**QuickSort**

**QuickSort Algorithm**

|  |
| --- |
| **1.** Choose any element of the array to be the **pivot**.  **2. Partition**: All elements less than the pivot - **left partition**. All elements greater than the pivot - **right partition**.  **3.** Use recursion to sort both partitions. Join the first sorted partition, the pivot, and the second sorted partition. |

**Efficiency of QuickSort**

|  |
| --- |
| The best pivot creates partitions of equal length (or lengths differing by 1). The worst pivot creates an empty partition (for example, if the pivot is the first or last element of a sorted array). The runtime of Quicksort ranges from [**O**](http://rosettacode.org/wiki/O)**(n log n)** with the best pivots, to [**O**](http://rosettacode.org/wiki/O)**(n2)** with the worst pivots, where *n* is the number of elements in the array. |

**The QuickSort Method**

|  |
| --- |
| **public static void quickSort**(**int arr**[], **int** left, **int** right)  {  int index = **partition**(arr, left, right);    **if** (left < index - 1)  **quickSort**(arr, left, index - 1);  **if** (index < right)  **quickSort**(arr, index, right);  } |

**The partition Method**

|  |
| --- |
| **public static int partition**(**int** arr[], **int** left**, int** right)  {  int i = left, j = right;  int pivot = arr[(left + right) / 2];  **while** (i <= j)  {  **while** (arr[i] < pivot)  i++;  **while** (arr[j] > pivot)  j--;  **if** (i <= j)  {  int tmp = arr[i];  arr[i] = arr[j];  arr[j] = tmp;  i++;  j--;  }  }  **return** i;  } |