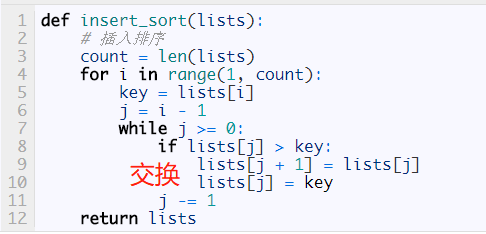
**排序方法**

* **稳定**：如果a原本在b前面，而a=b，排序之后a仍然在b的前面；
* **不稳定**：如果a原本在b的前面，而a=b，排序之后a可能会出现在b的后面；
* **内排序**：所有排序操作都在内存中完成；
* **外排序**：由于数据太大，因此把数据放在磁盘中，而排序通过磁盘和内存的数据传输才能进行；
* **时间复杂度：** 一个算法执行所耗费的时间。
* **空间复杂度**：运行完一个程序所需内存的大小。

****

1. **插入排序**

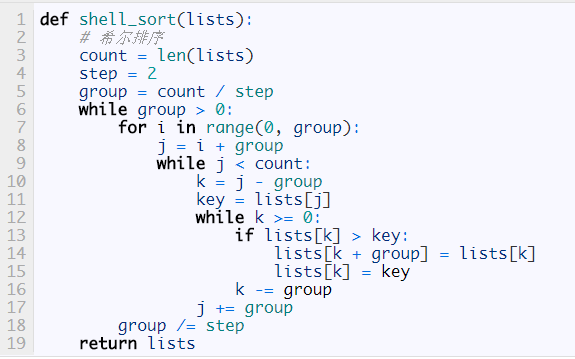
它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。算法适用于少量数据的排序，时间复杂度为O(n^2)。在对几乎排好的数据排序时，可达到线性复杂度，但一般来说是低效的。



1. **希尔排序**

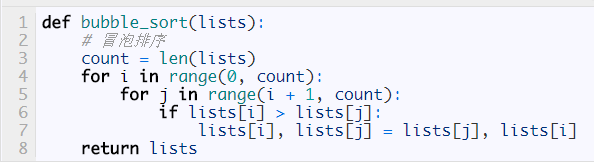
希尔排序，也称递减增量排序算法，是插入排序的一种高速而稳定的改进版本。希尔排序是非稳定排序算法。希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序；随着增量逐渐减少，每组包含的关键词越来越多，当增量减至1时，整个文件恰被分成一组，算法便终止。

（即先组间一一比较并交换，当组间距为1时，便是相邻两个比较）



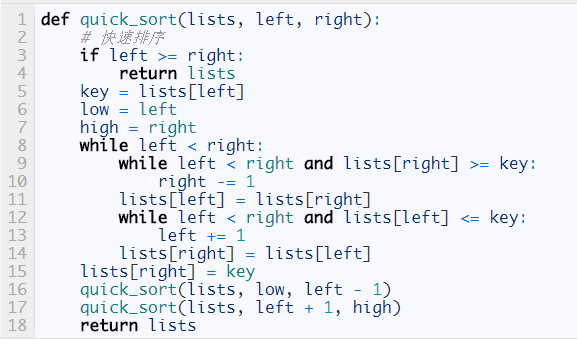
1. **冒泡排序**

它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。



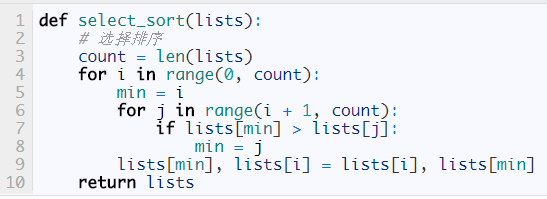
1. **快速排序**

通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。



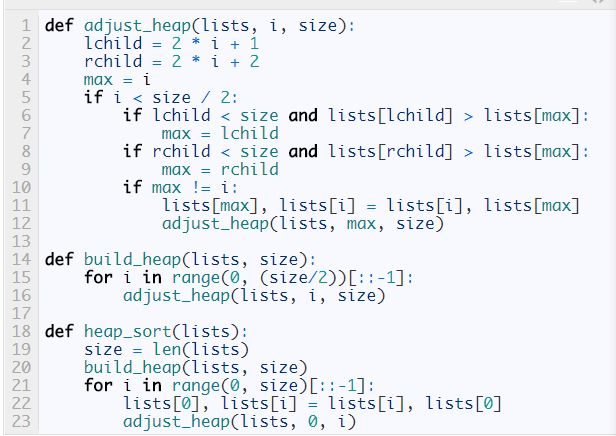
1. **直接选择排序**

第1趟，在待排序记录r1 ~ r[n]中选出最小的记录，将它与r1交换；第2趟，在待排序记录r2 ~ r[n]中选出最小的记录，将它与r2交换；以此类推，第i趟在待排序记录r[i] ~ r[n]中选出最小的记录，将它与r[i]交换，使有序序列不断增长直到全部排序完毕。



1. **堆排序**

堆排序(Heapsort)是指利用堆积树（堆）这种数据结构所设计的一种排序算法，它是选择排序的一种。可以利用数组的特点快速定位指定索引的元素。堆分为大根堆和小根堆，是完全二叉树。大根堆的要求是每个节点的值都不大于其父节点的值，即A[PARENT[i]] >= A[i]。在数组的非降序排序中，需要使用的就是大根堆，因为根据大根堆的要求可知，最大的值一定在堆顶。



1. **归并排序**

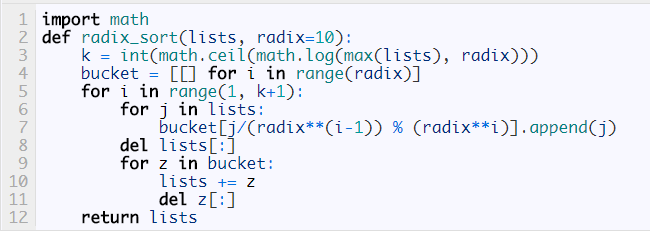
归并排序是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

归并过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元。归并排序的算法我们通常用递归实现，先把待排序区间[s,t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s,t]。



1. **基数排序**

基数排序（radix sort）属于“分配式排序”（distribution sort），又称“桶子法”（bucket sort）或bin sort，顾名思义，它是透过键值的部份资讯，将要排序的元素分配至某些“桶”中，藉以达到排序的作用，基数排序法是属于稳定性的排序，其时间复杂度为O (nlog(r)m)，其中r为所采取的基数，而m为堆数，在某些时候，基数排序法的效率高于其它的稳定性排序法。

****