Binsort

Serge Kruk

October 12, 2022

1 Problem statement

Given an (unsorted) array of n elements where each element is of the form

AAA#999#AA99#A9#

where the A are characters in the range 'a' to 'z' and the 9 are characters in the range '0' to '9' and the # characters are from the set {@#$%&*}.

Your task is to implement a sort of this array that will be as fast (asymptotically) as possible. (Much better than $n \log n$)

The signature of your function is

```
def fast_sort(a):
  # Sorting the array a much faster than n log(n)...
  return a
```

2 Testing code

I provide here some code to generate random data in the appropriate format so you can test your code properly.

2.1 Generating one key of the appropriate structure

```
from random import choice
import string
def gen_key():
  letters = string.ascii_lowercase
  digits = string.digits
  key=""
  for l in range(3,0,-1):
    key+=''.join(choice(letters) for _ in range(l))
    key+=''.join(choice(digits) for _ in range(l))
    key+=''.join(choice("@#$%&*"))
  return key
```

2.2 For example here are some keys to be sorted

```
for _ in range(3):
  print(gen_key())
```

```
ywg076@yj63*r8#
zdk810%nc93&q7@
wbh734%js22#s6*
```

2.3 To test your code

This is a small test. Run it, after you have tested your code yourself. It may uncover more errors.

```
def test_fast_sort():
  a = [gen_key() for _ in range(3000)]
  if sorted(a) == fast_sort(a):
    print("Success")
  else:
    print("Failure")
test_fast_sort()
```

```
None
```

3 Comparisons

After you are convinced that you have a good solution and you write up the code, tests, runtime analysis and proof of correctness, I encourage you to code (copy from my slides) one of the "optimal" sorts (heap, merge or quick) and contrast the runtime on both small and large instances.