**Quicksort in Java**

**Quicksort** is an efficient in-place sorting algorithm that is often faster in practice in comparing to other sorting algorithms. It has the time complexity O(n \log n)*O*(*n*log*n*) in the average case and O(n^2)*O*(*n*2) in the worst case, but fortunately, it often works like in the average. We will consider some bad cases for the algorithm later.

**Quicksort**can be implemented as a recursive or iterative algorithm. We will consider only the recursive version here.

Below is a recursive version of quicksort in Java:

public static void quickSort(int[] array, int left, int right) {  
    if (left < right) {  
        int pivotIndex = partition(array, left, right); // the pivot is already on its place  
        quickSort(array, left, pivotIndex - 1);  // sort the left subarray  
        quickSort(array, pivotIndex + 1, right); // sort the right subarray   
    }  
}

The quickSort method takes an array of ints and a range of indexes (left, right) to sort the array between them (inclusive).

The partition method reorders the array and returns the index of the pivot to divide the array into two parts. The method swap rearranges two elements in the array. Here is an implementation of these methods:

private static int partition(int[] array, int left, int right) {  
    int pivot = array[right];  // choose the rightmost element as the pivot  
    int partitionIndex = left; // the first element greater than the pivot  
   
    /\* move large values into the right side of the array \*/  
    for (int i = left; i < right; i++) {  
        if (array[i] <= pivot) { // may be used '<' as well  
            swap(array, i, partitionIndex);  
            partitionIndex++;  
        }  
    }  
   
    swap(array, partitionIndex, right); // put the pivot on a suitable position  
   
    return partitionIndex;  
}  
   
private static void swap(int[] array, int i, int j) {  
    int temp = array[i];  
    array[i] = array[j];  
    array[j] = temp;  
}

This implementation of the partition method is known as **Lomuto partition scheme**. It chooses the rightmost element as a pivot.

Here are some tests:

int[] array1 = { 17, 25, 11, 16, 10, 13, 22, 14 };  
quickSort(array1, 0, array1.length - 1); // { 10, 11, 13, 14, 16, 17, 22, 25 }  
          
int[] array2 = { 19, 18, 17, 17, 16, 15 };  
quickSort(array2, 0, array2.length - 1); // { 15, 16, 17, 17, 18, 19 }

Although the algorithm is simple enough, the recursive implementation is not the best solution if we are going to sort really large arrays. In the worst cases, the depth of the recursion can be quite large and it will throw StackOverflowError. You may try to implement this algorithm using loops rather than recursion.