>
879     /// <typeparam name="TResult">The type of the value to be repeated in the
      result sequence.</typeparam>
880     /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
      see> that contains a repeated value.</returns>
881     /// <exception cref="T:System.ArgumentOutOfRangeException"><paramref
      name="count">count</paramref> is less than 0.</exception>
882     public static IEnumerable<TResult> Repeat<TResult>(TResult element, int
      count);
883     /// <summary>Inverts the order of the elements in a sequence.</summary>
884     /// <param name="source">A sequence of values to reverse.</param>
885     /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
886     /// <returns>A sequence whose elements correspond to those of the input
      sequence in reverse order.</returns>
887     /// <exception cref="T:System.ArgumentNullException"><paramref
      name="source">source</paramref> is null.</exception>
888     public static IEnumerable<TSource> Reverse<TSource>(this
      IEnumerable<TSource> source);
889     /// <summary>Projects each element of a sequence into a new form.</summary>
890     /// <param name="source">A sequence of values to invoke a transform function
      on.</param>
891     /// <param name="selector">A transform function to apply to each element.</
      param>
892     /// <typeparam name="TSource">The type of the elements of source.</
      typeparam>
893     /// <typeparam name="TResult">The type of the value returned by selector.</
      typeparam>
894     /// <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>
895     /// <exception cref="T:System.ArgumentNullException"><paramref

```

```

    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
896     public static IEnumerable<TResult> Select<TSource, TResult>(this           ↗
        IEnumerable<TSource> source, Func<TSource, TResult> selector);
897     /// <summary>Projects each element of a sequence into a new form by           ↗
        incorporating the element's index.</summary>
898     /// <param name="source">A sequence of values to invoke a transform function ↗
        on.</param>
899     /// <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>
900     /// <typeparam name="TSource">The type of the elements of source.</           ↗
        typeparam>
901     /// <typeparam name="TResult">The type of the value returned by selector.</   ↗
        typeparam>
902     /// <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>
903     /// <exception cref="T:System.ArgumentNullException"><paramref             ↗
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
904     public static IEnumerable<TResult> Select<TSource, TResult>(this           ↗
        IEnumerable<TSource> source, Func<TSource, int, TResult> selector);
905     /// <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>
906     /// <param name="source">A sequence of values to project.</param>
907     /// <param name="selector">A transform function to apply to each element.</   ↗
        param>
908     /// <typeparam name="TSource">The type of the elements of source.</           ↗
        typeparam>
909     /// <typeparam name="TResult">The type of the elements of the sequence         ↗
        returned by selector.</typeparam>
910     /// <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>
911     /// <exception cref="T:System.ArgumentNullException"><paramref             ↗
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
912     public static IEnumerable<TResult> SelectMany<TSource, TResult>(this       ↗
        IEnumerable<TSource> source, Func<TSource, IEnumerable<TResult>>
        selector);
913     /// <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>
914     /// <param name="source">A sequence of values to project.</param>
915     /// <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>
916     /// <typeparam name="TSource">The type of the elements of source.</           ↗
        typeparam>

```



```

917     /// <typeparam name="TResult">The type of the elements of the sequence returned by selector.</typeparam>
918     /// <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>
919     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
920     public static IEnumerable<TResult> SelectMany<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, int, IEnumerable<TResult>> selector);
921     /// <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>
922     /// <param name="source">A sequence of values to project.</param>
923     /// <param name="collectionSelector">A transform function to apply to each element of the input sequence.</param>
924     /// <param name="resultSelector">A transform function to apply to each element of the intermediate sequence.</param>
925     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
926     /// <typeparam name="TCollection">The type of the intermediate elements collected by collectionSelector.</typeparam>
927     /// <typeparam name="TResult">The type of the elements of the resulting sequence.</typeparam>
928     /// <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>
929     /// <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>
930     public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(this IEnumerable<TSource> source, Func<TSource, IEnumerable<TCollection>> collectionSelector, Func<TSource, TCollection, TResult> resultSelector);
931     /// <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>
932     /// <param name="source">A sequence of values to project.</param>
933     /// <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>
934     /// <param name="resultSelector">A transform function to apply to each element of the intermediate sequence.</param>
935     /// <typeparam name="TSource">The type of the elements of source.</typeparam>

```

```

936    /// <typeparam name="TCollection">The type of the intermediate elements
        collected by collectionSelector.</typeparam>
937    /// <typeparam name="TResult">The type of the elements of the resulting
        sequence.</typeparam>
938    /// <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>
939    /// <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>
940    public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>
        (this IEnumerable<TSource> source, Func<TSource, int,
        IEnumerable<TCollection>> collectionSelector, Func<TSource, TCollection,
        TResult> resultSelector);
941    /// <summary>Determines whether two sequences are equal by comparing the
        elements by using the default equality comparer for their type.</summary>
942    /// <param name="first">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to
        second.</param>
943    /// <param name="second">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to the
        first sequence.</param>
944    /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
945    /// <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>
946    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="first">first</paramref> or <paramref name="second">second</paramref>
        is null.</exception>
947    public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first,
        IEnumerable<TSource> second);
948    /// <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>
949    /// <param name="first">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to
        second.</param>
950    /// <param name="second">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to compare to the
        first sequence.</param>
951    /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to use to
        compare elements.</param>
952    /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
953    /// <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>
954    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="first">first</paramref> or <paramref name="second">second</paramref>
    is null.</exception>
955    public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first,
    IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
956    /// <summary>Returns the only element of a sequence, and throws an exception
    if there is not exactly one element in the sequence.</summary>
957    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the
    single element of.</param>
958    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
959    /// <returns>The single element of the input sequence.</returns>
960    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
961    /// <exception cref="T:System.InvalidOperationException">The input sequence
    contains more than one element. -or- The input sequence is empty.</
    exception>
962    public static TSource Single<TSource>(this IEnumerable<TSource> source);
963    /// <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>
964    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to return a
    single element from.</param>
965    /// <param name="predicate">A function to test an element for a condition.</
    param>
966    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
967    /// <returns>The single element of the input sequence that satisfies a
    condition.</returns>
968    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
969    /// <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>
970    public static TSource Single<TSource>(this IEnumerable<TSource> source,
    Func<TSource, bool> predicate);
971    /// <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>
972    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to return the
    single element of.</param>
973    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
974    /// <returns>The single element of the input sequence, or default(<paramref
    name="TSource">TSource</paramref>) if the sequence contains no elements.</

```

```

        returns>
975    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
976    /// <exception cref="T:System.InvalidOperationException">The input sequence
        contains more than one element.</exception>
977    public static TSource SingleOrDefault<TSource>(this IEnumerable<TSource>
        source);
978    /// <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>
979    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return a
        single element from.</param>
980    /// <param name="predicate">A function to test an element for a condition.</
        param>
981    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
982    /// <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>
983    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="predicate">predicate</
        paramref> is null.</exception>
984    public static TSource SingleOrDefault<TSource>(this IEnumerable<TSource>
        source, Func<TSource, bool> predicate);
985    /// <summary>Bypasses a specified number of elements in a sequence and then
        returns the remaining elements.</summary>
986    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return
        elements from.</param>
987    /// <param name="count">The number of elements to skip before returning the
        remaining elements.</param>
988    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
989    /// <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>
990    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
991    public static IEnumerable<TSource> Skip<TSource>(this IEnumerable<TSource>
        source, int count);
992    /// <param name="source"></param>
993    /// <param name="count"></param>
994    /// <typeparam name="TSource"></typeparam>
995    /// <returns></returns>
996    public static IEnumerable<TSource> SkipLast<TSource>(this
        IEnumerable<TSource> source, int count);
997    /// <summary>Bypasses elements in a sequence as long as a specified
        condition is true and then returns the remaining elements.</summary>
998    /// <param name="source">An <see
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return

```

```

        elements from.</param>
999    /// <param name="predicate">A function to test each element for a    7
        condition.</param>
1000   /// <typeparam name="TSource">The type of the elements of source.</    7
        typeparam>
1001   /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></    7
        see> that contains the elements from the input sequence starting at the    7
        first element in the linear series that does not pass the test specified    7
        by <paramref name="predicate">predicate</paramref>.</returns>
1002   /// <exception cref="T:System.ArgumentNullException"><paramref    7
        name="source">source</paramref> or <paramref name="predicate">predicate</    7
        paramref> is null.</exception>
1003   public static IEnumerable<TSource> SkipWhile<TSource>(this    7
        IEnumerable<TSource> source, Func<TSource, bool> predicate);
1004   /// <summary>Bypasses elements in a sequence as long as a specified    7
        condition is true and then returns the remaining elements. The element's    7
        index is used in the logic of the predicate function.</summary>
1005   /// <param name="source">An <see    7
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to return    7
        elements from.</param>
1006   /// <param name="predicate">A function to test each source element for a    7
        condition; the second parameter of the function represents the index of    7
        the source element.</param>
1007   /// <typeparam name="TSource">The type of the elements of source.</    7
        typeparam>
1008   /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></    7
        see> that contains the elements from the input sequence starting at the    7
        first element in the linear series that does not pass the test specified    7
        by <paramref name="predicate">predicate</paramref>.</returns>
1009   /// <exception cref="T:System.ArgumentNullException"><paramref    7
        name="source">source</paramref> or <paramref name="predicate">predicate</    7
        paramref> is null.</exception>
1010   public static IEnumerable<TSource> SkipWhile<TSource>(this    7
        IEnumerable<TSource> source, Func<TSource, int, bool> predicate);
1011   /// <summary>Computes the sum of a sequence of <see    7
        cref="T:System.Decimal"></see> values.</summary>
1012   /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>    7
        values to calculate the sum of.</param>
1013   /// <returns>The sum of the values in the sequence.</returns>
1014   /// <exception cref="T:System.ArgumentNullException"><paramref    7
        name="source">source</paramref> is null.</exception>
1015   /// <exception cref="T:System.OverflowException">The sum is larger than <see    7
        cref="F:System.Decimal.MaxValue"></see>.</exception>
1016   public static decimal Sum(this IEnumerable<decimal> source);
1017   /// <summary>Computes the sum of a sequence of <see    7
        cref="T:System.Double"></see> values.</summary>
1018   /// <param name="source">A sequence of <see cref="T:System.Double"></see>    7
        values to calculate the sum of.</param>
1019   /// <returns>The sum of the values in the sequence.</returns>
1020   /// <exception cref="T:System.ArgumentNullException"><paramref    7
        name="source">source</paramref> is null.</exception>
1021   public static double Sum(this IEnumerable<double> source);

```

```
1022    /// <summary>Computes the sum of a sequence of <see cref="T:System.Int32"></see> values.</summary>
1023    /// <param name="source">A sequence of <see cref="T:System.Int32"></see> values to calculate the sum of.</param>
1024    /// <returns>The sum of the values in the sequence.</returns>
1025    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
1026    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
1027    public static int Sum(this IEnumerable<int> source);
1028    /// <summary>Computes the sum of a sequence of <see cref="T:System.Int64"></see> values.</summary>
1029    /// <param name="source">A sequence of <see cref="T:System.Int64"></see> values to calculate the sum of.</param>
1030    /// <returns>The sum of the values in the sequence.</returns>
1031    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
1032    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int64.MaxValue"></see>.</exception>
1033    public static long Sum(this IEnumerable<long> source);
1034    /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Decimal"></see> values.</summary>
1035    /// <param name="source">A sequence of nullable <see cref="T:System.Decimal"></see> values to calculate the sum of.</param>
1036    /// <returns>The sum of the values in the sequence.</returns>
1037    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
1038    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
1039    public static decimal? Sum(this IEnumerable<decimal?> source);
1040    /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Double"></see> values.</summary>
1041    /// <param name="source">A sequence of nullable <see cref="T:System.Double"></see> values to calculate the sum of.</param>
1042    /// <returns>The sum of the values in the sequence.</returns>
1043    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
1044    public static double? Sum(this IEnumerable<double?> source);
1045    /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Int32"></see> values.</summary>
1046    /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></see> values to calculate the sum of.</param>
1047    /// <returns>The sum of the values in the sequence.</returns>
1048    /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
1049    /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
1050    public static int? Sum(this IEnumerable<int?> source);
1051    /// <summary>Computes the sum of a sequence of nullable <see cref="T:System.Int64"></see> values.</summary>
1052    /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></see> values to calculate the sum of.</param>
```



```

1053     /// <returns>The sum of the values in the sequence.</returns>
1054     /// <exception cref="T:System.ArgumentNullException"><paramref
1055         name="source">source</paramref> is null.</exception>
1056     /// <exception cref="T:System.OverflowException">The sum is larger than <see
1057         cref="F:System.Int64.MaxValue"></see>.</exception>
1058     public static long? Sum(this IEnumerable<long?> source);
1059     /// <summary>Computes the sum of a sequence of nullable <see
1060         cref="T:System.Single"></see> values.</summary>
1061     /// <param name="source">A sequence of nullable <see
1062         cref="T:System.Single"></see> values to calculate the sum of.</param>
1063     /// <returns>The sum of the values in the sequence.</returns>
1064     /// <exception cref="T:System.ArgumentNullException"><paramref
1065         name="source">source</paramref> is null.</exception>
1066     public static float? Sum(this IEnumerable<float?> source);
1067     /// <summary>Computes the sum of a sequence of <see
1068         cref="T:System.Single"></see> values.</summary>
1069     /// <param name="source">A sequence of <see cref="T:System.Single"></see>
1070         values to calculate the sum of.</param>
1071     /// <returns>The sum of the values in the sequence.</returns>
1072     /// <exception cref="T:System.ArgumentNullException"><paramref
1073         name="source">source</paramref> is null.</exception>
1074     public static float Sum(this IEnumerable<float> source);
1075     /// <summary>Computes the sum of the sequence of <see
1076         cref="T:System.Decimal"></see> values that are obtained by invoking a
1077         transform function on each element of the input sequence.</summary>
1078     /// <param name="source">A sequence of values that are used to calculate a
1079         sum.</param>
1080     /// <param name="selector">A transform function to apply to each element.</
1081         param>
1082     /// <typeparam name="TSource">The type of the elements of source.</
1083         typeparam>
1084     /// <returns>The sum of the projected values.</returns>
1085     /// <exception cref="T:System.ArgumentNullException"><paramref
1086         name="source">source</paramref> or <paramref name="selector">selector</
1087         paramref> is null.</exception>
1088     /// <exception cref="T:System.OverflowException">The sum is larger than <see
1089         cref="F:System.Decimal.MaxValue"></see>.</exception>
1090     public static decimal Sum<TSource>(this IEnumerable<TSource> source,
1091         Func<TSource, decimal> selector);
1092     /// <summary>Computes the sum of the sequence of <see
1093         cref="T:System.Double"></see> values that are obtained by invoking a
1094         transform function on each element of the input sequence.</summary>
1095     /// <param name="source">A sequence of values that are used to calculate a
1096         sum.</param>
1097     /// <param name="selector">A transform function to apply to each element.</
1098         param>
1099     /// <typeparam name="TSource">The type of the elements of source.</
1100         typeparam>
1101     /// <returns>The sum of the projected values.</returns>
1102     /// <exception cref="T:System.ArgumentNullException"><paramref
1103         name="source">source</paramref> or <paramref name="selector">selector</
1104         paramref> is null.</exception>

```

```

1081 public static double Sum<TSource>(this IEnumerable<TSource> source,
    Func<TSource, double> selector);
1082 /// <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>
1083 /// <param name="source">A sequence of values that are used to calculate a
    sum.</param>
1084 /// <param name="selector">A transform function to apply to each element.</
    param>
1085 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
1086 /// <returns>The sum of the projected values.</returns>
1087 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
1088 /// <exception cref="T:System.OverflowException">The sum is larger than <see
    cref="F:System.Int32.MaxValue"></see>.</exception>
1089 public static int Sum<TSource>(this IEnumerable<TSource> source,
    Func<TSource, int> selector);
1090 /// <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>
1091 /// <param name="source">A sequence of values that are used to calculate a
    sum.</param>
1092 /// <param name="selector">A transform function to apply to each element.</
    param>
1093 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
1094 /// <returns>The sum of the projected values.</returns>
1095 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
1096 /// <exception cref="T:System.OverflowException">The sum is larger than <see
    cref="F:System.Int64.MaxValue"></see>.</exception>
1097 public static long Sum<TSource>(this IEnumerable<TSource> source,
    Func<TSource, long> selector);
1098 /// <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>
1099 /// <param name="source">A sequence of values that are used to calculate a
    sum.</param>
1100 /// <param name="selector">A transform function to apply to each element.</
    param>
1101 /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
1102 /// <returns>The sum of the projected values.</returns>
1103 /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="selector">selector</
    paramref> is null.</exception>
1104 /// <exception cref="T:System.OverflowException">The sum is larger than <see
    cref="F:System.Decimal.MaxValue"></see>.</exception>
1105 public static decimal? Sum<TSource>(this IEnumerable<TSource> source,

```

```

    Func<TSource, decimal?> selector);
1106  /// <summary>Computes the sum of the sequence of nullable <see           7
    cref="T:System.Double"></see> values that are obtained by invoking a           7
    transform function on each element of the input sequence.</summary>
1107  /// <param name="source">A sequence of values that are used to calculate a           7
    sum.</param>
1108  /// <param name="selector">A transform function to apply to each element.</           7
    param>
1109  /// <typeparam name="TSource">The type of the elements of source.</           7
    typeparam>
1110  /// <returns>The sum of the projected values.</returns>
1111  /// <exception cref="T:System.ArgumentNullException"><paramref           7
    name="source">source</paramref> or <paramref name="selector">selector</           7
    paramref> is null.</exception>
1112  public static double? Sum<TSource>(this IEnumerable<TSource> source,           7
    Func<TSource, double?> selector);
1113  /// <summary>Computes the sum of the sequence of nullable <see           7
    cref="T:System.Int32"></see> values that are obtained by invoking a           7
    transform function on each element of the input sequence.</summary>
1114  /// <param name="source">A sequence of values that are used to calculate a           7
    sum.</param>
1115  /// <param name="selector">A transform function to apply to each element.</           7
    param>
1116  /// <typeparam name="TSource">The type of the elements of source.</           7
    typeparam>
1117  /// <returns>The sum of the projected values.</returns>
1118  /// <exception cref="T:System.ArgumentNullException"><paramref           7
    name="source">source</paramref> or <paramref name="selector">selector</           7
    paramref> is null.</exception>
1119  /// <exception cref="T:System.OverflowException">The sum is larger than <see           7
    cref="F:System.Int32.MaxValue"></see>.</exception>
1120  public static int? Sum<TSource>(this IEnumerable<TSource> source,           7
    Func<TSource, int?> selector);
1121  /// <summary>Computes the sum of the sequence of nullable <see           7
    cref="T:System.Int64"></see> values that are obtained by invoking a           7
    transform function on each element of the input sequence.</summary>
1122  /// <param name="source">A sequence of values that are used to calculate a           7
    sum.</param>
1123  /// <param name="selector">A transform function to apply to each element.</           7
    param>
1124  /// <typeparam name="TSource">The type of the elements of source.</           7
    typeparam>
1125  /// <returns>The sum of the projected values.</returns>
1126  /// <exception cref="T:System.ArgumentNullException"><paramref           7
    name="source">source</paramref> or <paramref name="selector">selector</           7
    paramref> is null.</exception>
1127  /// <exception cref="T:System.OverflowException">The sum is larger than <see           7
    cref="F:System.Int64.MaxValue"></see>.</exception>
1128  public static long? Sum<TSource>(this IEnumerable<TSource> source,           7
    Func<TSource, long?> selector);
1129  /// <summary>Computes the sum of the sequence of nullable <see           7
    cref="T:System.Single"></see> values that are obtained by invoking a           7

```

```

transform function on each element of the input sequence.</summary>
1130  /// <param name="source">A sequence of values that are used to calculate a  ↗
      sum.</param>
1131  /// <param name="selector">A transform function to apply to each element.</  ↗
      param>
1132  /// <typeparam name="TSource">The type of the elements of source.</  ↗
      typeparam>
1133  /// <returns>The sum of the projected values.</returns>
1134  /// <exception cref="T:System.ArgumentNullException"><paramref  ↗
      name="source">source</paramref> or <paramref name="selector">selector</  ↗
      paramref> is null.</exception>
1135  public static float Sum<TSource>(this IEnumerable<TSource> source,  ↗
      Func<TSource, float?> selector);
1136  /// <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>
1137  /// <param name="source">A sequence of values that are used to calculate a  ↗
      sum.</param>
1138  /// <param name="selector">A transform function to apply to each element.</  ↗
      param>
1139  /// <typeparam name="TSource">The type of the elements of source.</  ↗
      typeparam>
1140  /// <returns>The sum of the projected values.</returns>
1141  /// <exception cref="T:System.ArgumentNullException"><paramref  ↗
      name="source">source</paramref> or <paramref name="selector">selector</  ↗
      paramref> is null.</exception>
1142  public static float Sum<TSource>(this IEnumerable<TSource> source,  ↗
      Func<TSource, float> selector);
1143  /// <summary>Returns a specified number of contiguous elements from the  ↗
      start of a sequence.</summary>
1144  /// <param name="source">The sequence to return elements from.</param>
1145  /// <param name="count">The number of elements to return.</param>
1146  /// <typeparam name="TSource">The type of the elements of source.</  ↗
      typeparam>
1147  /// <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>
1148  /// <exception cref="T:System.ArgumentNullException"><paramref  ↗
      name="source">source</paramref> is null.</exception>
1149  public static IEnumerable<TSource> Take<TSource>(this IEnumerable<TSource>  ↗
      source, int count);
1150  /// <param name="source"></param>
1151  /// <param name="count"></param>
1152  /// <typeparam name="TSource"></typeparam>
1153  /// <returns></returns>
1154  public static IEnumerable<TSource> TakeLast<TSource>(this  ↗
      IEnumerable<TSource> source, int count);
1155  /// <summary>Returns elements from a sequence as long as a specified  ↗
      condition is true.</summary>
1156  /// <param name="source">A sequence to return elements from.</param>
1157  /// <param name="predicate">A function to test each element for a  ↗
      condition.</param>

```

```

1158    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1159    /// <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>
1160    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
1161    public static IEnumerable<TSource> TakeWhile<TSource>(this
    IEnumerable<TSource> source, Func<TSource, bool> predicate);
1162    /// <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>
1163    /// <param name="source">The sequence to return elements from.</param>
1164    /// <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>
1165    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1166    /// <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>
1167    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
1168    public static IEnumerable<TSource> TakeWhile<TSource>(this
    IEnumerable<TSource> source, Func<TSource, int, bool> predicate);
1169    /// <summary>Performs a subsequent ordering of the elements in a sequence in
    ascending order according to a key.</summary>
1170    /// <param name="source">An <see
    cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to
    sort.</param>
1171    /// <param name="keySelector">A function to extract a key from each
    element.</param>
1172    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1173    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
1174    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
    elements are sorted according to a key.</returns>
1175    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
1176    public static IOrderedEnumerable<TSource> ThenBy<TSource, TKey>(this
    IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector);
1177    /// <summary>Performs a subsequent ordering of the elements in a sequence in
    ascending order by using a specified comparer.</summary>
1178    /// <param name="source">An <see
    cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to
    sort.</param>
1179    /// <param name="keySelector">A function to extract a key from each
    element.</param>

```

```

1180    /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
        param>
1181    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
1182    /// <typeparam name="TKey">The type of the key returned by keySelector.</
        typeparam>
1183    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
        elements are sorted according to a key.</returns>
1184    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref
        name="keySelector">keySelector</paramref> is null.</exception>
1185    public static IOrderedEnumerable<TSource> ThenBy<TSource, TKey>(this
        IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector,
        IComparer<TKey> comparer);
1186    /// <summary>Performs a subsequent ordering of the elements in a sequence in
        descending order, according to a key.</summary>
1187    /// <param name="source">An <see
        cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to
        sort.</param>
1188    /// <param name="keySelector">A function to extract a key from each
        element.</param>
1189    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
1190    /// <typeparam name="TKey">The type of the key returned by keySelector.</
        typeparam>
1191    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
        elements are sorted in descending order according to a key.</returns>
1192    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref
        name="keySelector">keySelector</paramref> is null.</exception>
1193    public static IOrderedEnumerable<TSource> ThenByDescending<TSource, TKey>
        (this IOrderedEnumerable<TSource> source, Func<TSource, TKey>
        keySelector);
1194    /// <summary>Performs a subsequent ordering of the elements in a sequence in
        descending order by using a specified comparer.</summary>
1195    /// <param name="source">An <see
        cref="T:System.Linq.IOrderedEnumerable`1"></see> that contains elements to
        sort.</param>
1196    /// <param name="keySelector">A function to extract a key from each
        element.</param>
1197    /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
        param>
1198    /// <typeparam name="TSource">The type of the elements of source.</
        typeparam>
1199    /// <typeparam name="TKey">The type of the key returned by keySelector.</
        typeparam>
1200    /// <returns>An <see cref="T:System.Linq.IOrderedEnumerable`1"></see> whose
        elements are sorted in descending order according to a key.</returns>
1201    /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref

```



```

        name="keySelector">keySelector</paramref> is null.</exception>
1202     public static IOrderedEnumerable<TSource> ThenByDescending<TSource, TKey>  ↗
        (this IOrderedEnumerable<TSource> source, Func<TSource, TKey> keySelector, ↗
         IComparer<TKey> comparer);
1203     /// <summary>Creates an array from a <see                               ↗
        cref="T:System.Collections.Generic.IEnumerable`1"></see>.</summary>
1204     /// <param name="source">An <see                                       ↗
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to create an ↗
        array from.</param>
1205     /// <typeparam name="TSource">The type of the elements of source.</    ↗
        typeparam>
1206     /// <returns>An array that contains the elements from the input sequence.</ ↗
        returns>
1207     /// <exception cref="T:System.ArgumentNullException"><paramref          ↗
        name="source">source</paramref> is null.</exception>
1208     public static TSource[] ToArray<TSource>(this IEnumerable<TSource> source);
1209     /// <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>
1210     /// <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>
1211     /// <param name="keySelector">A function to extract a key from each     ↗
        element.</param>
1212     /// <typeparam name="TSource">The type of the elements of source.</    ↗
        typeparam>
1213     /// <typeparam name="TKey">The type of the key returned by keySelector.</ ↗
        typeparam>
1214     /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see> ↗
        that contains keys and values.</returns>
1215     /// <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>
1216     /// <exception cref="T:System.ArgumentException"><paramref              ↗
        name="keySelector">keySelector</paramref> produces duplicate keys for two ↗
        elements.</exception>
1217     public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(this  ↗
        IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
1218     /// <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>
1219     /// <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>
1220     /// <param name="keySelector">A function to extract a key from each     ↗
        element.</param>
1221     /// <param name="comparer">An <see                                       ↗
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare ↗

```

```

    keys.</param>
1222    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1223    /// <typeparam name="TKey">The type of the keys returned by keySelector.</
    typeparam>
1224    /// <returns>A <see cref="T:System.Collections.Generic.Dictionary`2"></see>
    that contains keys and values.</returns>
1225    /// <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>
1226    /// <exception cref="T:System.ArgumentException"><paramref
    name="keySelector">keySelector</paramref> produces duplicate keys for two
    elements.</exception>
1227    public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey>(this
    IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    IEqualityComparer<TKey> comparer);
1228    /// <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>
1229    /// <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>
1230    /// <param name="keySelector">A function to extract a key from each
    element.</param>
1231    /// <param name="elementSelector">A transform function to produce a result
    element value from each element.</param>
1232    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1233    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
1234    /// <typeparam name="TElement">The type of the value returned by
    elementSelector.</typeparam>
1235    /// <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>
1236    /// <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>
1237    /// <exception cref="T:System.ArgumentException"><paramref
    name="keySelector">keySelector</paramref> produces duplicate keys for two
    elements.</exception>
1238    public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey,
    TElement>(this IEnumerable<TSource> source, Func<TSource, TKey>
    keySelector, Func<TSource, TElement> elementSelector);
1239    /// <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>
1240    /// <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>
1241    /// <param name="keySelector">A function to extract a key from each
    element.</param>
1242    /// <param name="elementSelector">A transform function to produce a result
    element value from each element.</param>
1243    /// <param name="comparer">An <see
    cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
    keys.</param>
1244    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1245    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
1246    /// <typeparam name="TElement">The type of the value returned by
    elementSelector.</typeparam>
1247    /// <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>
1248    /// <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>
1249    /// <exception cref="T:System.ArgumentException"><paramref
    name="keySelector">keySelector</paramref> produces duplicate keys for two
    elements.</exception>
1250    public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey,
    TElement>(this IEnumerable<TSource> source, Func<TSource, TKey>
    keySelector, Func<TSource, TElement> elementSelector,
    IEqualityComparer<TKey> comparer);
1251    /// <param name="source"></param>
1252    /// <typeparam name="TSource"></typeparam>
1253    /// <returns></returns>
1254    public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource>
    source);
1255    /// <param name="source"></param>
1256    /// <param name="comparer"></param>
1257    /// <typeparam name="TSource"></typeparam>
1258    /// <returns></returns>
1259    public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource>
    source, IEqualityComparer<TSource> comparer);
1260    /// <summary>Creates a <see cref="T:System.Collections.Generic.List`1"></
    see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></
    see>.</summary>
1261    /// <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>

```

```

1262    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1263    /// <returns>A <see cref="T:System.Collections.Generic.List`1"></see> that
    contains elements from the input sequence.</returns>
1264    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> is null.</exception>
1265    public static List<TSource> ToList<TSource>(this IEnumerable<TSource>
    source);
1266    /// <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>
1267    /// <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>
1268    /// <param name="keySelector">A function to extract a key from each
    element.</param>
1269    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1270    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
1271    /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains keys
    and values.</returns>
1272    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
1273    public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(this
    IEnumerable<TSource> source, Func<TSource, TKey> keySelector);
1274    /// <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>
1275    /// <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>
1276    /// <param name="keySelector">A function to extract a key from each
    element.</param>
1277    /// <param name="comparer">An <see
    cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
    keys.</param>
1278    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1279    /// <typeparam name="TKey">The type of the key returned by keySelector.</
    typeparam>
1280    /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains keys
    and values.</returns>
1281    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref
    name="keySelector">keySelector</paramref> is null.</exception>
1282    public static ILookup<TKey, TSource> ToLookup<TSource, TKey>(this
    IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    IEqualityComparer<TKey> comparer);
1283    /// <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>
1284    /// <param name="source">The <see      7
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see      7
        cref="T:System.Linq.Lookup`2"></see> from.</param>
1285    /// <param name="keySelector">A function to extract a key from each      7
        element.</param>
1286    /// <param name="elementSelector">A transform function to produce a result      7
        element value from each element.</param>
1287    /// <typeparam name="TSource">The type of the elements of source.</      7
        typeparam>
1288    /// <typeparam name="TKey">The type of the key returned by keySelector.</      7
        typeparam>
1289    /// <typeparam name="TElement">The type of the value returned by      7
        elementSelector.</typeparam>
1290    /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains      7
        values of type <paramref name="TElement">TElement</paramref> selected from      7
        the input sequence.</returns>
1291    /// <exception cref="T:System.ArgumentNullException"><paramref      7
        name="source">source</paramref> or <paramref      7
        name="keySelector">keySelector</paramref> or <paramref      7
        name="elementSelector">elementSelector</paramref> is null.</exception>
1292    public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>(this      7
        IEnumerable<TSource> source, Func<TSource, TKey> keySelector,      7
        Func<TSource, TElement> elementSelector);
1293    /// <summary>Creates a <see cref="T:System.Linq.Lookup`2"></see> from an      7
        <see cref="T:System.Collections.Generic.IEnumerable`1"></see> according to      7
        a specified key selector function, a comparer and an element selector      7
        function.</summary>
1294    /// <param name="source">The <see      7
        cref="T:System.Collections.Generic.IEnumerable`1"></see> to create a <see      7
        cref="T:System.Linq.Lookup`2"></see> from.</param>
1295    /// <param name="keySelector">A function to extract a key from each      7
        element.</param>
1296    /// <param name="elementSelector">A transform function to produce a result      7
        element value from each element.</param>
1297    /// <param name="comparer">An <see      7
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare      7
        keys.</param>
1298    /// <typeparam name="TSource">The type of the elements of source.</      7
        typeparam>
1299    /// <typeparam name="TKey">The type of the key returned by keySelector.</      7
        typeparam>
1300    /// <typeparam name="TElement">The type of the value returned by      7
        elementSelector.</typeparam>
1301    /// <returns>A <see cref="T:System.Linq.Lookup`2"></see> that contains      7
        values of type <paramref name="TElement">TElement</paramref> selected from      7
        the input sequence.</returns>
1302    /// <exception cref="T:System.ArgumentNullException"><paramref      7
        name="source">source</paramref> or <paramref      7
        name="keySelector">keySelector</paramref> or <paramref      7
        name="elementSelector">elementSelector</paramref> is null.</exception>
1303    public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement>(this      7

```

```

    IEnumerable<TSource> source, Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector, IEqualityComparer<TKey>
    comparer);
1304    /// <summary>Produces the set union of two sequences by using the default
    equality comparer.</summary>
1305    /// <param name="first">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
    elements form the first set for the union.</param>
1306    /// <param name="second">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
    elements form the second set for the union.</param>
1307    /// <typeparam name="TSource">The type of the elements of the input
    sequences.</typeparam>
1308    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
    see> that contains the elements from both input sequences, excluding
    duplicates.</returns>
1309    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="first">first</paramref> or <paramref name="second">second</paramref>
    is null.</exception>
1310    public static IEnumerable<TSource> Union<TSource>(this IEnumerable<TSource>
    first, IEnumerable<TSource> second);
1311    /// <summary>Produces the set union of two sequences by using a specified
    <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</
    summary>
1312    /// <param name="first">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
    elements form the first set for the union.</param>
1313    /// <param name="second">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
    elements form the second set for the union.</param>
1314    /// <param name="comparer">The <see
    cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
    values.</param>
1315    /// <typeparam name="TSource">The type of the elements of the input
    sequences.</typeparam>
1316    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
    see> that contains the elements from both input sequences, excluding
    duplicates.</returns>
1317    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="first">first</paramref> or <paramref name="second">second</paramref>
    is null.</exception>
1318    public static IEnumerable<TSource> Union<TSource>(this IEnumerable<TSource>
    first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
1319    /// <summary>Filters a sequence of values based on a predicate.</summary>
1320    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to filter.</
    param>
1321    /// <param name="predicate">A function to test each element for a
    condition.</param>
1322    /// <typeparam name="TSource">The type of the elements of source.</
   typeparam>
1323    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></

```



```

    see> that contains elements from the input sequence that satisfy the
    condition.</returns>
1324    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
1325    public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource>
    source, Func<TSource, bool> predicate);
1326    /// <summary>Filters a sequence of values based on a predicate. Each
    element's index is used in the logic of the predicate function.</summary>
1327    /// <param name="source">An <see
    cref="T:System.Collections.Generic.IEnumerable`1"></see> to filter.</
    param>
1328    /// <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>
1329    /// <typeparam name="TSource">The type of the elements of source.</
    typeparam>
1330    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
    see> that contains elements from the input sequence that satisfy the
    condition.</returns>
1331    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="source">source</paramref> or <paramref name="predicate">predicate</
    paramref> is null.</exception>
1332    public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource>
    source, Func<TSource, int, bool> predicate);
1333    /// <summary>Applies a specified function to the corresponding elements of
    two sequences, producing a sequence of the results.</summary>
1334    /// <param name="first">The first sequence to merge.</param>
1335    /// <param name="second">The second sequence to merge.</param>
1336    /// <param name="resultSelector">A function that specifies how to merge the
    elements from the two sequences.</param>
1337    /// <typeparam name="TFirst">The type of the elements of the first input
    sequence.</typeparam>
1338    /// <typeparam name="TSecond">The type of the elements of the second input
    sequence.</typeparam>
1339    /// <typeparam name="TResult">The type of the elements of the result
    sequence.</typeparam>
1340    /// <returns>An <see cref="T:System.Collections.Generic.IEnumerable`1"></
    see> that contains merged elements of two input sequences.</returns>
1341    /// <exception cref="T:System.ArgumentNullException"><paramref
    name="first">first</paramref> or <paramref name="second">second</paramref>
    is null.</exception>
1342    public static IEnumerable<TResult> Zip<TFirst, TSecond, TResult>(this
    IEnumerable<TFirst> first, IEnumerable<TSecond> second, Func<TFirst,
    TSecond, TResult> resultSelector);
1343    }
1344 }
1345

```