

8.5. bisect — Array bisection algorithm

New in version 2.1.

Source code: [Lib/bisect.py](#)

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 `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!).

The following functions are provided:

`bisect.bisect_left(a, x, lo=0, hi=len(a))`

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.

`bisect.bisect_right(a, x, lo=0, hi=len(a))`

`bisect.bisect(a, x, lo=0, hi=len(a))`

Similar to `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.

`bisect.insort_left(a, x, lo=0, hi=len(a))`

Insert `x` in `a` in sorted order. This is equivalent to `a.insert(bisect.bisect_left(a, x, lo, hi), x)` assuming that `a` is already sorted. Keep in mind that the $O(\log n)$ search is dominated by the slow $O(n)$ insertion step.

`bisect.insort_right(a, x, lo=0, hi=len(a))`

`bisect.insort(a, x, lo=0, hi=len(a))`

Similar to `insort_left()`, but inserting `x` in `a` after any existing entries of `x`.

See also: [SortedCollection recipe](#) that 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.

8.5.1. Searching Sorted Lists

The above `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:

```
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
```

8.5.2. Other Examples

The `bisect()` function can be useful for numeric table lookups. This example uses `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:

```
>>> 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']
```

Unlike the `sorted()` function, it does not make sense for the `bisect()` functions to have `key` or `reversed` arguments because that would lead to an inefficient design (successive calls to `bisect` functions would not “remember” all of the previous key lookups).

Instead, it is better to search a list of precomputed keys to find the index of the record in question:

```
>>> data = [('red', 5), ('blue', 1), ('yellow', 8), ('black', 0)]
>>> data.sort(key=lambda r: r[1])
>>> keys = [r[1] for r in data]           # precomputed 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)
```
