Python Sorted Collections

Grant Jenks
PyCon 2016

Python
Sordid Collections

Grant Jenks
PyCon 2016

A Short Argument for Sorted Collections

8.5. heapq — Heap queue algorithm

This module provides an implementation of the heap queue algorithm, also known as the priority queue algorithm.

8.6. bisect — Array bisection algorithm

This module provides support for maintaining a list in sorted order without having to sort the list after each insertion.

17.7. queue — A synchronized queue class

The queue module implements multi-producer, multi-consumer queues.

class queue.**PriorityQueue**(*maxsize=0*)

import heapq, bisect, queue


I DON'T ALWAYS IMPORT SORTED TYPES
BUT WHEN I DO, I EXPECT THEM IN THE STDLIB
memegenerator.net

May 2016	Programming Language
1	Java
2	C
3	C++
4	C#
5	Python

TIOBE Index

Third-Party Solutions

What are sorted collections types?

SortedList

```
class SortedList(collections.MutableSequence):

    def __init__(self, iterable=(), key=None):

        ...

    def bisect(self, value):

        ...
```

SortedDict

```
class SortedDict(collections.MutableMapping):

    def __init__(self, [key,] *args, **kwargs):

        ...

    def bisect(self, key):

        ...
```

SortedSet

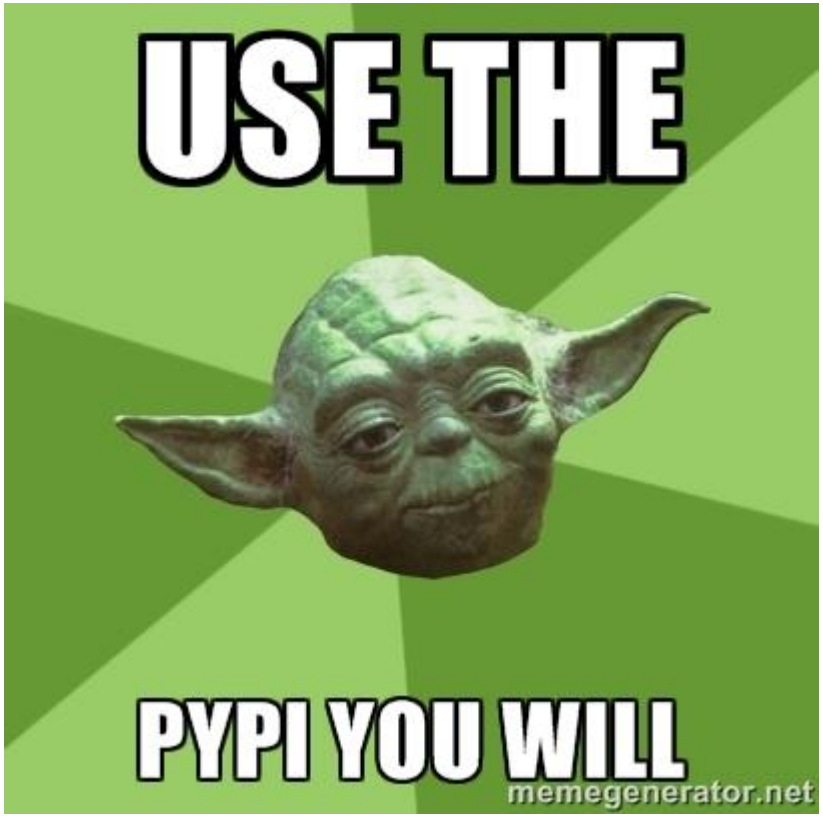
```
class SortedSet(collections.MutableSet, collections.Sequence):

    def __init__(self, iterable=(), key=None):

        ...

    def bisect(self, value):

        ...
```


USE THE
PYPI YOU WILL
memegenerator.net

A Brief History Of Sorted Collections

blist

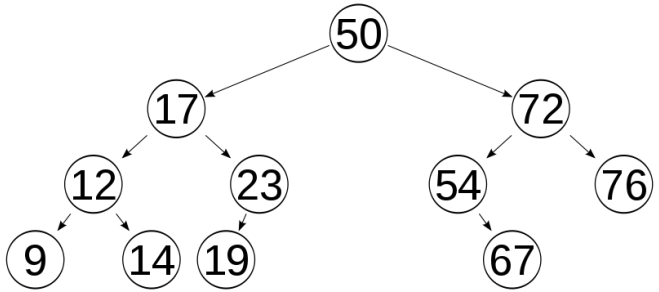
- Daniel Stutzbach; 2006 start, 2014 last PyPI update.
- `blist.blist` B-tree based replacement for `list`.
- Sorted collections based on `blist.blist` type.
- Full-featured, long-standing API.

sortedcollection

- Raymond Hettinger; published on ActiveState, 2010.
- Linked from the Python Standard Library docs.
- Mostly meant for read-only workloads.

bintrees

- Manfred Moitzi; 2010 start, 2015 last PyPI update.
- Multiple tree implementations: Binary, AVL, Red-Black.
- API extends `blist` with tree traversal for slicing by value.

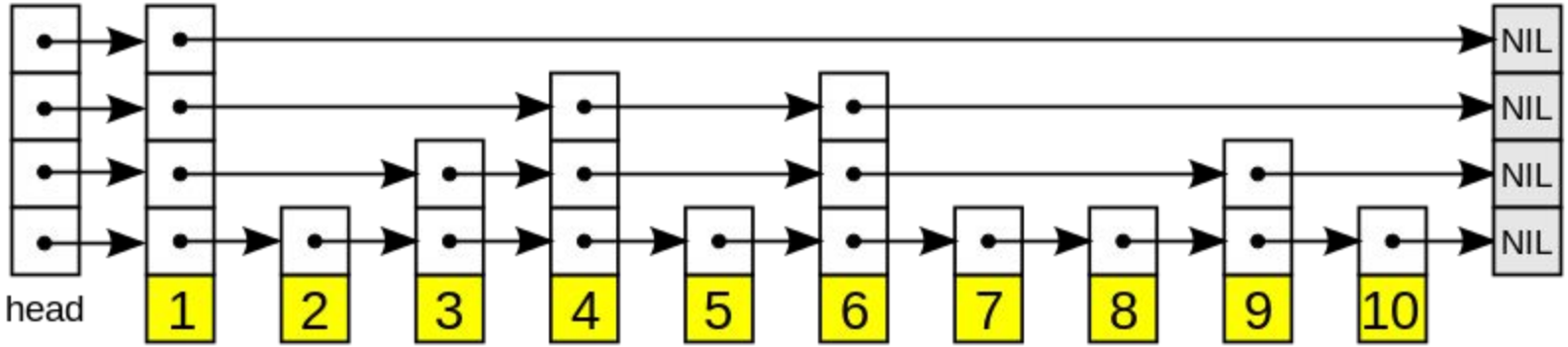


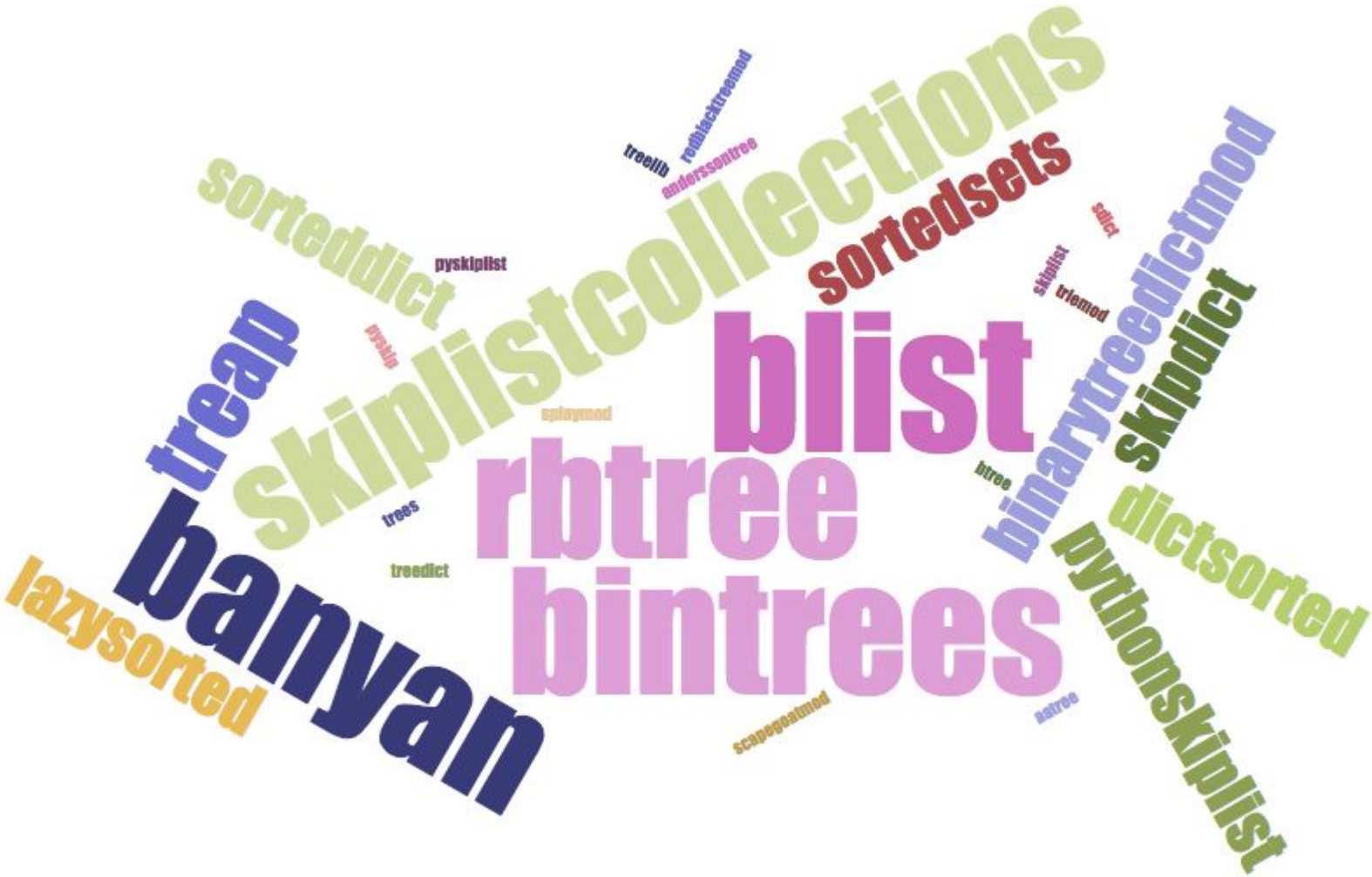
banyan

- Ami Tavory; 2013 start, 2013 last PyPI update.
- Highly optimized C++ implementation.
- Supports tree-augmentation with metadata.

skiplistcollections

- Jakub Stasiak; 2013 start, 2014 last PyPI update.
- Pure-Python with competitive performance.




sortedcollections
skiplistcollections
sortedsets
blist
rbtree
bintrees
treap
banyan
lazysorted
sorteddict
pyskiplist
pyskip
splaymod
trees
treedict
treelib
redblacktreemod
anderssontree
sdict
skiplist
trlemod
binarytreedictmod
skipdict
btree
dictsorted
pythonskiplist
scapegoatmod
aatree

The Missing Battery: SortedContainers

Star 514

SortedContainers provides sorted container types, written in pure-Python and fast as C-extensions.

Give Support

If you or your organization uses SortedContainers, consider financial support:

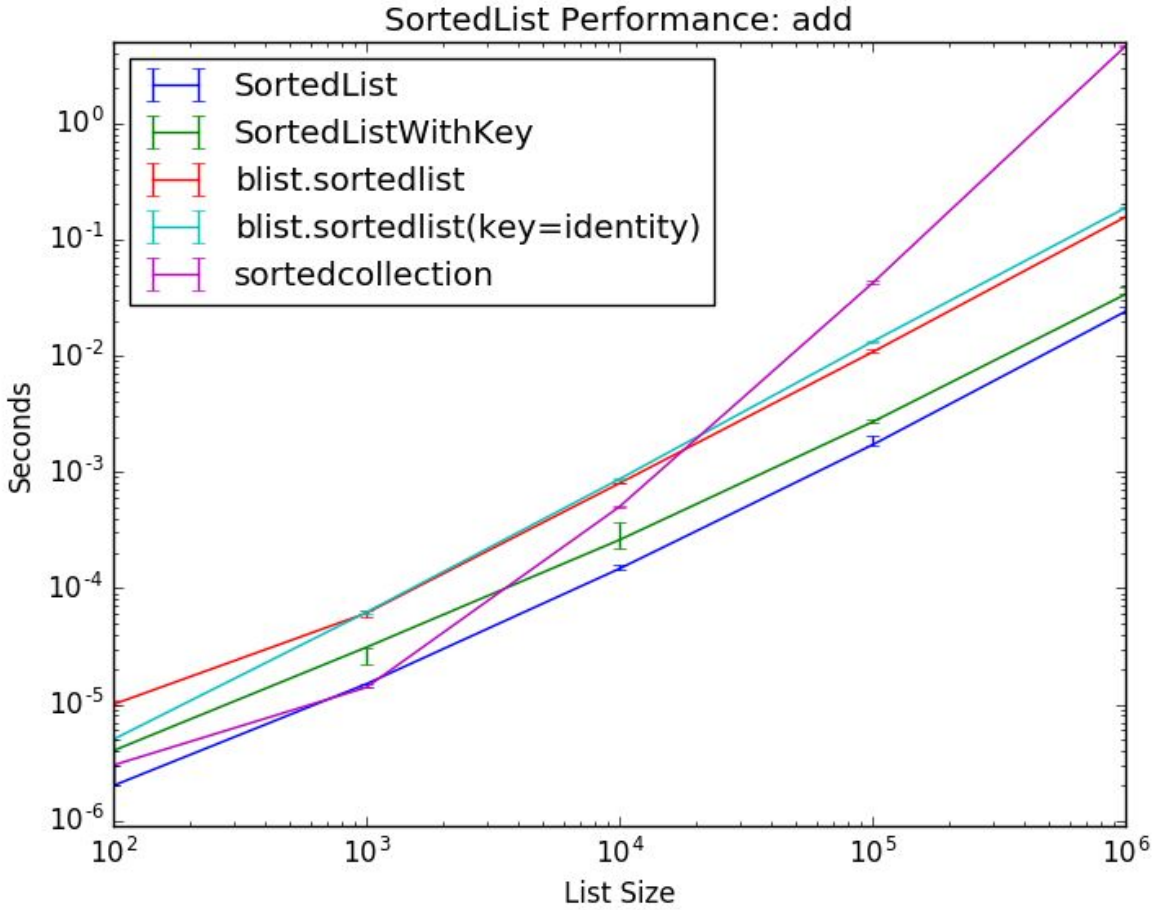
SortedContainers

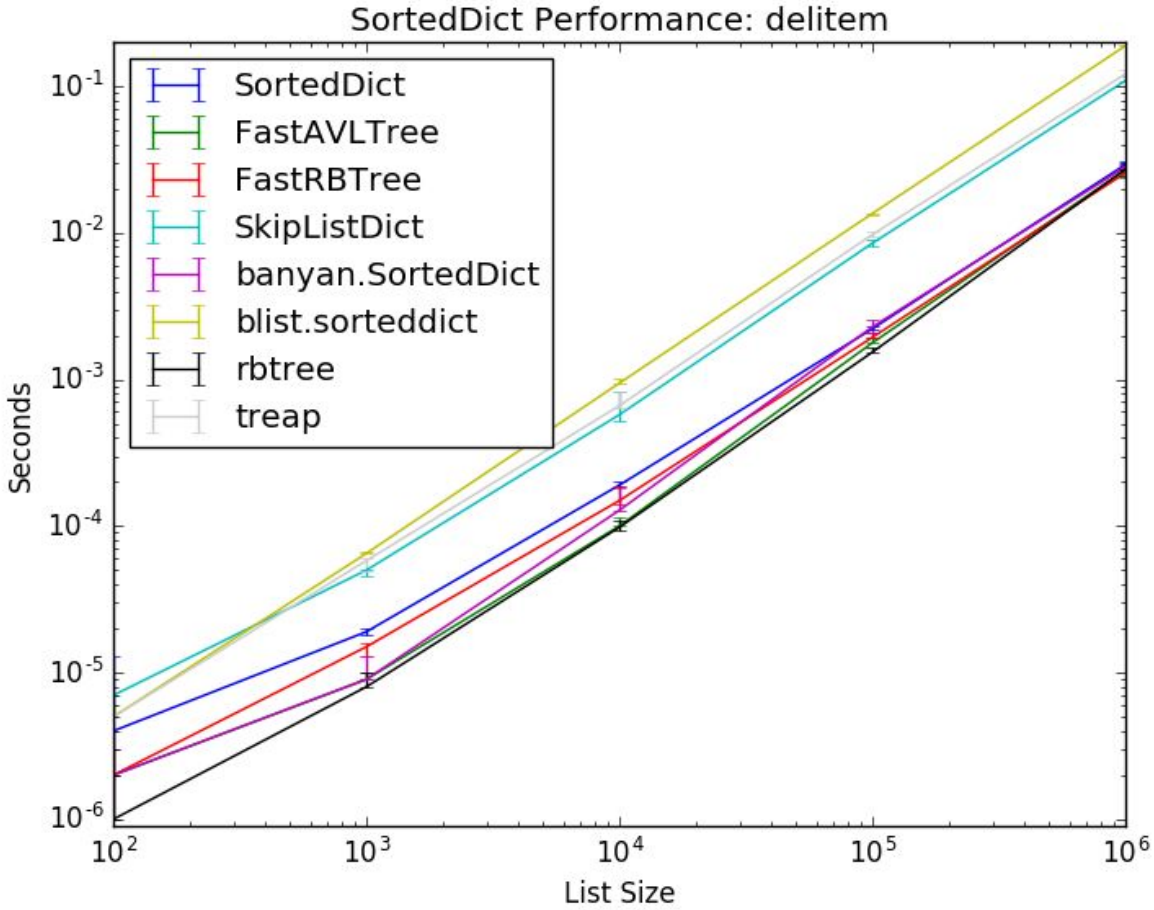
SortedContainers is an *Apache2 Licensed* containers library, written in pure-Python, and fast as C-extensions.

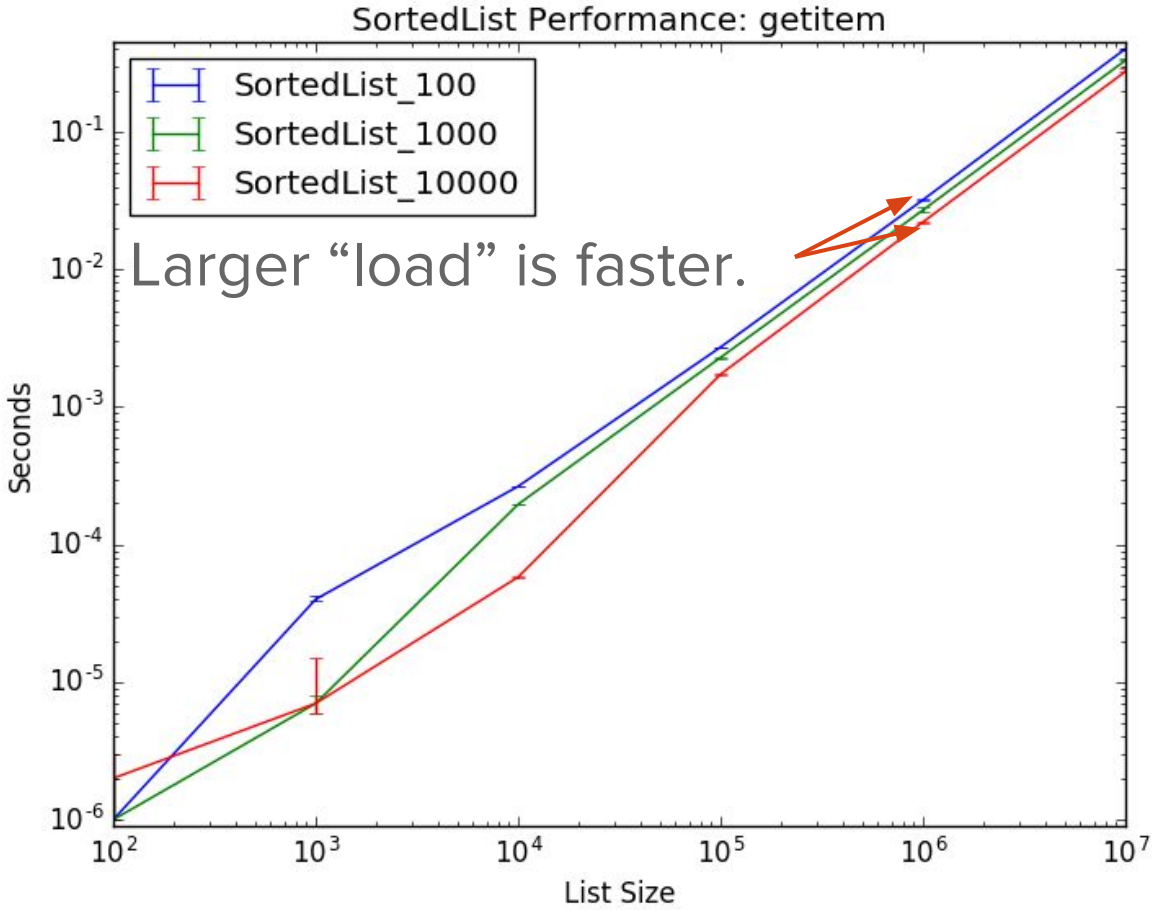
Python's standard library is great until you need a sorted container type. Many will attest that you can get really far without one, but the moment you **really need** a sorted list, dict, or set, you're faced with a dozen different implementations, most using C-extensions without great documentation and benchmarking.

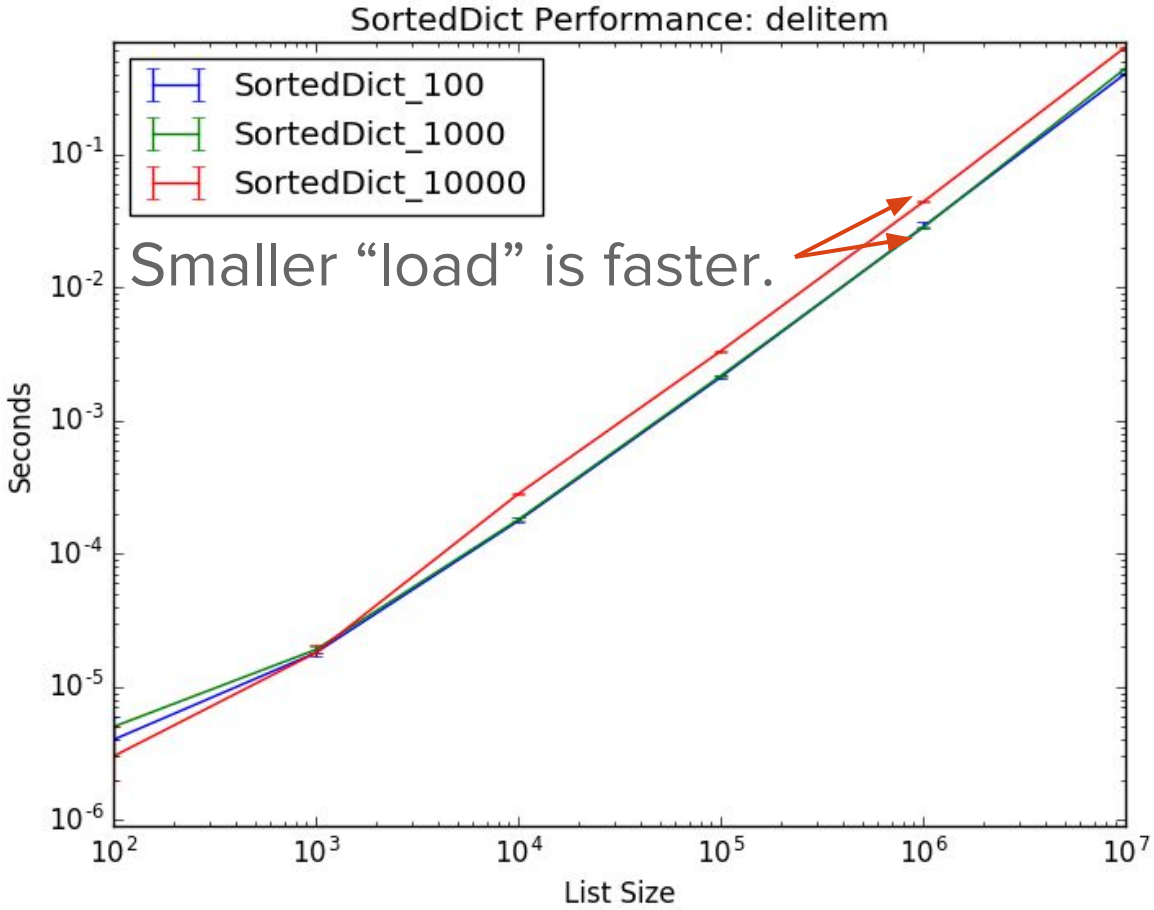
Things shouldn't be this way. Not in Python.

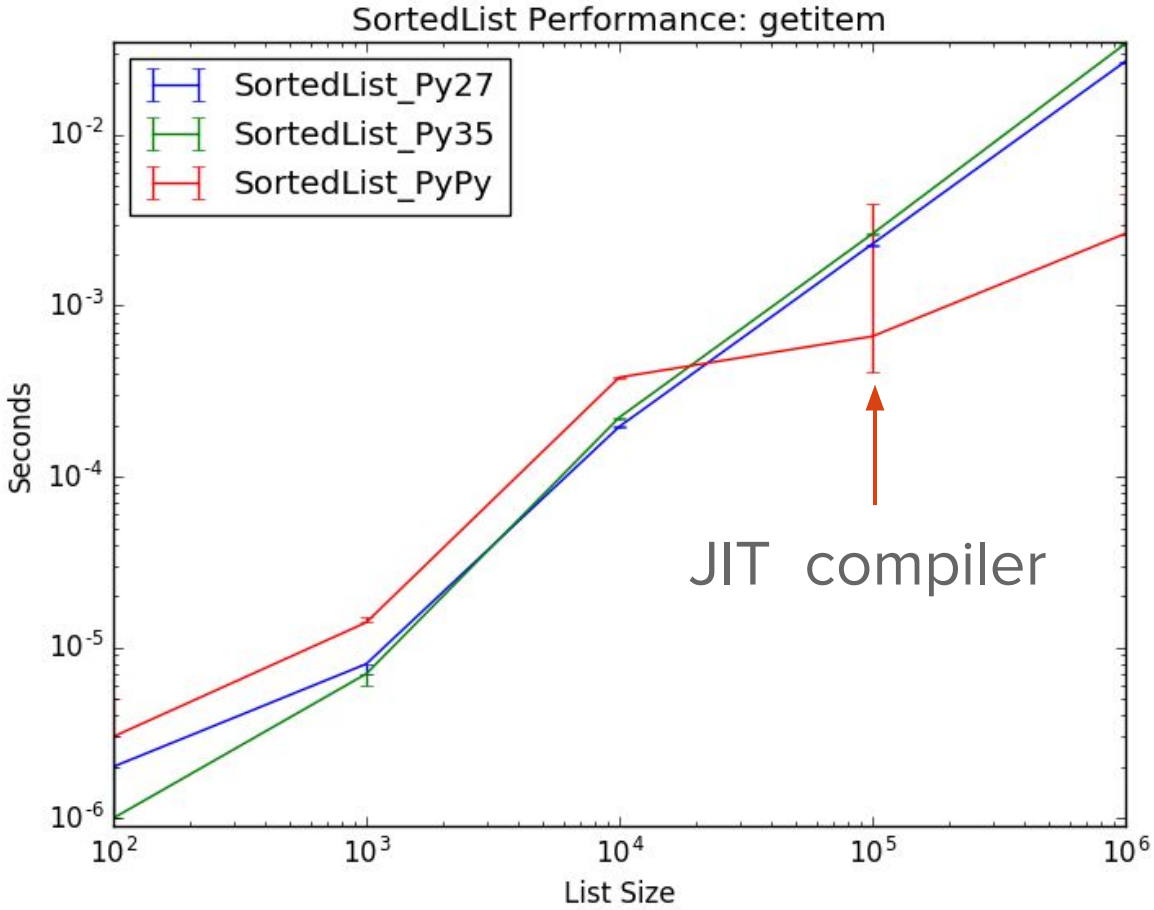
```
>>> sl = sortedcontainers.SortedList(xrange(10000000))
>>> 1234567 in sl
True
>>> sl[7654321]
7654321
>>> sl.add(1234567)
>>> sl.count(1234567)
2
>>> sl *= 3
>>> len(sl)
30000003
```

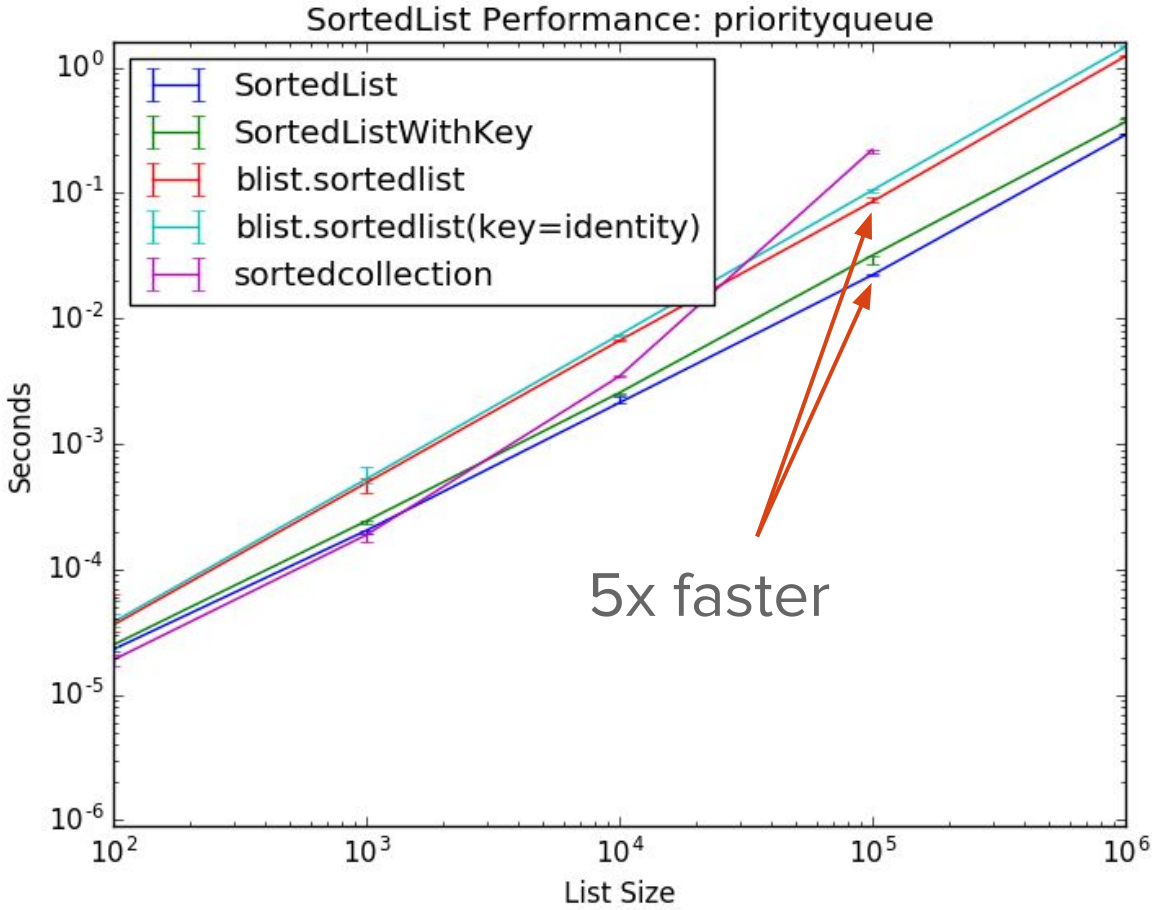

SortedList Performance: add
SortedList
SortedListWithKey
blist.sortedlist
blist.sortedlist(key=identity)
sortedcollection
Seconds
List Size
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}


SortedDict Performance: delitem
SortedDict
FastAVLTree
FastRBTree
SkipListDict
banyan.SortedDict
blist.sorteddict
rbtree
treap
Seconds
List Size
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}


SortedList Performance: getitem
SortedList_100
SortedList_1000
SortedList_10000
Larger “load” is faster.
Seconds
List Size
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}


SortedDict Performance: delitem
SortedDict_100
SortedDict_1000
SortedDict_10000
Smaller “load” is faster.
Seconds
List Size
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}


SortedList Performance: getitem
SortedList_Py27
SortedList_Py35
SortedList_PyPy
Seconds
List Size
JIT compiler


SortedList Performance: priorityqueue
SortedList
SortedListWithKey
blist.sortedlist
blist.sortedlist(key=identity)
sortedcollection
Seconds
List Size
5x faster

Features

```
1  sorted_set.pop()

2  sorted_list.bisect_right('carol')

3  sorted_dict.irange('bob', 'eve')

4  sorted_dict.iloc[-5:]

5  sorted_set.islice(10, 50)
```

Recipes

- ValueSortedDict - dictionary sorted by item value.
- ItemSortedDict - key, value sort order function.
- OrderedDict - insertion order with positional indexing.
- IndexableSet - supports positional indexing.
- `$ pip install sortedcollections`

Alex Martelli, Wikipedia

Good stuff! ... I like the *simple, effective implementation* idea of splitting the sorted containers into smaller "fragments" to avoid the O(N) insertion costs.

Jeff Knupp, Review of SortedContainers

That last part, "fast as C-extensions," was difficult to believe. I would need some sort of *performance comparison* to be convinced this is true. The author includes this in the docs. It is.

Kevin Samuel, Formations Python

I'm quite amazed, not just by the code quality (it's incredibly readable and has more comment than code, wow), but the actual amount of work you put at stuff that is *not* code: documentation, benchmarking, implementation explanations. Even the git log is clean and the unit tests run out of the box on Python 2 and 3.

Testimonials

Under The Hood: SortedContainers

8.6. `bisect` — Array bisection algorithm

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)*)

bisect module

List of Sublists

```
[    # _lists
     [ 0,  1,  2, 3],
     [ 4,  5,  6],
     [ 7,  8,  9, 10, 11, 12],
     [13, 14, 15, 16, 17],
]
```

List of Maxes

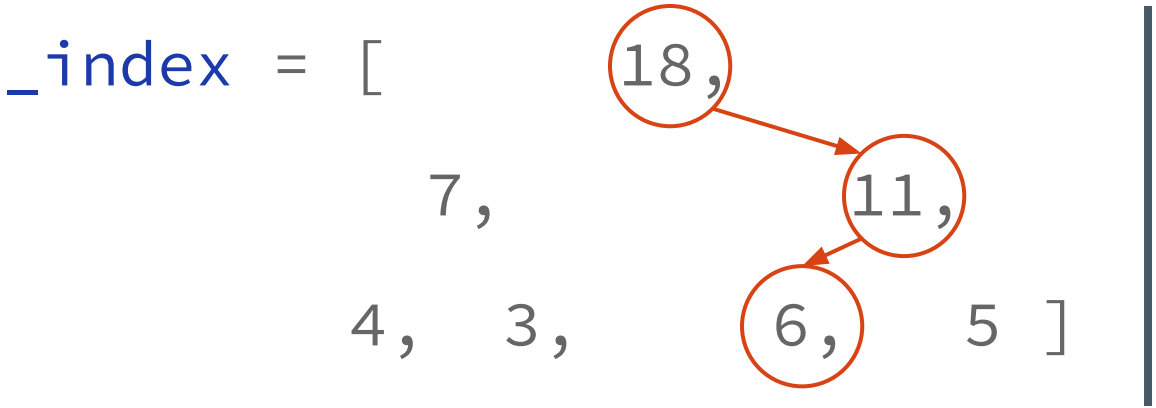
```
[    # _lists
     [ 0,  1,  2, 3],
     [ 4,  5,  6],
     [ 7,  8,  9, 10, 11, 12],
     [13, 14, 15, 16, 17],
]
```

```
[    # _maxes
     3,
     6,
     12,
     17,
]
```

“Jenks” Index

```
1  lengths = [  4,  3,  6,  5 ]
2  pair_wise_sums1 = [  7, 11 ]
3  pair_wise_sums2 = [ 18 ]
4  _index = [ 18,  7, 11,  4,  3,  6,  5 ]
5  _offset = 3
```

Positional Indexing



1 @18, index = 8, position = 0

2 @11, index = 1, position = 2

3 @6, index = 1, position = 5, topindex = 2

Builtin types are *fast*.

SortedList.__contains__

```
1  def __contains__(self, val):
2      _lists = self._lists
3      pos = bisect_left(self._maxes, val)
4      idx = bisect_left(_lists[pos], val)
5      return _lists[pos][idx] == val
```

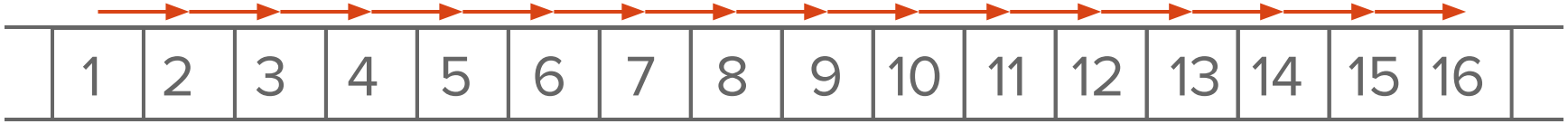
Program ~~in Python~~ your interpreter.

Memory is Tiered

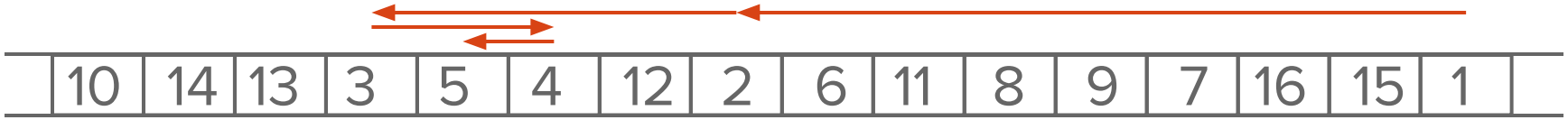
- Registers - dozenish.
- L1 Instruction/Data Cache - 32 KB.
- L2 Cache - 256 KB.
- [L3 Cache (Shared) - 8 MB.]
- Main Memory - Gigabytes.

Memory Access Patterns

- Sequential



- Random



- Data-dependent

list.insert

```
1   for (i = n; --i >= where; )
2       items[i+1] = items[i];
3   Py_INCREF(v);
4   items[where] = v;
```

tiered.

Memory is tiered.

tiered.

tiered.

SortedList.__init__

```
1  values = sorted(iterable)
2  _lists = [values[pos:pos+load] for pos in
3            range(0, len(values), load)]
4  _maxes = [sub[-1] for sub in _lists]
```

SortedSet.add

```
1  def add(self, value):
2      _set, _list = self._set, self._list
3      if value not in _set:
4          _set.add(value)
5          _list.add(value)
```

Cheat, if you can.

Runtime Complexity

- Punchline: O($\sqrt[3]{n}$)
- Billion integers in CPython: 30 GBs.
- Timsort: comparisons are expensive.
- Memory is expensive.
- Performance at Scale: 10,000,000,000

Measure.
Measure.
Measure.

SortedContainers Performance

- Builtin types are ***fast***.
- Program ~~in Python~~ your interpreter.
- Memory is tiered.
- Cheat, if you can.
- Measure. Measure. Measure.

Star 514

SortedContainers provides sorted container types, written in pure-Python and fast as C-extensions.

Give Support

If you or your organization uses SortedContainers, consider financial support:

SortedContainers

SortedContainers is an *Apache2 Licensed* containers library, written in pure-Python, and fast as C-extensions.

Python's standard library is great until you need a sorted container type. Many will attest that you can get really far without one, but the moment you **really need** a sorted list, dict, or set, you're faced with a dozen different implementations, most using C-extensions without great documentation and benchmarking.

Things shouldn't be this way. Not in Python.

```
>>> sl = sortedcontainers.SortedList(xrange(10000000))
>>> 1234567 in sl
True
>>> sl[7654321]
7654321
>>> sl.add(1234567)
>>> sl.count(1234567)
2
>>> sl *= 3
>>> len(sl)
30000003
```