**排序算法的Python实现**

**插入排序**

直接插入排序的核心思想就是：将数组中的所有元素依次跟前面已经排好的元素相比较，如果选择的元素比已排序的元素小，则交换，直到全部元素都比较过。

因此，从上面的描述中我们可以发现，直接插入排序可以用两个循环完成：

第一层循环：遍历待比较的所有数组元素

第二层循环：将本轮选择的元素(selected)与已经排好序的元素(ordered)相比较。如果：selected < ordered，那么将二者交换

**def** insert\_sort(lists):  
 *# 插入排序* count = len(lists)  
 **for** i **in** range(1, count):  
 key = lists[i]  
 j = i - 1  
 **while** j >= 0:  
 **if** lists[j] > key:  
 lists[j + 1] = lists[j]  
 lists[j] = key  
 j -= 1  
 **return** lists

**简单选择排序**

简单选择排序的基本思想：比较+交换。

从待排序序列中，找到关键字最小的元素；

如果最小元素不是待排序序列的第一个元素，将其和第一个元素互换；

从余下的 N - 1 个元素中，找出关键字最小的元素，重复(1)、(2)步，直到排序结束。

因此我们可以发现，简单选择排序也是通过两层循环实现。

第一层循环：依次遍历序列当中的每一个元素

第二层循环：将遍历得到的当前元素依次与余下的元素进行比较，符合最小元素的条件，则交换。

def select\_sort(lists):  
 # 选择排序  
 count = len(lists)  
 for i in range(0, count):  
 min = i  
 for j in range(i + 1, count):  
 if lists[min] > lists[j]:  
 min = j  
 lists[min], lists[i] = lists[i], lists[min]  
 return lists

**堆排序**

基本思想：

堆排序可以按照以下步骤来完成：

首先将序列构建称为大顶堆（这样满足了大顶堆那条性质：位于根节点的元素一定是当前序列的最大值）；

构建大顶堆

1.取出当前大顶堆的根节点，将其与序列末尾元素进行交换（此时：序列末尾的元素为已排序的最大值；由于交换了元素，当前位于根节点的堆并不一定满足大顶堆的性质）；

2.对交换后的n-1个序列元素进行调整，使其满足大顶堆的性质；

重复2.3步骤，直至堆中只有1个元素为止。

# 调整堆  
def adjust\_heap(lists, i, size):  
 lchild = 2 \* i + 1  
 rchild = 2 \* i + 2  
 max = i  
 if i < size / 2:  
 if lchild < size and lists[lchild] > lists[max]:  
 max = lchild  
 if rchild < size and lists[rchild] > lists[max]:  
 max = rchild  
 if max != i:  
 lists[max], lists[i] = lists[i], lists[max]  
 adjust\_heap(lists, max, size)  
  
# 创建堆  
 def build\_heap(lists, size):  
 for i in range(0, (size/2))[::-1]:  
 adjust\_heap(lists, i, size)  
  
# 堆排序  
def heap\_sort(lists):  
 size = len(lists)  
 build\_heap(lists, size)  
 for i in range(0, size)[::-1]:  
 lists[0], lists[i] = lists[i], lists[0]  
 adjust\_heap(lists, 0, i)

**冒泡排序**

def bubbleSort(nums):  
 for i in range(len(nums)-1): # 这个循环负责设置冒泡排序进行的次数  
 for j in range(len(nums)-i-1): # ｊ为列表下标  
 if nums[j] > nums[j+1]:  
 nums[j], nums[j+1] = nums[j+1], nums[j]  
 return nums

**快速排序**

def quick\_sort(lists, left, right):  
 # 快速排序  
 if left >= right:  
 return lists  
 key = lists[left]  
 low = left  
 high = right  
 while left < right:  
 while left < right and lists[right] >= key:  
 right -= 1  
 lists[left] = lists[right]  
 while left < right and lists[left] <= key:  
 left += 1  
 lists[right] = lists[left]  
 lists[right] = key  
 quick\_sort(lists, low, left - 1)  
 quick\_sort(lists, left + 1, high)  
 return lists

**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的一个典型的应用。它的基本操作是：将已有的子序列合并，达到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。

归并排序其实要做两件事：

分解：将序列每次折半拆分

合并：将划分后的序列段两两排序合并

因此，归并排序实际上就是两个操作，拆分+合并

如何合并？

L[first...mid]为第一段，L[mid+1...last]为第二段，并且两端已经有序，现在我们要将两端合成达到L[first...last]并且也有序。

首先依次从第一段与第二段中取出元素比较，将较小的元素赋值给temp[]

重复执行上一步，当某一段赋值结束，则将另一段剩下的元素赋值给temp[]

此时将temp[]中的元素复制给L[]，则得到的L[first...last]有序

如何分解？

在这里，我们采用递归的方法，首先将待排序列分成A,B两组；然后重复对A、B序列分组；直到分组后组内只有一个元素，此时我们认为组内所有元素有序，则分组结束。

def merge(left, right):  
 i, j = 0, 0  
 result = []  
 while i < len(left) and j < len(right):  
 if left[i] <= right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
 result += left[i:]  
 result += right[j:]  
 return result  
  
def merge\_sort(lists):  
 # 归并排序  
 if len(lists) <= 1:  
 return lists  
 num = len(lists) / 2  
 left = merge\_sort(lists[:num])  
 right = merge\_sort(lists[num:])  
 return merge(left, right)

**性能比较**

