LAB211 Assignment

Type:	**Short Assignment**
Code:	**J1.S.P0004**
LOC:	**70**
Slot(s):	**1**

Title

Quick sort algorithm.

Background Context

Quicksort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice. On the average, it has O(n log n) complexity, making quicksort suitable for sorting big data volumes. The idea of the algorithm is quite simple and once you realize it, you can write quicksort as fast as bubble sort.

Program Specifications

Design a program that allows users to input the number of array. Generate random integer in number range input. Display unsorted array and sorted array using quick sort.

Function details:

1. Display a screen to prompt users to input a positive decimal number.
   - Users run the program, display a screen to ask users to enter a positive decimal number.
   - Users input a positive decimal number. Then, perform **Function 2**.
2. Display & sort array.
   - Generate random integer in number range for each array element.
   - Display array before and after sorting.

Expectation of User interface:

```
Enter number of array:
10
Unsorted array: [2, 6, 3, 6, 8, 6, 1, 2, 9, 8]
Sorted array: [1, 2, 2, 3, 6, 6, 6, 8, 8, 9]BUILD SUCCESSFUL (total time: 1 second)
```

Guidelines

Algorithm

The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

1. **Choose a pivot value.** We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.
2. **Partition.** Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.
3. **Sort both parts.** Apply quicksort algorithm recursively to the left and the right parts.

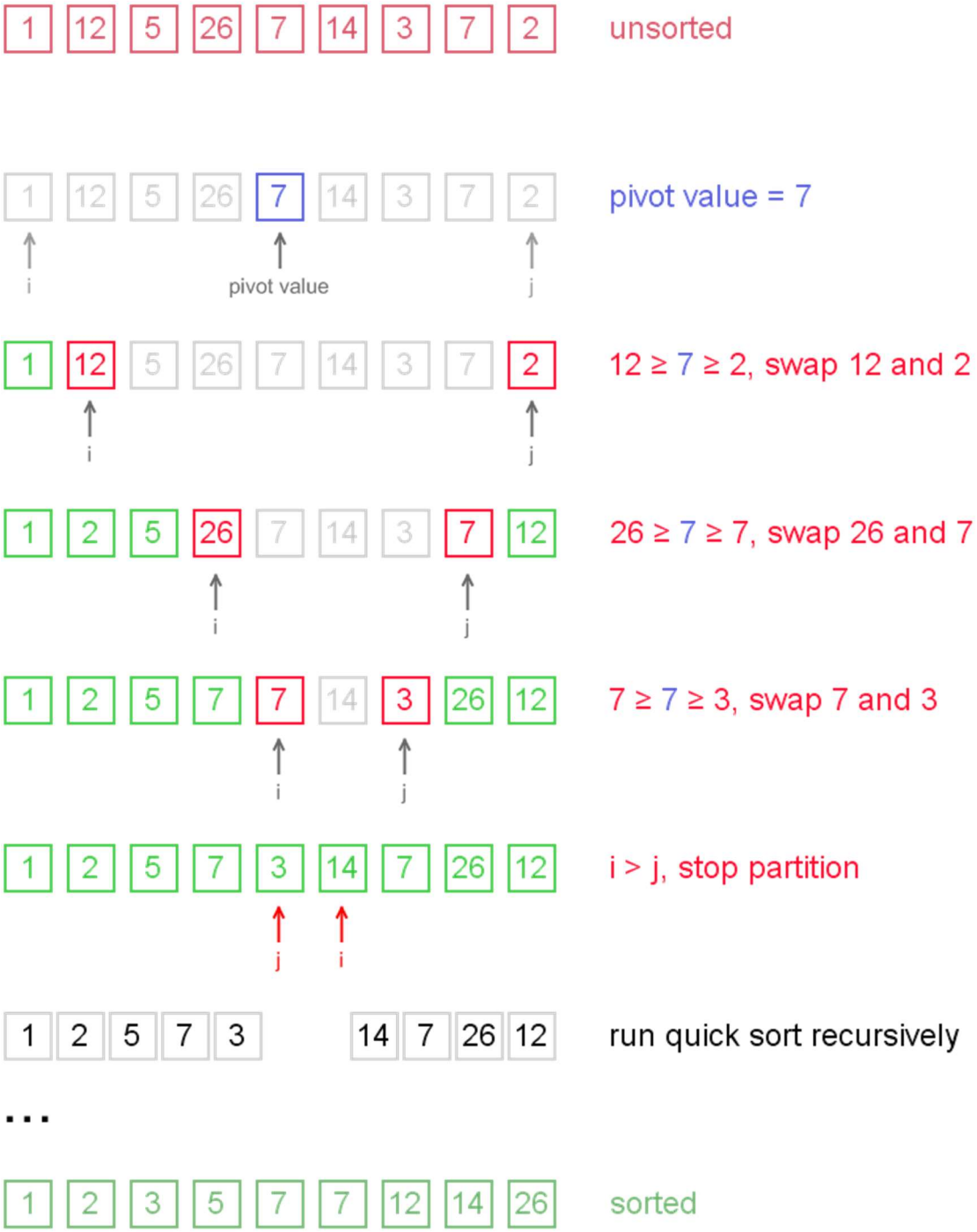
Partition algorithm in detail

There are two indices **i** and **j** and at the very beginning of the partition algorithm **i** points to the first element in the array and **j** points to the last one. Then algorithm moves **i** forward, until an element with

value greater or equal to the pivot is found. Index **j** is moved backward, until an element with value lesser or equal to the pivot is found. If **i ≤ j** then they are swapped and i steps to the next position (**i + 1**), j steps to the previous one **(j - 1)**. Algorithm stops, when **i** becomes greater than **j**.

After partition, all values before **i-th** element are less or equal than the pivot and all values after **j-th** element are greater or equal to the pivot.

Example. Sort {1, 12, 5, 26, 7, 14, 3, 7, 2} using quicksort.



Notice, that we show here only the first recursion step, in order not to make example too long. But, in fact, {1, 2, 5, 7, 3} and {14, 7, 26, 12} are sorted then recursively.

Why does it work?

On the partition step algorithm divides the array into two parts and every element **a** from the left part is less or equal than every element **b** from the right part. Also **a** and **b** satisfy **a ≤ pivot ≤ b** inequality. After completion of the recursion calls both of the parts become sorted and, taking into account arguments stated above, the whole array is sorted.