二叉树排序(堆排序Heap Sort)算法

# 排序思想：

## 堆排序也是选择排序的一种，根据堆的特性，每次把最大或最小值（本次以最大值为例）拿出来，按序排列；

## 堆排序是对普通选择排序的一种优化：

如果是一个**稳定堆**，每次在选择最大值时，只用沿着二叉树其中一个分叉去交换即可，其他分叉符合堆的特性（因是排好的稳定堆），可以看作是稳定的，不用重排交换，省去了绝大多数的比较交换步骤，数组的数越多，分支越多，该算法的优势就越明显；

## 步骤

第一步，将数组初始化为**稳定堆**;

稳定堆的特性：**二叉树之父节点总比其左右子节点大！**初始化稳定堆有很多方法，可以从堆顶向堆底方向初始化，也可以从堆底向堆顶方向排列初始化，也可以通过小三角递归调等完成初始化，本例为理解方便，选用从堆底向堆顶方向初始化，即，每次从小三角里找到最大值放在父节点，从最后一个小三角向前循环，第一遍，找到所有数的最大值，第二遍循环找到次最大值放在第二层节点上，依次类推，完成稳定堆得初始化；

第二步，初始化完稳定堆之后，将选出的最大值与最后一个数字交换放在数组的最后固定下来（以后循环不再用到此数字）。

此时除了顶部三角不稳定，下面都是稳定的，根据稳定堆的特性（父节点总是大于其左右子节点），从顶部往底部寻找交换，只沿着变动的那个分叉交换下去（只用单线一次循环即可），其他分叉不用动，交换完毕后，**次最大数又被移到顶部，**此时的堆仍然是一个稳定堆，再将顶部的最大值交换的数组的后面固定下来，重复这个步骤，依次类推，即可完成。

排序趟数：

1，初始化堆所用趟数：不确定，最大趟数是二叉树的层数，最小一趟；

2，初始化堆后所用趟数：length-2次；

排序原理：

稳定堆特性，二叉树排序，具体见上述排序思想

# 复杂度分析

## 空间复杂度

堆排和快排都是**原址排序算法**，即除了输入数组外，仅需要**常数个额外的存储空间**。

空间复杂度都是 O(1) 。

## 时间复杂度

堆排的时间复杂度跟算法相关，比较稳定。因为不管数据如何，堆排都要先构造最大堆，然后一个一个的把堆顶元素移到堆尾，之后再维持最大堆的性质。

而快排的时间复杂度受数据的影响。但一般情况下都不会差到哪里去。

不管是以上那种排序算法，都涉及到交换数组中两个数据的位置。

关于快排的不稳定我们常说快排不稳定，跟数据有关，最坏运行时间是 ，那什么情况下才会发生？了解快排的同学应该知道，快排其实是把数组按照某个基准数据 进行分割，分为大于 的数据（放右边）和小于 的数据（放左边），接着把 放中间；然后对左右分别递归此过程。那么只有当每次分割数据时，所有数据都小于 ，或者都大于 时，才会产生 的时间复杂度。分割操作的时间复杂度是 所以时间复杂度为： 发生这种情况的概率是： ，约等于0.从统计学上来说，快排产生的划分 80% 以上都比 9:1 更平衡，另外的 20% 的划分比 9:1 更不平衡。就算所有的划分都是 9:1 ，快排的时间复杂度依然是 .

**时间对比**

快排是非稳定时间的，堆排序是稳定时间的，堆排序的排序时间与数据无关，快排与数据有关，（堆排序对相同长度的数组比较次数是固定的，这点估计在教材某个角落没注意吧？）意味着同样大小的数据，可能已经排好序的，堆排序仍然需要花上同样的时间去排序，大部分情况下快排比堆排序要快 所以由统计学意义来讲，快排远比堆要优越

# 堆排序的缺点

每次移出堆顶最大元素后，都需要从顶部维护最大堆性质导致了过多的数据交换操作。

# 对 heap sort “make poor use of cache memory" 做解释

在**《算法导论》**上看基数排序是也看到了这么一句话：**快速排序通常比基数排序更有效地使用硬件的缓存**。仔细想一想，快排除了数据交换操作少之外，它的循环比较操作是拿 与 进行比较，其中i是循环的下标，k是选中的基准点，一般选array首部或尾部。而堆排序比较的是array[i]、array[left(i)]与array[right(i)]。在硬件基础上，快排的基准点k每次循环时都要访问，更容易缓存命中，但堆排的中的数据相比来说没那么容易。所以说 heap sort “make poor use of cache memory"。









