Sorting on three elements

Serge Kruk

September 28, 2023

1 Partitioning an array of three distinct elements

The first function you have to write is `sort_three`. It receives two parameters.

```
def sort_three(array, elements)
  # ...
  return p0,p1,p2
```

The first parameter is an array that contains multiple instances of at most three elements, but unsorted and possibly repeated multiple times. For example, the array could be

```
[0,1,1,1,2,0,0,2,2,2,1,1,1,0,0,2,2,1]
```

The second parameter is an array of exactly three elements (the same as the first array) that indicates in which order to sort the first array. For example the second array could be

```
[2,0,1]
```

meaning that you are to sort the first array by putting all `2`, then all `0`, then all `1`.

You are not allowed any additional space. The sort must be in place.

The function sorts the array in time $\Theta(n)$ where n is the length of the array. Hint: use a scanning approach.

The function returns a tuple of three indices p_0, p_1, p_2 such that the array, after it is sorted, satisfies

- $[0..p_0)$ contains the first element
- $[p_0..p_1)$ contains the second element
- $[p_2..n)$ contains the last element

Test, prove correct, and argue that it is linear in time.

1.1 Testing code

To help here is some testing code.

```
from random import choice
def test_sort():
  n=20
  elements=[[0,1,2],[2,0,1],[2,1,0],['a','c','b']]
  for e in elements:
    a=[choice(e) for _ in range(n)]
    (p0,p1,p2)=sort_three(a,e)
    if not all([a[0:p0]==[e[0]]*p0,a[p0:p1]==[e[1]]*(p2-p0),a[p2:]==[e[2]]*(n-p2)]):
      print("Failed on ", a, e)
      return
  print("Success")
```

Now we test

```
test_sort()
```

```
Success
```