**快速排序的改进**

只对长度大于k的子序列递归调用快速排序,让原序列基本有序，然后再对整个基本有序序列用插入排序算法排序。实践证明，改进后的算法时间复杂度有所降低，且当k取值为 8 左右时,改进算法的性能最佳。

**选择基准元的方式**

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。

方法1 固定基准元

如果输入序列是随机的，处理时间是可以接受的。如果数组已经有序时，此时的分割就是一个非常不好的分割。

方法2 随机基准元

这是一种相对安全的策略。由于基准元的位置是随机的，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。

方法3 三数取中

引入的原因：虽然随机选取基准时，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基准。

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基准元而得到。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基准元。

**快速排序的时间复杂度/空间复杂度**

**时间复杂度**

快速排序涉及到递归调用，所以该算法的时间复杂度还需要从递归算法的复杂度开始说起；

递归算法的时间复杂度公式：T[n] = aT[n/b] + f(n) ；对于递归算法的时间复杂度这里就不展开来说了；

**最优情况下时间复杂度**

快速排序最优的情况就是每一次取到的元素都刚好平分整个数组(很显然我上面的不是)；

此时的时间复杂度公式则为：T[n] = 2T[n/2] + f(n)；T[n/2]为平分后的子数组的时间复杂度，f[n] 为平分这个数组时所花的时间；

下面来推算下，在最优的情况下快速排序时间复杂度的计算(用迭代法)：

T[n] = 2T[n/2] + n ----------------第一次递归

令：n = n/2 = 2 { 2 T[n/4] + (n/2) } + n ----------------第二次递归

= 2^2 T[ n/ (2^2) ] + 2n

令：n = n/(2^2) = 2^2 { 2 T[n/ (2^3) ] + n/(2^2)} + 2n ----------------第三次递归

= 2^3 T[ n/ (2^3) ] + 3n

......................................................................................

令：n = n/( 2^(m-1) ) = 2^m T[1] + mn ----------------第m次递归(m次后结束)

当最后平分的不能再平分时，也就是说把公式一直往下跌倒，到最后得到T[1]时，说明这个公式已经迭代完了（T[1]是常量了）。

得到：T[n/ (2^m) ] = T[1] ===>> n = 2^m ====>> m = logn；

T[n] = 2^m T[1] + mn ；其中m = logn;

T[n] = 2^(logn) T[1] + nlogn = n T[1] + nlogn = n + nlogn ；其中n为元素个数

又因为当n >= 2时：nlogn >= n (也就是logn > 1)，所以取后面的 nlogn；

综上所述：快速排序最优的情况下时间复杂度为：O( nlogn )

**最差情况下时间复杂度**

最差的情况就是每一次取到的元素就是数组中最小/最大的，这种情况其实就是冒泡排序了(每一次都排好一个元素的顺序)

这种情况时间复杂度就好计算了，就是冒泡排序的时间复杂度：T[n] = n \* (n-1) = n^2 + n;

综上所述：快速排序最差的情况下时间复杂度为：O( n^2 )

**平均时间复杂度**

快速排序的平均时间复杂度也是：O(nlogn)

**空间复杂度**

其实这个空间复杂度不太好计算，因为有的人使用的是非就地排序，那样就不好计算了（因为有的人用到了辅助数组，所以这就要计算到你的元素个数了）；我就分析下就地快速排序的空间复杂度吧；

首先就地快速排序使用的空间是O(1)的，也就是个常数级；而真正消耗空间的就是递归调用了，因为每次递归就要保持一些数据；

最优的情况下空间复杂度为：O(logn) ；每一次都平分数组的情况

最差的情况下空间复杂度为：O( n ) ；退化为冒