快速排序 Quicksort

快速排序的平均性能相当好，它的期望运行时间为 O(nlgn)，且隐含的常数因子很小。但在最坏情况下，它的运行时间会来到 O(n^2)。与堆排序 Heapsort 相同，快速排序同样是一种原地 in place 排序。

分治法

很多算法在结构上是递归的：为了解决一个给定的问题，算法要一次或多次地递归调用其自身来解决相关的子问题。分治法在每一次递归上都有三个步骤：

1.分解（Divide）: 将原问题分解成一系列子问题.

2. 解决（Conquer）: 递归地解各个子问题。若子问题足够小，则直接求解.

3. 合并 （Combine）: 将子问题的结果合并成原问题的解。

快速排序是基于分治模式的。

1.分解（Divide）: Array A 按 PARTITION 过程得到的 index 分割成两个（可能空）的子数组。

2. 解决（Conquer）: 递归地调用快速排序，分别对子数组进行排序。

3. 合并 （Combine）: 由于快速排序是 in-place 排序，所以已不需要合并操作。

快速排序的具体过程

```
QUICKSORT(A, p, r)
1  if p<r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q−1)
4          QUICKSORT(A, q+1, r)
```

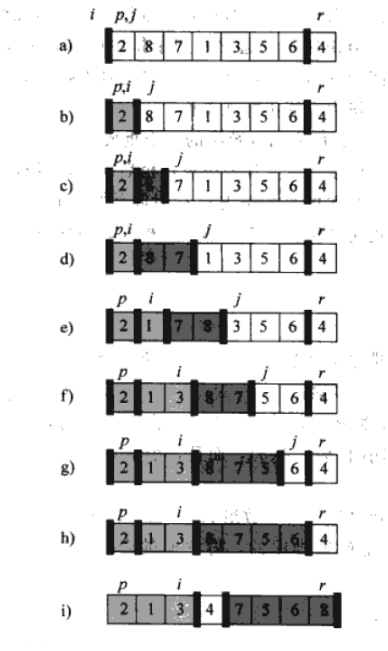
先看总体过程，这是一个很容易理解的伪代码，第 1 行代码确保递归在子数组无法再次分割后停止。第 2 行是 PARTITION 过程，该过程会返回选取的主元 pivot element 经过排序后所在位置的 index。（该位置左侧所有元素都小于等于主元，右侧反之）以此为新的分割点，将 array 分割成 2 个子数组。这就是完整的过程，递归时 array 中元素的位置不断改变，以主元 pivot element 为界限，主元左侧的元素全部小于等于主元，右侧则全部大于。

下面我们不妨来看看 PARTITION 到底是如何实现的。

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p−1
3  for j ← p to r−1
4       do if A[j]≤x
5             then i ← i+1
6                  exchange A[i]↔A[j]
7  exchange A[i+1]↔A[r]
8  return i+1
```

代码第 1 行和第 2 行非常便于理解，其中 x 就是我们要用到的主元 pivot element,这里是将数组末尾元素作为主元。变量 i 的作用是帮助换位，这会在后面展现，i 的初始值是 array 的开始位置前一位。

接下来是遍历整个数组的过程，从 array 头部开始直到尾部，每个元素都会与 pivot element 进行比较，如果该元素小于或等于 pivot element，则 i+=1，并让索引变量 j 所在位置与 i+=1 位置元素置换。这么做的结果是，让所有小于或等于 pivot element 的元素都来到左侧。这里用下图方便理解。



首先可以看到，i 的起始位置是 array 开始位置-1，这是因为 array 的第一个元素可以是小于等于 pivot element 的，就像左图显示的那样。I 会与 j 进行位置交换，i+=1 的结果就是，i 与 j 同处同一个格子当中了。所以它们实际上没有发生具体交换。但如果 array 的第一个元素大于 pivot element，那么 j 将会向右移动，之后发生的置换就会确实发生了。

过程结束后，我们就能得到一个这样的 array：以 pivot element 为分割点，左侧所有元素全部小于等于 pivot element，右侧全部大于。

PARTITION 过程所做的，就是在一次次递归中，将指定范围的 array 进行一次次排序。所有元素的位置在一次次的置换后，最终得到了排序后的结果。

快速排序的最坏情况

快速排序的最坏情况发生在分割点，也就是 pivot element 的选择每次都是最糟糕的情况。在这种情况下，pivot element 左侧只有 1 个元素，而右侧则存在 length -1 个元素。这时，Quicksort 的运行时间为 O(n^2)。

平衡的划分

快速排序的最佳情况运行时间为 O(nlgn)。而它的平均情况运行时间与最佳情况非常接近。其平均情况运行时间为 O(nlgn)。具体求解过程请参考 算法导论 第 89 页。