```
    public static void quickSort(int[] array) {
        // Sort the whole array.
        quickSort(array, 0, array.length - 1);
    }

    /**
     * Sort a part of the array using the quicksort algorithm.
     *
     * @param array The array to be sorted
     * @param first The index of the low bound
     * @param last  The index of the high bound
     * @post The part of array from first through last is sorted.
     */

    private static void quickSort(int[] array, int first, int last) {
        if (first < last) {  // There is data to be sorted.
            // Partition the array.
            int pivIndex = partition(array, first, last);
            // Sort the left half.
            quickSort(array, first, pivIndex - 1);
            // Sort the right half.
            quickSort(array, pivIndex + 1, last);
        }
    }

    /**
     * Partition the array so that values from first to pivIndex
     * are less than or equal to the pivot value, and values from
     * pivIndex to last are greater than the pivot value.
     *
     * @param array The array to be partitioned
     * @param first The index of the low bound
     * @param last  The index of the high bound
     * @return The location of the pivot value
     */
    private static int partition(int[] array, int first, int last) {

        // Select the first item as the pivot value.
        int pivot = array[first];

        int up = first;
        int down = last;

        do {
            // Goal of this loop: All items in array[first . . . up - 1] <= pivot
            //                    All items in array[down + 1 . . . last] > pivot
            while ((up < last) && (pivot >= array[up])) {
                up++;
            }
            // After above loop ends: up equals last or array[up] > pivot.

            while (pivot < array[down]) {
                down--;
            }
            // After above loop ends: down equals first or array[down] <= pivot.

            if (up < down) {  // if up is to the left of down.
                // Exchange array[up] and array[down].
                swap(array, up, down);
            }
        } while (up < down); // Repeat while up is left of down.

        // Exchange array[first] and array[down] thus putting the
        // pivot value where it belongs.
        swap(array, first, down);

        // Return the index of the pivot value.
        return down;
    }
```