Fast sort

Serge Kruk

October 14, 2024

1 Problem statement

Given an (unsorted) array of n elements where each element has a key of the form `AAA#999#AA99#A9#` here 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 {`@#$%&*`}. (Assume whatever ordering you want for the special characters, as long as you are consistent. Assume alphabetical ordering for the letters and digits.)

The key is at position `k`, an integer in the range 0 to length of one element.

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, k):
  # Sorting the array a on key at position k, faster than n log(n)
  return a
```

1.1 For example here are some elements, key at position 5

```
for _ in range(3):
  print(gen_element(5,60))
```

```
HHHHHrng634#av97$f6*HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
NNNNNnvc005*oe89@y8$NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
LLLLLurv943&it75$d7$LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
```

2 Warning

- You are never allowed to call any Python library function (with obvious exceptions, like `len`, `char`, `ord`). In case of doubt about whether a call is allowed, ask me.

- Every single line of code must be your core, not code you pilfered by googling.

3 Testing code

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

3.1 Generating one key of the appropriate structure

```
from random import choice
import string
letters = string.ascii_lowercase
digits = string.digits
def gen_key():
  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
def gen_element(k,l):
  e = choice(letters).upper()
  key = gen_key()
  return ''.join(e for _ in range(k)) + key + ''.join(e for _ in range(l-k-len(key)))
def gen_elements(k,l,n):
  return [gen_element(k,l) for _ in range(n)]
```

4 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.