**基于数值比较的排序问题的计算复杂性下界**

对[排序算法](https://so.csdn.net/so/search?q=%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95&spm=1001.2101.3001.7020)计算时间的分析可以遵循若干种不同的准则，通常以排序过程所需要的算法步数作为度量，有时也以排序过程中所作的键比较次数作为度量。特别是当作一次键比较需要较长时间，例如，当键是较长的字符串时，常以键比较次数作为排序算法计算时间复杂性的度量。当排序时需要移动记录，且记录都很大时，还应该考虑记录的移动次数。究竟采用哪种度量方法比较合适要根据具体情况而定。在下面的讨论中我们主要考虑用比较的次数作为复杂性的度量。

为了对有n个元素的线性表进行排序，至少必须扫描线性表一遍以获取这n个元素的信息，因此排序问题的[计算复杂性下界为Ω(n)](http://writeblog.csdn.net/complexity/chapter3.htm)。

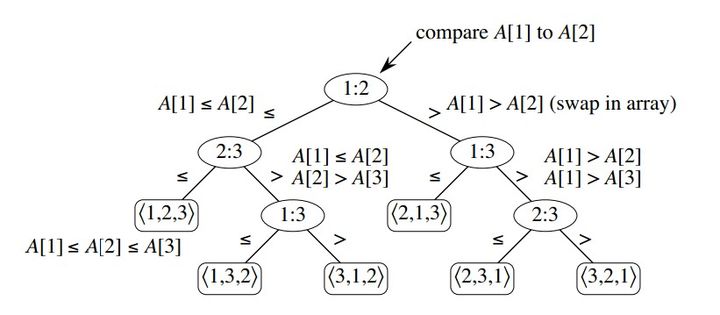
如果我们对输入的数据不做任何要求，我们所能获得的唯一信息就是各个元素的具体的值，我们仅能通过比较来确定输入序列<a1,a2,..,an>的元素间次序。即给定两个元素ai和aj，通过测试ai<aj，ai≤aj,ai=aj,ai≥aj,ai>aj中的哪一个成立来确定ai和aj间的相对次序。这样的排序算法称为***比较排序算法***。

下面我们讨论一下比较排序算法在最坏情况下**至少**需要多少次比较，即比较排序算法的最坏情况复杂性下界。

我们假设每次比较只测试ai≤aj，如果ai≤aj成立则ai排在aj前面，否则ai排在aj后面。任何一个比较排序算法可以描述为一串比较序列:

 (ai,aj),(ak,al),..,(am,an),...

表示我们首先比较(ai,aj)，然后比较(ak,al)，...，比较(am,an)，...，直到我们获取了足够的信息可以确定所有元素的顺序。显而易见，如果我们对所有的元素两两进行一次比较的话(总共比较了Cn2次)，就一定可以确定所有元素的顺序。但是，如果我们运气足够好的话，我们可能不必对所有元素两两进行一次比较。比如说对于有三个元素a1,a2,a3的线性表进行排序，如果我们先比较a1和a2，得到a1≤a2；然后比较a2和a3，得到a2≤a3；则不必比较a1和a3，因为根据偏序集的传递性，必有a1≤a3；但是如果a2≥a3，我们还必须比较a1和a3才能确定a1和a3的相对位置。如果我们适当的安排比较的次序的话，也可以减少比较的次数。这样我们可以用一棵二叉树表示比较的顺序，如下图所示：



该树的每一个非叶节点表示一次比较，每一根树枝表示一种比较结果，每一个叶节点表示一种排列顺序。这样的一棵二叉树叫做***决策树***，它用树枝表示了每次决策做出的选择。如此我们可以将任何一个比较排序算法用一棵[决策树](https://so.csdn.net/so/search?q=%E5%86%B3%E7%AD%96%E6%A0%91&spm=1001.2101.3001.7020)来表示。

请注意上图只表明了对三个元素的一种比较算法，这种比较算法依次比较(a1,a2)(a2,a3)(a1,a3)，一旦中间某步得到足够的信息就可以停止比较，但是当算法执行完后（三次比较后），一定可以确定三个元素间的次序。因此我们有理由将算法在最坏情况下的比较次数作为算法复杂性的度量，对于本例该算法在最坏情况下要进行C32=3次比较。

显然，一棵决策树中最高叶节点的高度就是该决策树对应的算法在最坏情况下所需的比较次数，而决策树中最低叶节点的高度就是该决策树对应的算法在最好情况下所需的比较次数。

我们的问题就变为：对于任意一棵决策树（任意一种比较排序算法），它的最高的树叶的高度是多少？这个高度就对应于比较排序算法所需的最多比较次数（在运气最坏的情况下）；换句话说，对于任何一个输入，该算法至少需要比较多少次就可以对元素进行排序。

我们发现，决策树的每个叶节点对应一个n个元素的排列，其中可能有重复的；但是由于决策树表明了所有可能遇到的情况，因而n个元素的所有排列都在决策树中出现过。n个元素共有n!种排列，即决策树的叶节点数目**至少**为n!。又因为一棵高度为h的二叉树（指二叉树的最高树叶高度为h）的叶节点数目最多为2h个（这时正好是满二叉树，即每个非叶节点都有两个子节点），因此n!≤2h，得到h≥log(n!)，其中log以2为底。根据Stirling公式有n!>(n/e)n，于是h>nlogn-nloge，即h=Ω(nlogn)。

这样我们就证明了对于任意一种利用比较来确定元素间相对位置的排序算法，其最坏情况下复杂性为Ω(nlogn)。

在下文中我们将讨论几种比较排序算法，其中快速排序在平均情况下复杂性为*O*(nlogn)，最坏情况下复杂性为*O*(n2)；堆排序和合并排序在最坏情况下复杂性为*O*(nlogn)，因此堆排序和合并排序是渐进最优的比较排序算法。