## 1、快速排序的基本思想：

快速排序使用分治的思想，通过一趟排序将待排序列分割成两部分，其中一部分记录的关键字均比另一部分记录的关键字小。之后分别对这两部分记录继续进行排序，以达到整个序列有序的目的。

## 2、快速排序的三个步骤：

(1)选择基准：在待排序列中，按照某种方式挑出一个元素，作为 “基准”（pivot）

(2)分割操作：以该基准在序列中的实际位置，把序列分成两个子序列。此时，在基准左边的元素都比该基准小，在基准右边的元素都比基准大

(3)递归地对两个序列进行快速排序，直到序列为空或者只有一个元素。

## 3、选择基准的方式：

**对于分治算法，当每次划分时，算法若都能分成两个等长的子序列时，那么分治算法效率会达到最大。**也就是说，基准的选择是很重要的。选择基准的方式决定了两个分割后两个子序列的长度，进而对整个算法的效率产生决定性影响。

**最理想的方法是，选择的基准恰好能把待排序序列分成两个等长的子序列**

## 4、选择基准的三种方法

### 1.固定位置

思想：取序列的第一个或最后一个元素作为基准



### 2.随机选取基准

引入的原因：在待排序列是部分有序时，固定选取枢轴使快排效率底下，要缓解这种情况，就引入了随机选取枢轴

思想：取待排序列中任意一个元素作为基准



测试数据分析：:这是一种相对安全的策略。由于枢轴的位置是随机的，那么产生的分割也不会总是会出现劣质的分割。在整个数组数字全相等时，仍然是最坏情况，时间复杂度是O(n^2）。**实际上，随机化快速排序得到理论最坏情况的可能性仅为1/(2^n）**。所以随机化快速排序可以对于绝大多数输入数据达到O(nlogn）的期望时间复杂度。一位前辈做出了一个精辟的总结：“随机化快速排序可以满足一个人一辈子的人品需求。”

### 三数取中

引入的原因：虽然随机选取枢轴时，减少出现不好分割的几率，但是还是最坏情况下还是O(n^2），要缓解这种情况，就引入了三数取中选取枢轴

分析：最佳的划分是将待排序的序列分成等长的子序列，最佳的状态我们可以使用序列的中间的值，也就是第N/2个数。可是，这很难算出来，并且会明显减慢快速排序的速度。这样的中值的估计可以通过随机选取三个元素并用它们的中值作为枢纽元而得到。事实上，随机性并没有多大的帮助，**因此一般的做法是使用左端、右端和中心位置上的三个元素的中值作为枢纽元。**显然使用三数中值分割法消除了预排序输入的不好情形，并且减少快排大约14%的比较次数

举例：待排序序列为：8 1 4 9 6 3 5 2 7 0

左边为：8，右边为0，中间为6.

我们这里取三个数排序后，中间那个数作为枢轴，则枢轴为6



## 5、四种优化方法

### 1. 当待排序序列的长度分割到一定大小后，使用插入排序

原因：**对于很小和部分有序的数组，快排不如插排好**。当待排序序列的长度分割到一定大小后，继续分割的效率比插入排序要差，此时可以使用插排而不是快排

截止范围：**待排序序列长度N = 10**，虽然在5~20之间任一截止范围都有可能产生类似的结果，这种做法也避免了一些有害的退化情形。摘自《数据结构与算法分析》Mark Allen Weiness 著

### 2. 在一次分割结束后，可以把与Key相等的元素聚在一起，继续下次分割时，不用再对与key相等元素分割

具体过程：在处理过程中，会有两个步骤

第一步，在划分过程中，把与key相等元素放入数组的两端

第二步，划分结束后，把与key相等的元素移到枢轴周围



### 3. 优化递归操作

快排函数在函数尾部有两次递归操作，我们可以对其使用尾递归优化

优点：如果待排序的序列划分极端不平衡，递归的深度将趋近于n，而栈的大小是很有限的，每次递归调用都会耗费一定的栈空间，函数的参数越多，每次递归耗费的空间也越多。优化后，可以缩减堆栈深度，由原来的O(n)缩减为O(logn)，将会提高性能。

1. **void** QSort(**int** arr[],**int** low,**int** high)
2. {
3. **int** pivotPos = -1;
4. **if** (high - low + 1 < 10)
5. {
6. InsertSort(arr,low,high);
7. **return**;
8. }
9. **while**(low < high)
10. {
11. pivotPos = Partition(arr,low,high);
12. QSort(arr,low,pivot-1);
13. low = pivot + 1;
14. }
15. }

注意：在第一次递归后，low就没用了，此时第二次递归可以使用循环代替



### 4.使用并行或多线程处理子序列（暂无

## 总结：

概括：这里效率最好的快排组合 是：三数取中+插排+聚集相等元素,它和STL中的Sort函数效率差不多

注意：由于测试数据不稳定，数据也仅仅反应大概的情况。如果时间上没有成倍的增加或减少，仅仅有小额变化的话，我们可以看成时间差不多。