9.5. `bisect` — Array bisection algorithm¶

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`(*list*, *item*[, *lo*[, *hi*]])¶

Locate the proper insertion point for *item* in *list* 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 *item* is already present in *list*, 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()`. This assumes that *list* is already sorted.

New in version 2.1.

`bisect.bisect_right`(*list*, *item*[, *lo*[, *hi*]])¶

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

New in version 2.1.

`bisect.bisect`(...)¶
Alias for `bisect_right()`.

`bisect.insort_left`(*list*, *item*[, *lo*[, *hi*]])¶

Insert *item* in *list* in sorted order. This is equivalent to `list.insert(bisect.bisect_left(list, item, lo, hi), item)`. This assumes that *list* is already sorted.

New in version 2.1.

`bisect.insort_right`(*list*, *item*[, *lo*[, *hi*]])¶

Similar to `insort_left()`, but inserting *item* in *list* after any existing entries of *item*.

New in version 2.1.

`bisect.insort`(...)¶
Alias for `insort_right()`.

9.5.1. Examples¶

The `bisect()` function is generally useful for categorizing numeric data. This example uses `bisect()` to look up a letter grade for an exam total (say) based on a set of ordered numeric breakpoints: 85 and up is an ‘A’, 75..84 is a ‘B’, etc.

```
>>> grades = "FEDCBA"
>>> breakpoints = [30, 44, 66, 75, 85]
>>> from bisect import bisect
>>> def grade(total):
...           return grades[bisect(breakpoints, total)]
...
>>> grade(66)
'C'
>>> map(grade, [33, 99, 77, 44, 12, 88])
['E', 'A', 'B', 'D', 'F', '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 inefficent 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)
```


Previous topic

9.4. `heapq` — Heap queue algorithm

Next topic

9.6. `array` — Efficient arrays of numeric values

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 9. Data Types »