Quicksort Partitioning Pseudocode

Algorithms and Data Structures I

In this brief note we will present the pseudocode for the partitioning of the Quicksort algorithm. The partition algorithm is where most of the work is being done in the Quicksort algorithm, because once the input vector (or array) has been partitioned we apply Quicksort to each of the smaller sub-vectors (or sub-arrays).

The partition we present in this module is called the Hoare partition, named after the inventor of Quicksort, Tony Hoare. There are other methods of partitioning; one such method described in module textbook Introduction to Algorithms (Third Edition) by Cormen *et al* (in section 7.1) is called the Lomuto partition, attributed to Nico Lomuto. In your studies you can use whichever partition function you wish as long as it works.

We will assume that the SWAP(*vector*, i, j) function is already defined, which swaps the values in elements i and j in a vector called *vector*. Let's define the partition function based on the Hoare partition:

```
1:  function PARTITION(vector, i, j)
2:      m ← ⌊LENGTH(vector/2)⌋                ▷ this is the mid-point in the vector
3:      pivot ← vector[m]                     ▷ this is the pivot value stored at the mid-point
4:      final ← m                             ▷ this will be the final location in the vector of the pivot value
5:      while i < j do
6:          while vector[i] < pivot do
7:              i ← i + 1                     ▷ this will increase the index on the left until a value should be swapped
8:          end while
9:          while vector[j] > pivot do
10:             j ← j − 1                     ▷ this will decrease the index on the right until a value should be swapped
11:         end while
12:         if i < j then
13:             SWAP(vector, i, j)            ▷ two values swapped at i and j
14:             if i = final then
15:                 final ← j                 ▷ updates the location of the pivot value in the vector if it is being swapped
16:                 i ← i + 1
17:             else if j = final then
18:                 final ← i                 ▷ updates the location of the pivot value in the vector if it is being swapped
19:                 j ← j − 1
20:             else
21:                 i← i + 1
22:                 j ← j − 1
23:             end if
24:         end if
25:     end while
26:     return final                          ▷ final location of the pivot
27: end function
```

We have some comments on the right to help us follow the logic of the function. The important thing to observe is that the variable final keeps track of the location of the pivot value because it might be swapped with other values. So if we call the index f the value returned by the function PARTITION(*vector*, 1, LENGTH[*vector*]) there are now at most two smaller sub-vectors from element 1 to f $-$ 1, and another from element f $+$ 1 to LENGTH[*vector*]. We can then recursively apply the Quicksort algorithm to each of these sub-vectors, thereby partitioning them further.