QuickSort

It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Always pick the first element as a pivot.
Always pick the last element as a pivot (implemented below)
Pick a random element as a pivot.
Pick median as the pivot.
The key process in quickSort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Partition Algorithm:

There can be many ways to do partition, following pseudo-code adopts the method given in the CLRS book. The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise, we ignore the current element.

Pseudo Code for recursive QuickSort function:

```
/* low  –> Starting index,  high  –> Ending index */

quickSort(arr[], low, high) {

   if (low < high) {
```

```
        /* pi is partitioning index, arr[pi] is now at right place */

        pi = partition(arr, low, high);

        quickSort(arr, low, pi – 1);  // Before pi

        quickSort(arr, pi + 1, high); // After pi

    }

}
```

Pseudo code for partition()

```
/* This function takes last element as pivot, places the pivot element
at its correct position in sorted array, and places all smaller (smaller
than pivot) to left of pivot and all greater elements to right of pivot
*/

partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low – 1)  // Index of smaller element and indicates the
    // right position of pivot found so far

    for (j = low; j <= high- 1; j++){

        // If current element is smaller than the pivot
        if (arr[j] < pivot){
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
```

```
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

C# example:

```
using System;

class GFG {

    // A utility function to swap two elements
    static void swap(int[] arr, int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    /* This function takes last element as pivot, places
         the pivot element at its correct position in sorted
         array, and places all smaller (smaller than pivot)
         to left of pivot and all greater elements to right
         of pivot */
    static int partition(int[] arr, int low, int high)
    {

        // pivot
        int pivot = arr[high];

        // Index of smaller element and
        // indicates the right position
        // of pivot found so far
        int i = (low - 1);

        for (int j = low; j <= high - 1; j++) {

            // If current element is smaller
            // than the pivot
            if (arr[j] < pivot) {

                // Increment index of
                // smaller element
                i++;
                swap(arr, i, j);
```

```
            }
        }
        swap(arr, i + 1, high);
        return (i + 1);
    }

    /* The main function that implements QuickSort
               arr[] --> Array to be sorted,
               low --> Starting index,
               high --> Ending index
       */
    static void quickSort(int[] arr, int low, int high)
    {
        if (low < high) {

            // pi is partitioning index, arr[p]
            // is now at right place
            int pi = partition(arr, low, high);

            // Separately sort elements before
            // partition and after partition
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    // Function to print an array
    static void printArray(int[] arr, int size)
    {
        for (int i = 0; i < size; i++)
            Console.Write(arr[i] + " ");

        Console.WriteLine();
    }

    // Driver Code
    public static void Main()
    {
        int[] arr = { 10, 7, 8, 9, 1, 5 };
        int n = arr.Length;

        quickSort(arr, 0, n - 1);
        Console.Write("Sorted array: ");
        printArray(arr, n);
    }
}
```