7个经典排序算法

快速排序

基本思想：
1. 从数列中挑出一个元素，称谓“基准”
2. 重新排序数列，所有比基准值小的元素放在基准前面，所有比基准值大的元素放在基准后面（相同的数可以放到任何一边）。在这个分割结束之后，该基准就处于数列的中间位置。
3.递归地把小于基准元值素的子树列和大于基准值元素的子树列排序

快速排序又称为划分交换排序，简称快排。快速排序使用分治法策略把一个序列分为两个子序列

实现示例

```python
def quick_sort(lst):
    if len(lst) <= 1: return lst
    less = []
    greater = []
    pivot = lst.pop()

    for item in lst:
        if item < pivot:
            less.append(item)
        else:
            greater.append(item)
    lst.append(pivot)
    return quick_sort(less) + [pivot] + quick_sort(greater)
```

冒泡排序

算法介绍：是一种简单的排序算法。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。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数
3. 针对所有的元素重复以上的步骤，除了最后一个
4. 持续每次对越来越少的元素重复上面步骤，直到没有任何一对数字需要比较

实现示例

```python
def bubble_sorted(interable):
    new_list = list(interable)
    list_len = len(new_list)
    for i in range(list_len - 1):
        for j in range(list_len - 1, i, -1):
            if new_list[j] < new_list[j -1]:
                new_list[j], new_list[j -1 ] = new_list[j -1], new_list[j]
    return new_list

## Test
testlist = [27, 33, 28, 4, 2, 26, 13, 35, 8, 14]
print "bubble_sorted: ", bubble_sorted(testlist)
```

堆排序

堆排序是指利用堆这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆积的性质：即子节点的键值或索引总是小于（或者大于）它的父节点

堆节点的访问：
通常堆是通过一维数组来实现。在数组起始位置为0的情形中
1. 父节点i的左子节点在位置（2i + 1）
2. 父节点i的右子节点在位置（2i + 2）
3. 子节点i的父节点在位置floor((i − 1) / 2)

堆的操作：
1. 最大堆调整(Max Heapify): 将堆的末端子节点作调整，使得子节点永远小于父节点
2. 创建最大堆(Build Max Heap): 将堆中的所有数据重新排序
3. 堆排序(HeapSort): 移除位在第一个数据的根节点，并做最大堆调整的递归运算

实现示例

```python
#-*- coding:cut-8 -*-

def heap_sort(lst):
    def sift_down(start, end):
        """最大堆调整"""
        root = start
        while True:
            child = 2 * root + 1
            if child > end:
                break
            if child + 1 <= end and lst[child] < lst[child + 1]:
                child += 1
            if lst[root] < lst[child]:
                lst[root], lst[child] = lst[child], lst[root]
                root = child
            else:
                break
    # 创建最大堆
    for start in xrange((len(lst) - 2) // 2, -1, -1):
        sift_down(start, len(lst) - 1)

    # 堆排序
    for end in xrange(len(lst) - 1, 0, -1):
        lst[0], lst[end] = lst[end], lst[0]
        sift_down(0, end - 1)
    return lst

def main():
    l = [0, 2, 1, 7, 6, 8, 5, 3, 4]
    print heap_sort(l)

if __name__ == "__main__":
    main()
```

插入排序

算法介绍：是一种简单直观的排序算法。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

1. 从第一元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素大于新元素，将该元素移到下一个位置
4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到新位置后，重复步骤2-5

实现示例

```python
def insert_sort(lst):
    if n == 1: return lst
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j] < lst[j-1]:
                lst[j], lst[j-1] = lst[j-1],lst[j]
            else:
                break
    return lst
```

归并排序

归并排序是创建在归并操作上一种有效的排序算法，该算法是采用分治法的一个非常典型的应用，且各层分治递归可以同时进行

归并操作，也叫归并算法，指将两个已经排序的序列合并成一个序列的操作。
算法思想（递归法）：
1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列
2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置
3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到小一个位置
4. 重复步骤3直到某一指针到达序列尾
5. 将另一序列剩下的所有元素直接复制到合并序列尾

实现示例

```python
# Recursively implementation of merge sort
def merge(left, right):
    result = []
    while left and right:
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    if left:
        result += left
    if right:
        result += right
    return result

def merge_sort(L):
    if len(L) <= 1:
        return L
    mid = len(L) // 2
    left = L[:mid]
    right = L[mid:]

    left = merge_sort(left)
    right = merge_sort(right)

    return merge(left, right)

if __name__ == "__main__":
    test = [1, 4, 2, 3.6, -1, 0, 25, -34, 8, 9, 1, 0]
    print("original:", test)
    print("Sorted:", merge_sort(test))
```

选择排序

工作原理：
1. 在未排序序列中找到最小（大）元素，存放到排序序列的起始位置
2. 再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾
3. 重复1-2步骤，直到所有元素均排序完毕

实现示例

```python
def selection_sort(arr):
    for i in range(len(arr)):
        minIndex = i
        for j in range(i + 1, len(arr)):
            if arr[minIndex] > arr[j]:
                minIndex = j
        if i == minIndex:
            pass
        else:
            arr[i], arr[minIndex] = arr[minIndex], arr[i]
    return arr
```

希尔排序

希尔排序也称为递减增量排序算法，是插入排序的改进版本。它通过将比较的全部元素分为几个区域来提升插入排序的性能

工作原理：
将数组列在一个表中并对列排序，重复这过程，不过每次用更长的列来进行。最后整个表就只有一列。」

实现示例

```python
def shell_sort(list):
    n = len(list)
    //初始步长
    gap = n // 2
    while gap > 0:
        for i in range(gap, n):
            # 每个步长进行插入排序
            temp = list[i]
            j = i
            # 插入排序
            while j >= gap and list[j - gap] > temp:
                list[j] = list[j - gap]
                j -= gap
            list[j] = temp

        # 新的步长
        gap = gap // 2
    return list
```