:mod:`bisect` --- Array bisection algorithm

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 4)

Unknown directive type "module".

```
.. module:: bisect
   :synopsis: Array bisection algorithms for binary searching.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 6)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Fred L. Drake, Jr. <fdrake@acm.org>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 7)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Raymond Hettinger <python at rcn.com>
```

Source code: :source:`Lib/bisect.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 10); *backlink*

Unknown interpreted text role "source".

This module provides support for maintaining a list in sorted order without having to sort the list after each insertion. For long lists of items with expensive comparison operations, this can be an improvement over the more common approach. The module is called :mod:`bisect` because it uses a basic bisection algorithm to do its work. The source code may be most useful as a working example of the algorithm (the boundary conditions are already right!).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 14); *backlink*

Unknown interpreted text role "mod".

The following functions are provided:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst, line 24)

Unknown directive type "function".

```
.. function:: bisect_left(a, x, lo=0, hi=len(a), *, key=None)

   Locate the insertion point for *x* in *a* to maintain sorted order.
   The parameters *lo* and *hi* may be used to specify a subset of the list
   which should be considered; by default the entire list is used.  If *x* is
   already present in *a*, the insertion point will be before (to the left of)
   any existing entries.  The return value is suitable for use as the first
   parameter to ``list.insert()`` assuming that *a* is already sorted.

   The returned insertion point *i* partitions the array *a* into two halves so
   that ``all(val < x for val in a[lo : i])`` for the left side and
   ``all(val >= x for val in a[i : hi])`` for the right side.
   *key* specifies a :term:`key function` of one argument that is used to
   extract a comparison key from each input element.  The default value is
   ``None`` (compare the elements directly).

   .. versionchanged:: 3.10
      Added the *key* parameter.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 45)

Unknown directive type "function".

```
.. function:: bisect_right(a, x, lo=0, hi=len(a), *, key=None)
              bisect(a, x, lo=0, hi=len(a), *, key=None)

   Similar to :func:`bisect_left`, but returns an insertion point which comes
   after (to the right of) any existing entries of *x* in *a*.

   The returned insertion point *i* partitions the array *a* into two halves so
   that ``all(val <= x for val in a[lo : i])`` for the left side and
   ``all(val > x for val in a[i : hi])`` for the right side.

   *key* specifies a :term:`key function` of one argument that is used to
   extract a comparison key from each input element.  The default value is
   ``None`` (compare the elements directly).

   .. versionchanged:: 3.10
      Added the *key* parameter.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 63)

Unknown directive type "function".

```
.. function:: insort_left(a, x, lo=0, hi=len(a), *, key=None)

   Insert *x* in *a* in sorted order.

   *key* specifies a :term:`key function` of one argument that is used to
   extract a comparison key from each input element.  The default value is
   ``None`` (compare the elements directly).

   This function first runs :func:`bisect_left` to locate an insertion point.
   Next, it runs the :meth:`insert` method on *a* to insert *x* at the
   appropriate position to maintain sort order.

   Keep in mind that the ``O(log n)`` search is dominated by the slow O(n)
   insertion step.

   .. versionchanged:: 3.10
      Added the *key* parameter.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 82)

Unknown directive type "function".

```
.. function:: insort_right(a, x, lo=0, hi=len(a), *, key=None)
              insort(a, x, lo=0, hi=len(a), *, key=None)

   Similar to :func:`insort_left`, but inserting *x* in *a* after any existing
   entries of *x*.

   *key* specifies a :term:`key function` of one argument that is used to
   extract a comparison key from each input element.  The default value is
   ``None`` (compare the elements directly).

   This function first runs :func:`bisect_right` to locate an insertion point.
   Next, it runs the :meth:`insert` method on *a* to insert *x* at the
   appropriate position to maintain sort order.
```

```
    Keep in mind that the ``O(log n)`` search is dominated by the slow O(n)
    insertion step.

    .. versionchanged:: 3.10
       Added the *key* parameter.
```

Performance Notes

When writing time sensitive code using *bisect()* and *insort()*, keep these thoughts in mind:

- Bisection is effective for searching ranges of values. For locating specific values, dictionaries are more performant.
- The *insort()* functions are `O(n)` because the logarithmic search step is dominated by the linear time insertion step.
- The search functions are stateless and discard key function results after they are used. Consequently, if the search functions are used in a loop, the key function may be called again and again on the same array elements. If the key function isn't fast, consider wrapping it with :func:`functools.cache` to avoid duplicate computations. Alternatively, consider searching an array of precomputed keys to locate the insertion point (as shown in the examples section below).

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 115); *backlink*

  Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 123)

Unknown directive type "seealso".

```
.. seealso::

   * `Sorted Collections
     <http://www.grantjenks.com/docs/sortedcollections/>`_ is a high performance
     module that uses *bisect* to managed sorted collections of data.

   * The `SortedCollection recipe
     <https://code.activestate.com/recipes/577197-sortedcollection/>`_ uses
     bisect to build a full-featured collection class with straight-forward search
     methods and support for a key-function.  The keys are precomputed to save
     unnecessary calls to the key function during searches.
```

Searching Sorted Lists

The above :func:`bisect` functions are useful for finding insertion points but can be tricky or awkward to use for common searching tasks. The following five functions show how to transform them into the standard lookups for sorted lists:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 139); *backlink*

Unknown interpreted text role "func".

```
def index(a, x):
    'Locate the leftmost value exactly equal to x'
    i = bisect_left(a, x)
    if i != len(a) and a[i] == x:
        return i
    raise ValueError

def find_lt(a, x):
    'Find rightmost value less than x'
    i = bisect_left(a, x)
    if i:
        return a[i-1]
    raise ValueError

def find_le(a, x):
    'Find rightmost value less than or equal to x'
    i = bisect_right(a, x)
    if i:
        return a[i-1]
    raise ValueError

def find_gt(a, x):
    'Find leftmost value greater than x'
    i = bisect_right(a, x)
    if i != len(a):
        return a[i]
    raise ValueError

def find_ge(a, x):
    'Find leftmost item greater than or equal to x'
    i = bisect_left(a, x)
    if i != len(a):
        return a[i]
    raise ValueError
```

Examples

The :func:`bisect` function can be useful for numeric table lookups. This example uses :func:`bisect` to look up a letter grade for an exam score (say) based on a set of ordered numeric breakpoints: 90 and up is an 'A', 80 to 89 is a 'B', and so on:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 185); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)bisect.rst`, line 185); *backlink*

Unknown interpreted text role "func".

```
>>> def grade(score, breakpoints=[60, 70, 80, 90], grades='FDCBA'):
...     i = bisect(breakpoints, score)
...     return grades[i]
...
>>> [grade(score) for score in [33, 99, 77, 70, 89, 90, 100]]
['F', 'A', 'C', 'C', 'B', 'A', 'A']
```

One technique to avoid repeated calls to a key function is to search a list of precomputed keys to find the index of a record:

```
>>> data = [('red', 5), ('blue', 1), ('yellow', 8), ('black', 0)]
>>> data.sort(key=lambda r: r[1])       # Or use operator.itemgetter(1).
>>> keys = [r[1] for r in data]         # Precompute a list of keys.
>>> data[bisect_left(keys, 0)]
('black', 0)
>>> data[bisect_left(keys, 1)]
('blue', 1)
>>> data[bisect_left(keys, 5)]
('red', 5)
>>> data[bisect_left(keys, 8)]
('yellow', 8)
```