Python Heap Libraries

Shane Kerr
<shane@time-travellers.org>
Amsterdam Python Meetup
2019-09-26

Disclaimer

This presentation contains no information about asyncio, machine learning (called “AI” these days), Docker, Ansible, Kubernetes, Jupyter notebooks, or blockchain.

It does talk about a very old data structure. In Python.

Lets go.

Storing Cached Data

- In DNS, information about names is cached
- Two types of lookup to cache:
  - By *name*, like **`www.ns1.com`**
  - By *expiration*, like 2019-09-25 19:45:13
- For names, the **`dict`** is perfect
- For expiration, the **`heapq`** library is perfect
- Mixing **`dict`** and **`heapq`** is non-trivial

Three Virtues

According to Larry Wall, the original author of the Perl programming language, there are three great virtues of a programmer; Laziness, Impatience and Hubris.

- Laziness: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.
- Impatience: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.
- Hubris: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

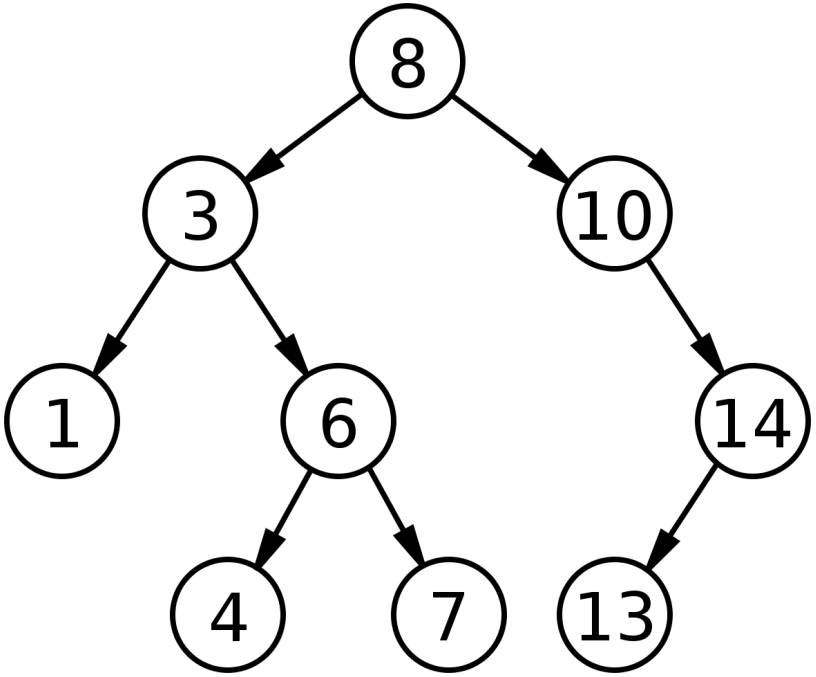
heapdict

- A quick search on PyPI reveals heapdict
  - It’s a heap!
  - It’s a dictionary!
- It’s perfect!
- Or…. is it?
  - Documentation doesn’t match code
  - Not super fast
- Lets start a little investigation….

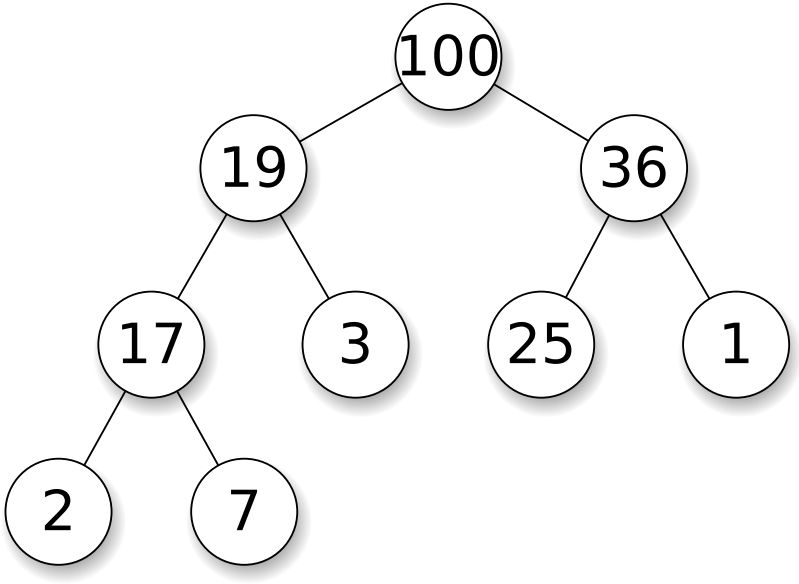
What is a “heap”, anyway?

Kind of like a tree...

In a tree left is smaller, right is bigger



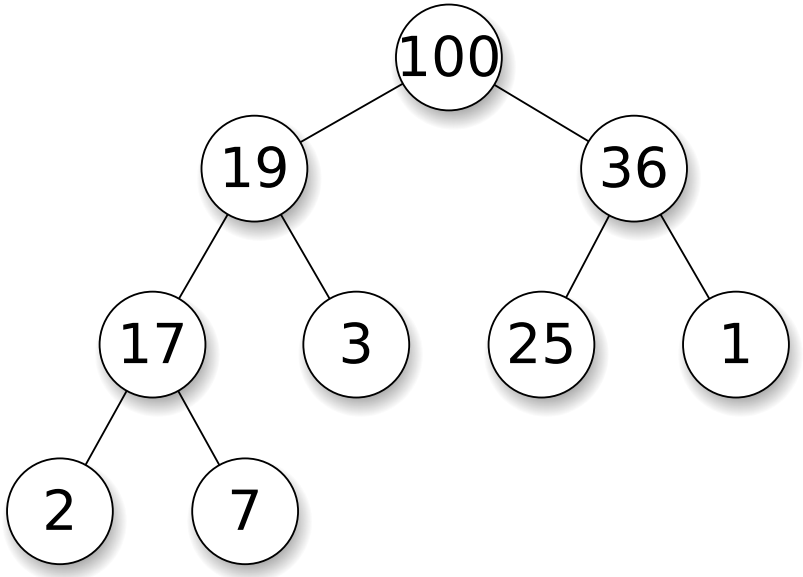
In a heap,
a node is bigger than its children



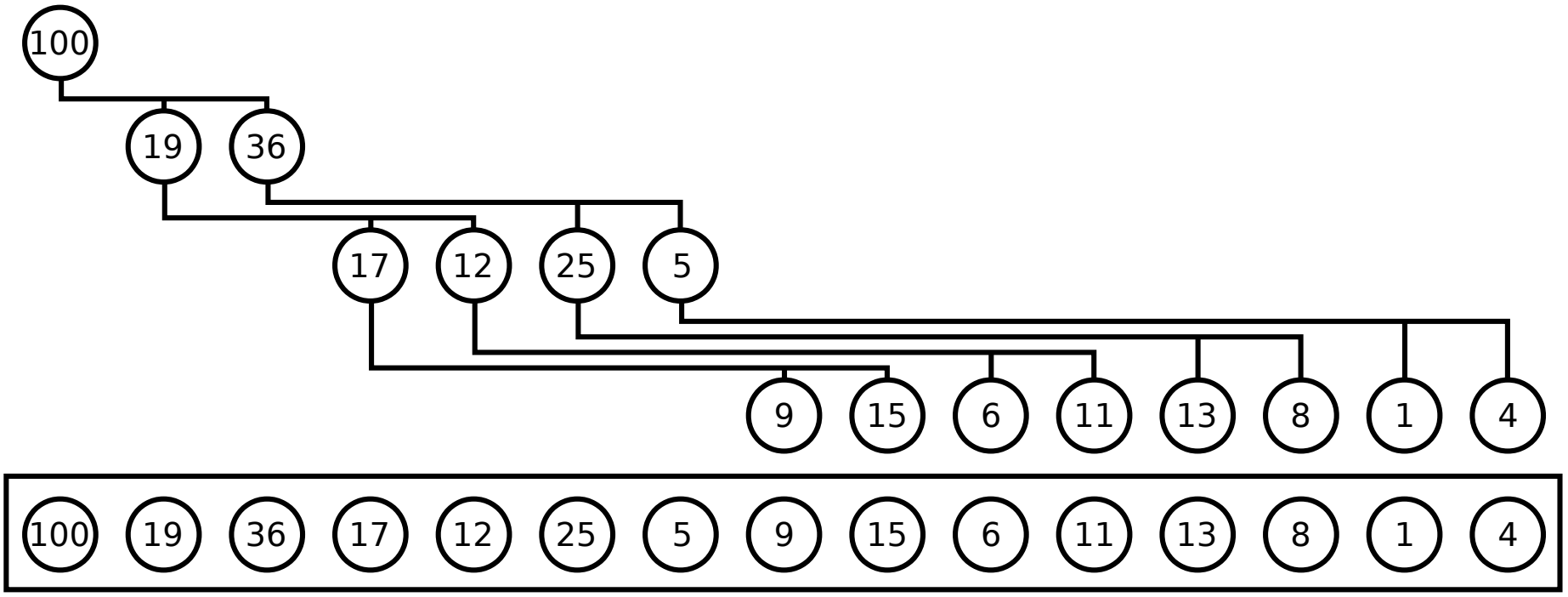
That’s it!

Heap Operations

- Check top of heap – O(1)
- Add to heap – O(logN)
- Remove top of heap – O(logN)
- Increase key – O(logN)
- Remove key – O(logN)
- Replace top of heap – O(logN)
- Create a heap – O(N)



A typical heap layout



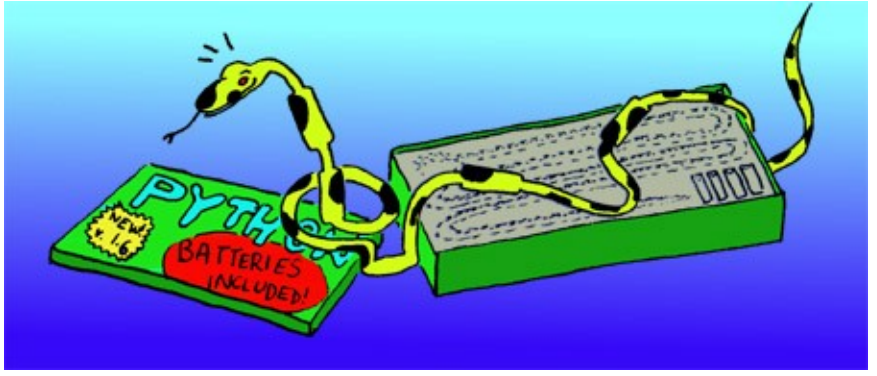
Why would I care about heaps?

- Basis for heapsort
  - O(N) to build the heap
  - Remove top element at O(logN) N times
  - Slightly worse than quicksort, but better worst-case
- Great for priority queues
- Used for A* path finding

Python heaps

heapq

- Batteries included!
- Use any list as a heap
- C implementation
  - Does not use things like custom `__getitem__()`
  - Python implementation available



Heap libraries on PyPI (2017)

```
$ python3 -m pip search heap
```

- HeapDict
- binaryheap
- ~~heapqueue~~*
- fibonacci-heap-mod**

* Library has disappeared from pypi and GitHub
** A *Fibonacci* heap, not a binary heap!

MOAR heap libraries on PyPI (2019)

```
$ python3 -m pip search heap
```

- libheap*
- heapy
- bhpq**
- fibheap***

* One-line whitespace fix needed to use
** Not measured under pypy3, uses Python 3.*really-new* feature
*** A *Fibonacci* heap, not a binary heap!

Accurate measurement is the beginning of all wisdom. – Imhotep

“Meten is weten.” – Dutch saying

Benchmarking is hard… in theory

- Slight changes in memory layout
  - Potential big changes in performance
  - CPU caching especially impacted
- CPU affinity
  - Also hyperthreading
- CPU throttling due to temperature
- And on and on and on...

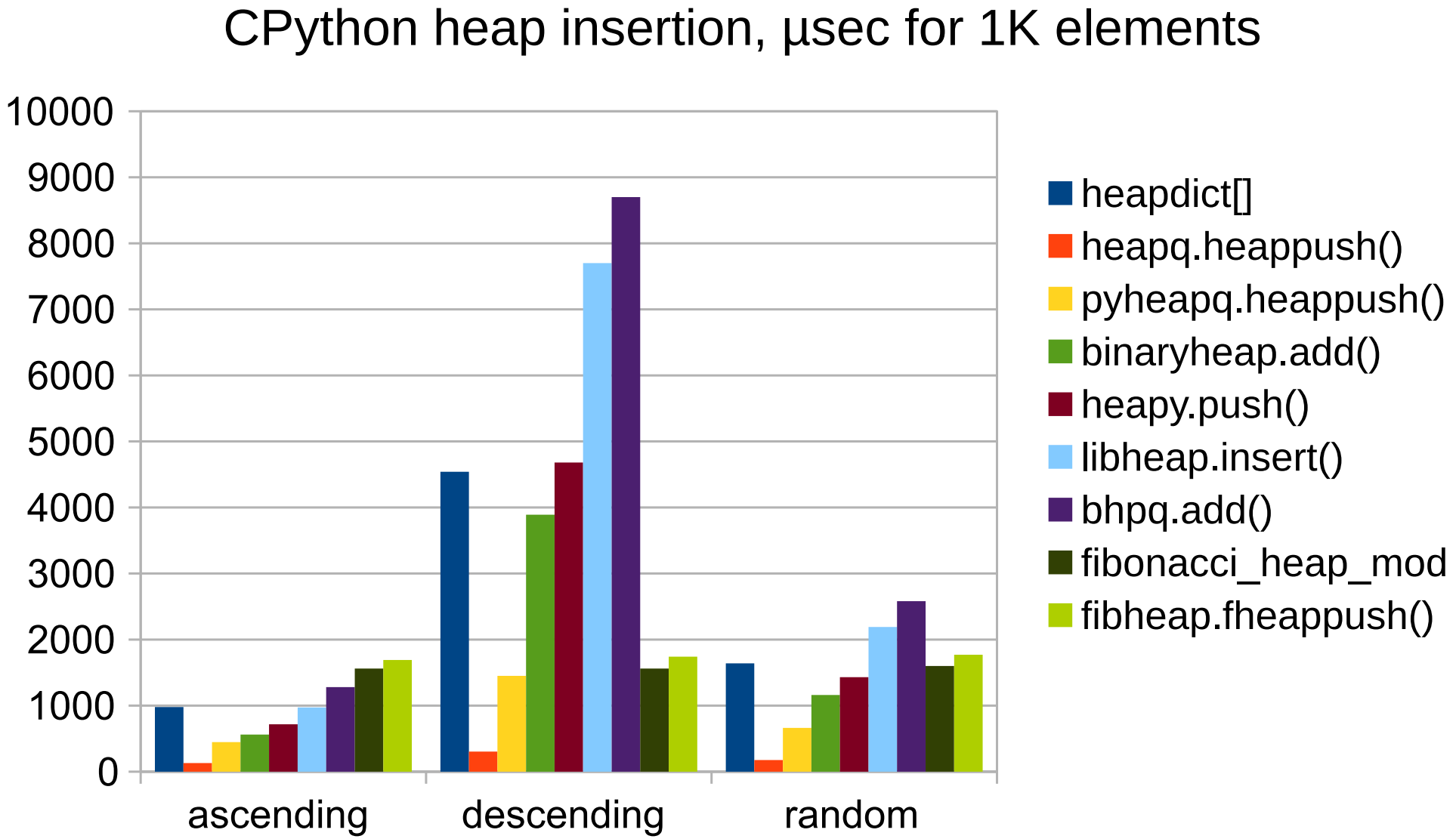
Benchmarking is easy… in practice

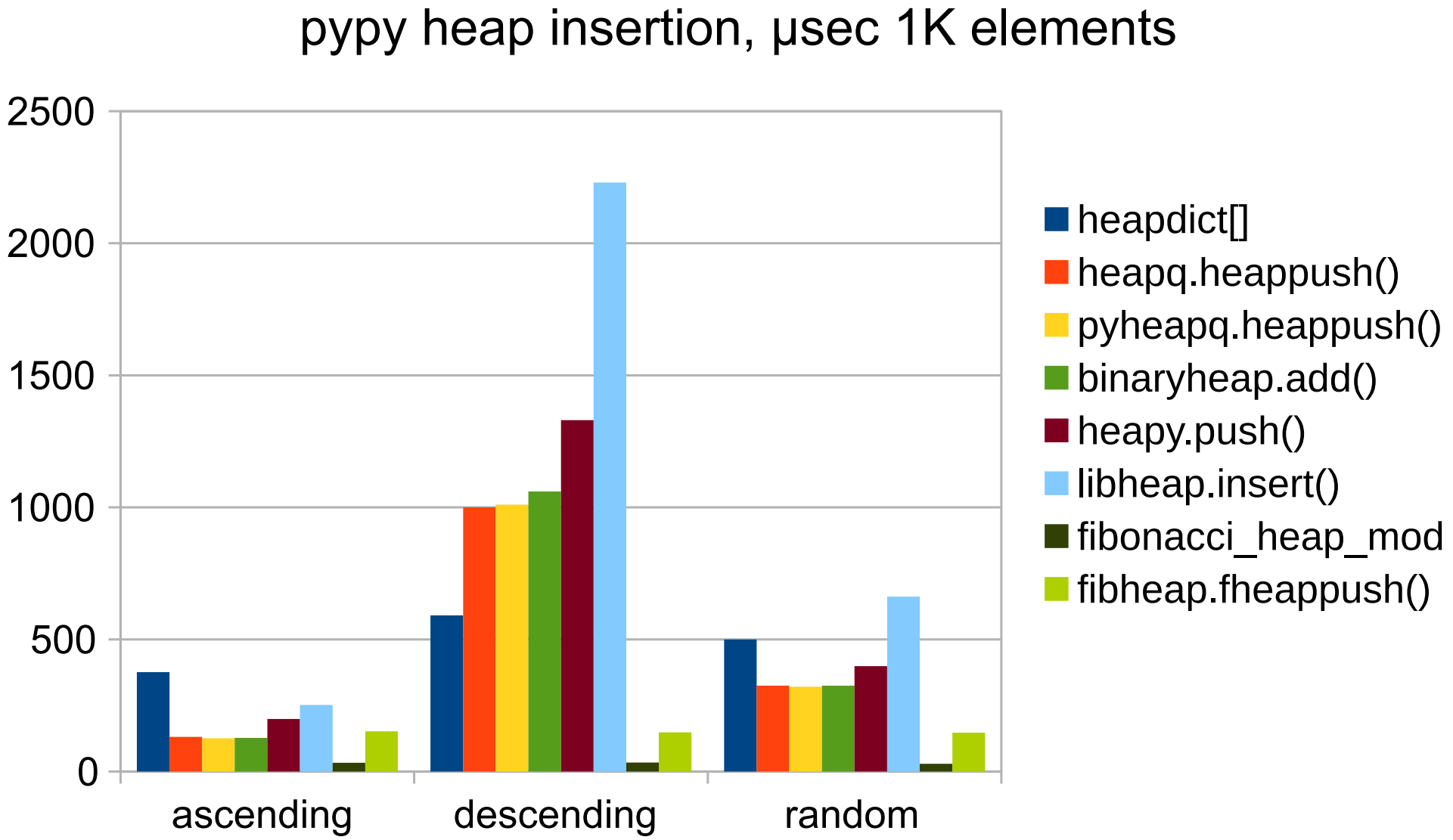
```
import pyperf
```

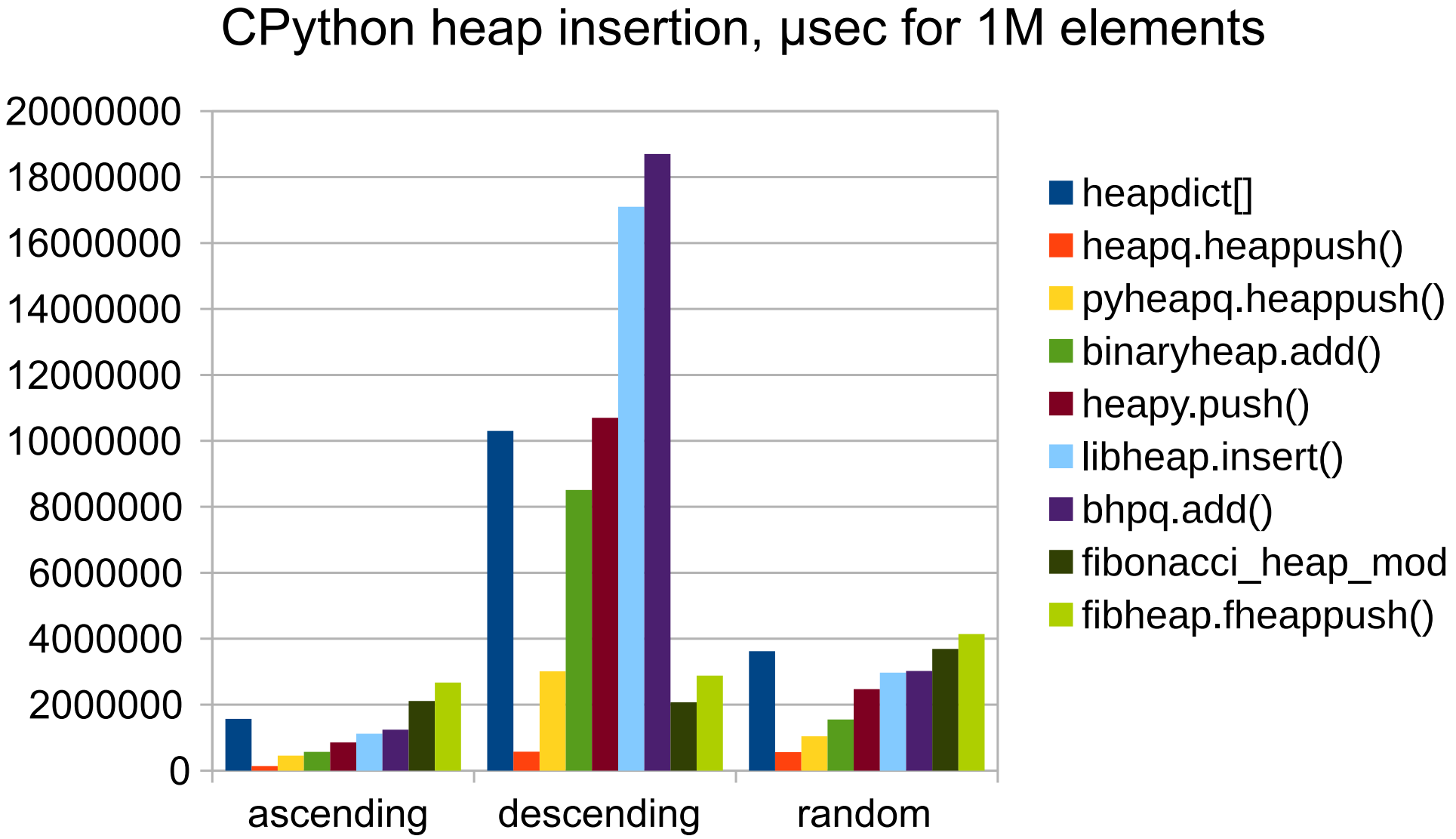
- Put packages in **`requirements.txt`**
  - **`pip install`** everything
- Test expected usage
- Test degenerate cases
  - Worst-case performance important for security

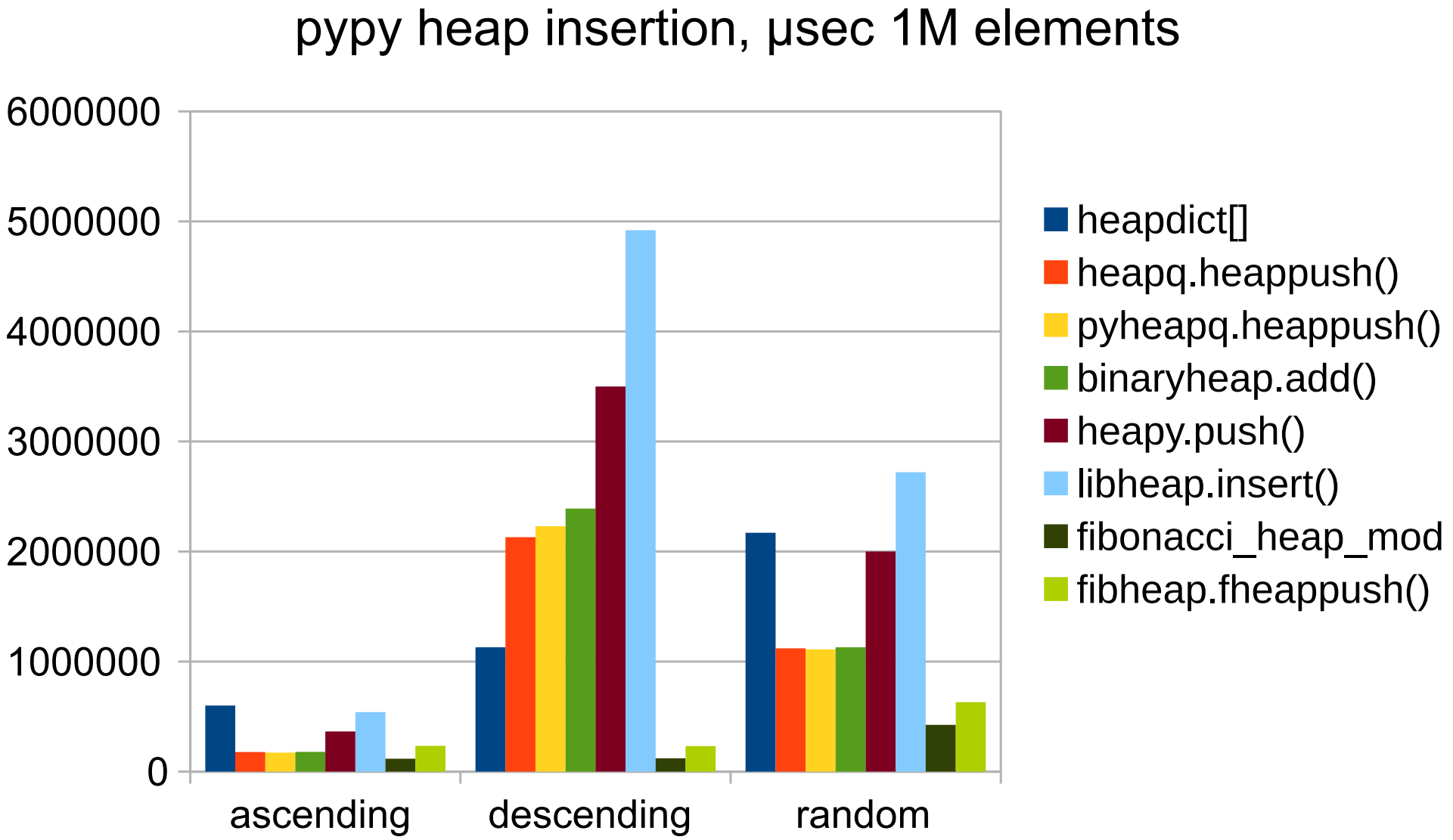
The benchmarks

Adding stuff

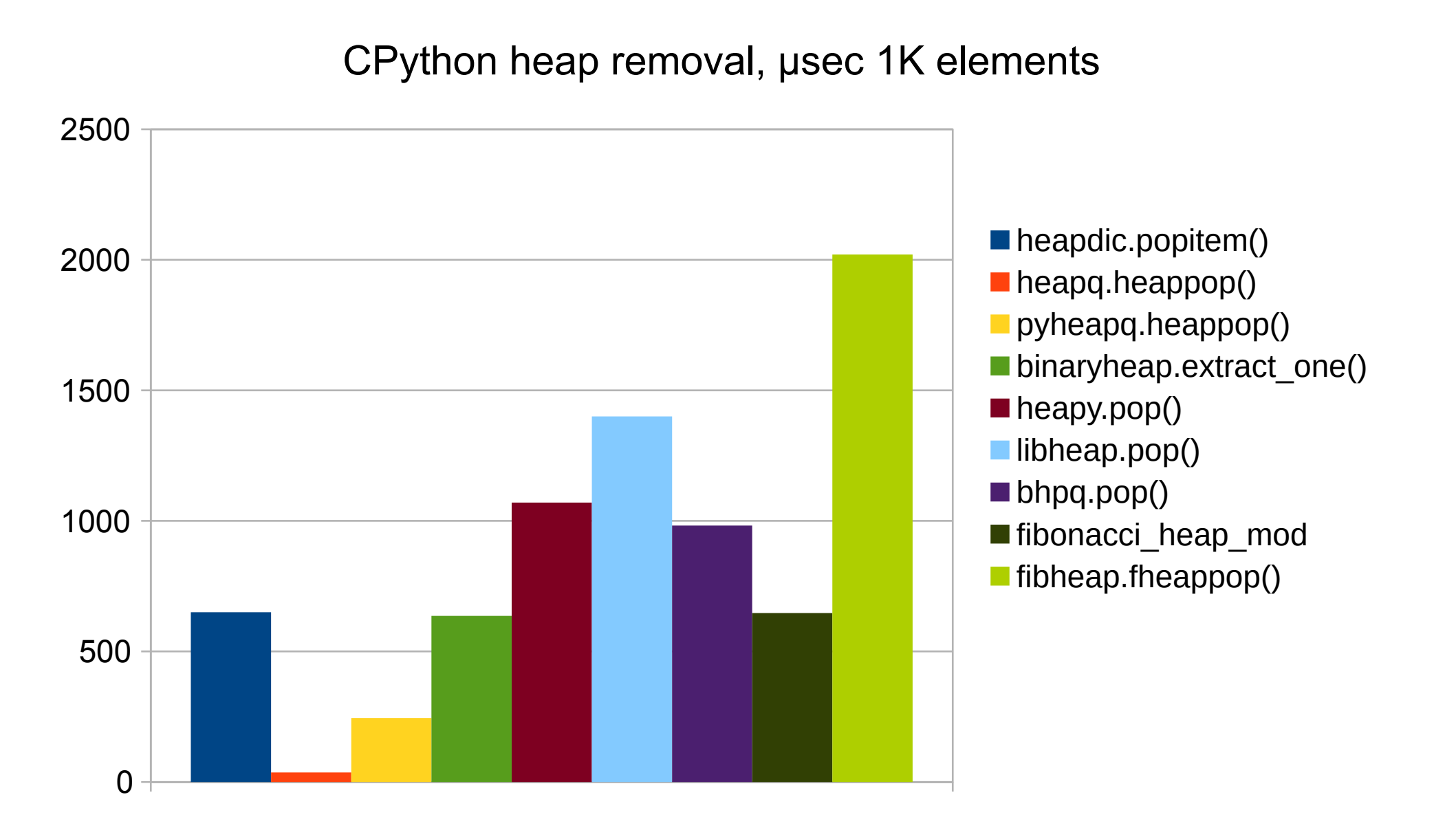

CPython heap insertion, µsec for 1K elements
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
ascending
descending
random
heapdict[]
heapq.heappush()
pyheapq.heappush()
binaryheap.add()
heapy.push()
libheap.insert()
bhpq.add()
fibonacci_heap_mod
fibheap.fheappush()

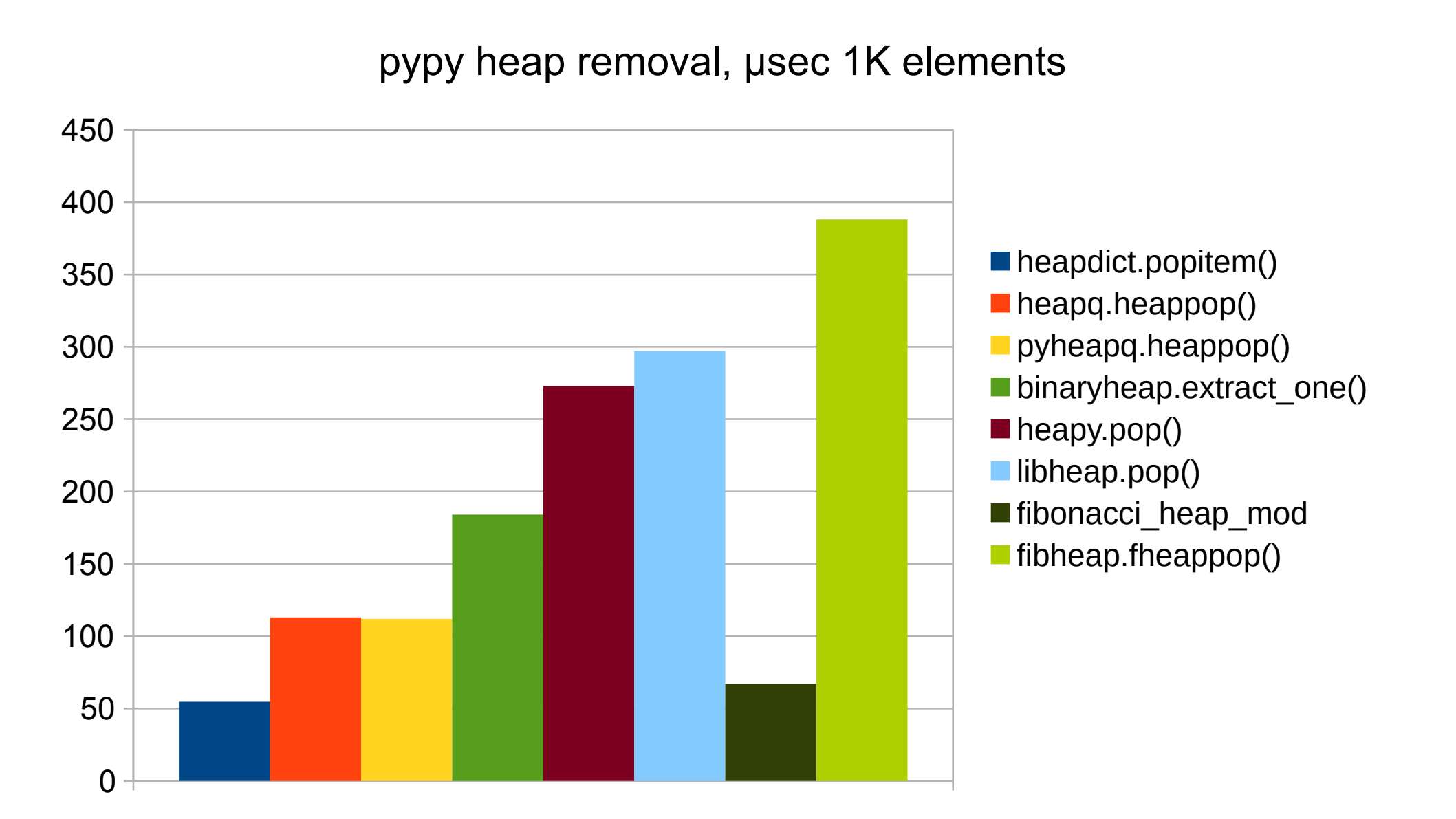

pypy heap insertion, µsec 1K elements
2500
2000
1500
1000
500
0
ascending
descending
random
heapdict[]
heapq.heappush()
pyheapq.heappush()
binaryheap.add()
heapy.push()
libheap.insert()
fibonacci_heap_mod
fibheap.fheappush()

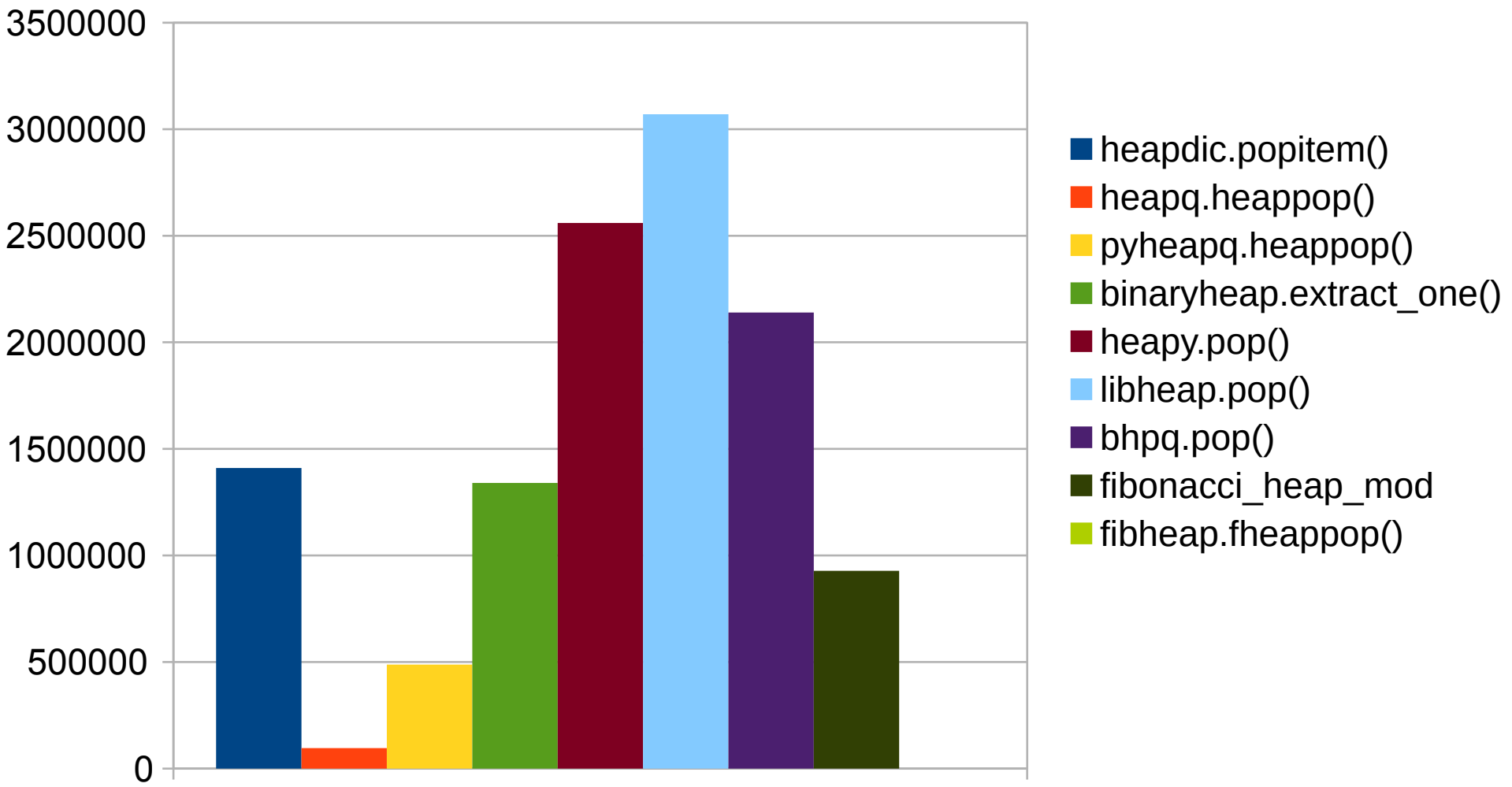

CPython heap insertion, µsec for 1M elements
20000000
18000000
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
ascending
descending
random
heapdict[]
heapq.heappush()
pyheapq.heappush()
binaryheap.add()
heapy.push()
libheap.insert()
bhpq.add()
fibonacci_heap_mod
fibheap.fheappush()

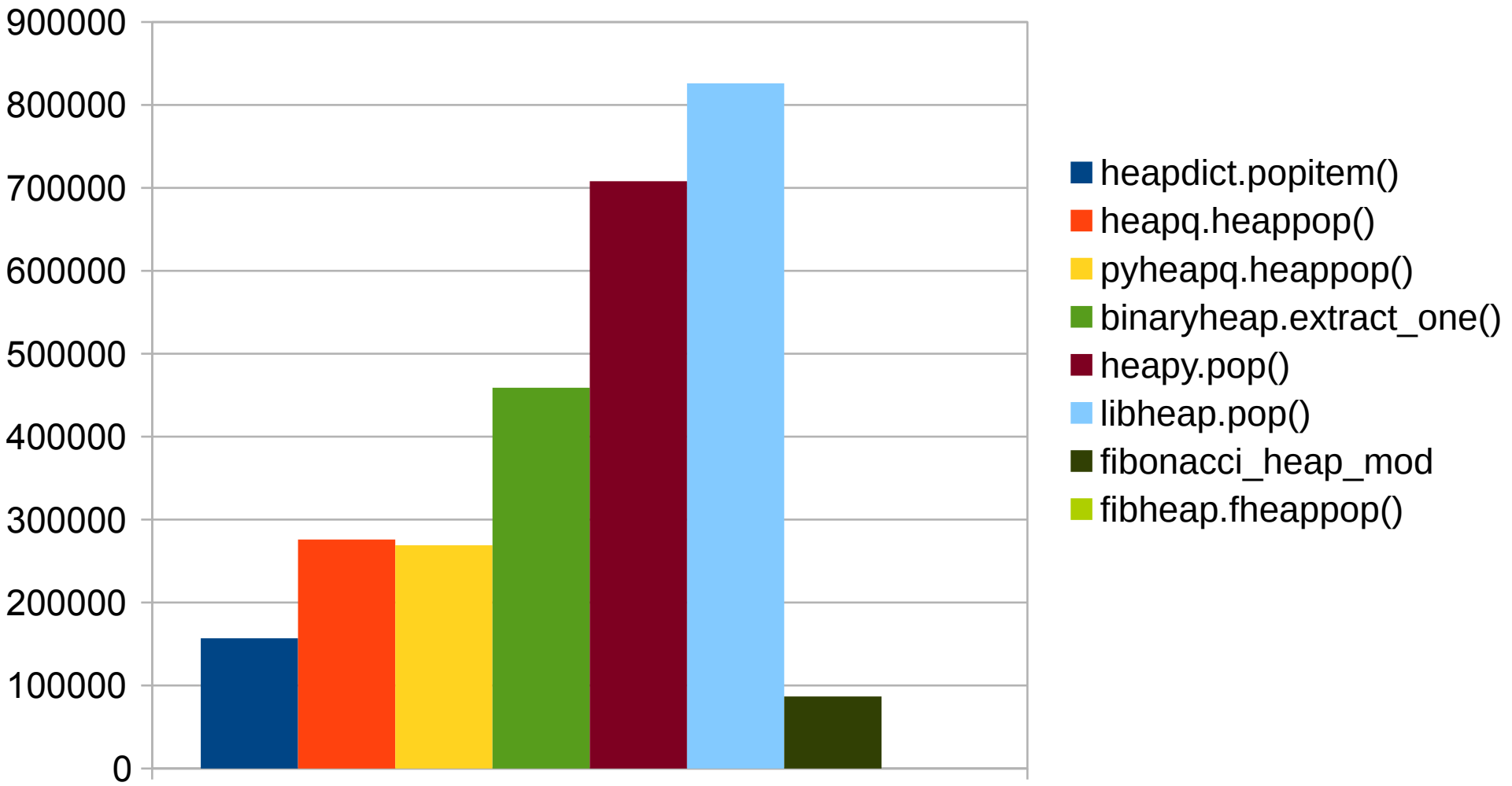

pypy heap insertion, µsec 1M elements
6000000
5000000
4000000
3000000
2000000
1000000
0
ascending
descending
random
heapdict[]
heapq.heappush()
pyheapq.heappush()
binaryheap.add()
heapy.push()
libheap.insert()
fibonacci_heap_mod
fibheap.fheappush()

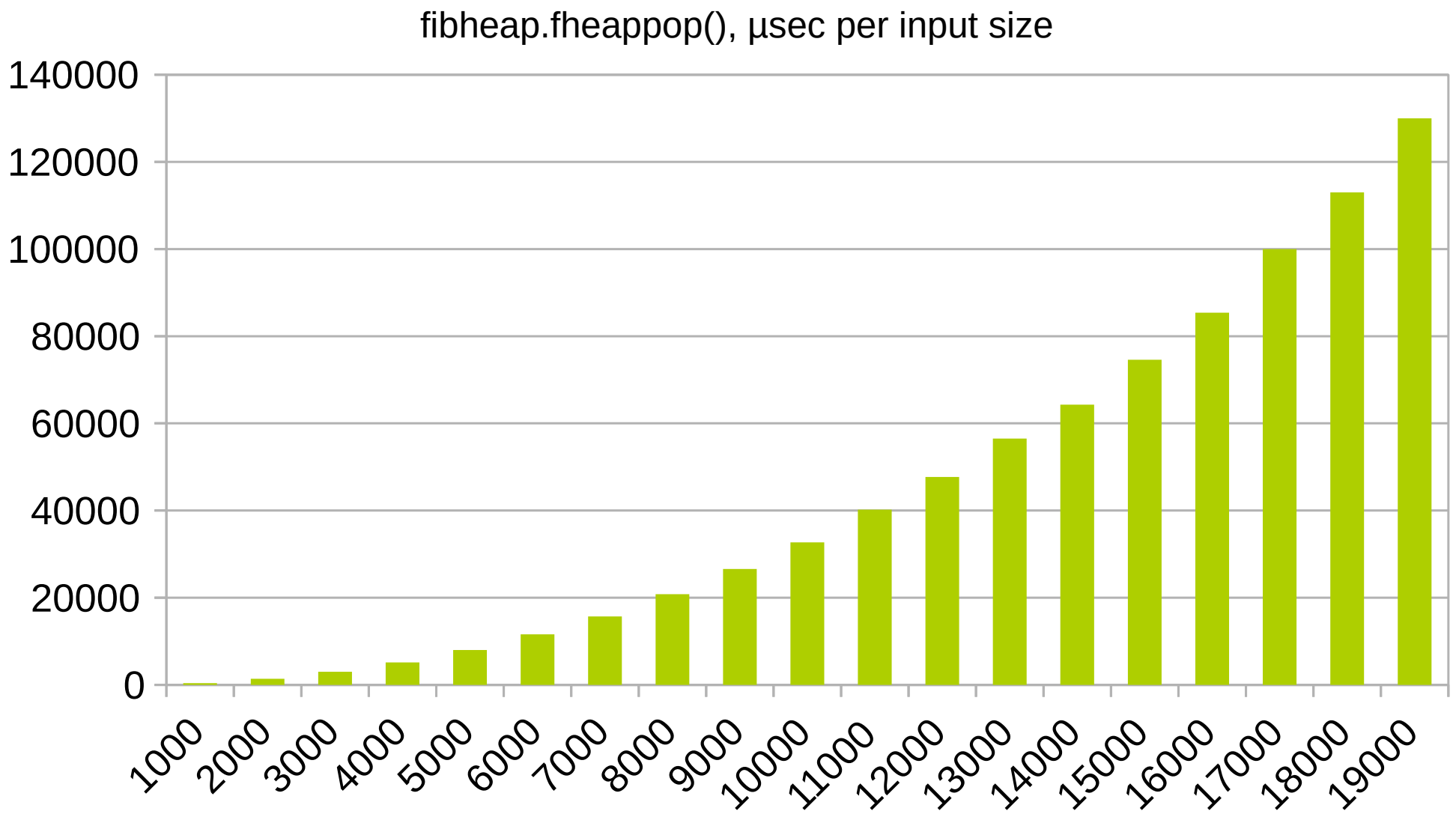
Removing stuff


CPython heap removal, µsec 1K elements
2500
2000
1500
1000
500
0
heapdic.popitem()
heapq.heappop()
pyheapq.heappop()
binaryheap.extract_one()
heapy.pop()
libheap.pop()
bhpq.pop()
fibonacci_heap_mod
fibheap.fheappop()


pypy heap removal, µsec 1K elements
450
400
350
300
250
200
150
100
50
0
heapdict.popitem()
heapq.heappop()
pyheapq.heappop()
binaryheap.extract_one()
heapy.pop()
libheap.pop()
fibonacci_heap_mod
fibheap.fheappop()


CPython heap removal, µsec 1M elements
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
heapdic.popitem()
heapq.heappop()
pyheapq.heappop()
binaryheap.extract_one()
heapy.pop()
libheap.pop()
bhpq.pop()
fibonacci_heap_mod
fibheap.fheappop()


pypy heap removal, µsec 1M elements
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
heapdict.popitem()
heapq.heappop()
pyheapq.heappop()
binaryheap.extract_one()
heapy.pop()
libheap.pop()
fibonacci_heap_mod
fibheap.fheappop()


fibheap.fheappop(), µsec per input size
140000
120000
100000
80000
60000
40000
20000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000

One Obvious Way to Do It?

- Use heapq
- Use pypy
- *Consider* fibonacci-heap-mod if using pypy
  - Fastest (amortized) value
  - May have big performance difference between operations
  - Looks & feels like a ported Java library

Diagram Sources

- Binary tree: https://en.wikipedia.org/wiki/File:Binary_tree.svg
- Heap: https://en.wikipedia.org/wiki/File:Max-Heap.svg
- Heap as array: https://en.wikipedia.org/wiki/File:Heap-as-array.svg
- Python “Batteries Included”: https://commons.wikimedia.org/wiki/File:Python_batteries_included.jpg

Links

- Original blog post
  https://dnshane.wordpress.com/2017/02/14/benchmarking-python-heaps/
- GitHub repository
  https://github.com/shane-kerr/heapbench/tree/ams-python-meetup