# -\*- coding:utf-8 -\*-

import sys

from collections import deque

import copy

"""

数组的排序算法，包括各种算法的递归排序，非递归排序，以及不新建数组的排序

"""

"""

稳定的排序算法：直接插入排序，冒泡排序，归并排序

算法1：堆排序，先用数组建立最大堆，然后每次从堆中取出最大的一个数字，调整最大堆: 时间复杂度O(nlogn)

算法2：快速排序，每次找出第一个元素所在位置: O(nlogn), 最坏O(n^2)

算法3：归并排序，采用二分查找的方法，替换原数组的位置：时间复杂度O(n logn)

算法4：归并排序，不用进行替换: 时间复杂度O(n logn)

算法5：希尔排序：最好O(N),平均O（nlogn）最坏O(n^2)

算法6：简单选择排序: 时间复杂度为O(N^2)

算法7：冒泡排序: 时间复杂度为O(N^2)

算法8：简单插入排序:最好情况下为O(N) 最坏情况下为O(N^2)

"""

class Array(object):

def main(self, array):

self.array = array

# 堆排序

array = copy.copy(self.array)

self.heap\_sort(array)

print "heap: ",array

# 快速排序

array = copy.copy(self.array)

# self.quick\_sort(array, 0, len(array)-1)

self.quick\_sort\_no\_rec(array)

print "quick: ", array

# 归并排序，原地替换

array = copy.copy(self.array)

self.merge\_sort\_place(array, 0, len(array)-1)

print "merge replace: ", array

# 归并排序，创建新的

array = copy.copy(self.array)

array = self.merge\_sort\_create(array)

print "merge create: ", array

# 简单选择排序

aray = copy.copy(self.array)

array = self.select\_sort(array)

print "select: ",array

# 冒泡排序，对相邻元素进行交换，每次将最最大元素推向最后一个

array = copy.copy(self.array)

array = self.bubble(array)

print "bubble: ",array

# 直接插入排序

array = copy.copy(self.array)

array = self.insert\_sort(array)

print "insert: ", array

# 希尔排序，直接插入排序的增量版

array = [49, 38, 65, 97, 76, 13, 27, 49, 55, 4]

array = copy.copy(self.array)

array = self.shell\_sort(array)

print "shell: ", array

# 堆排序

def heap\_sort(self, array):

"""

堆排序的思想：先对列表中的所有数建立最大堆

假设列表有n个元素，列表已经是一个最大堆

建立最大堆：将堆中所有数据进行排序，建堆的过程其实就是不断调整堆的过程，从len/2进行调整，一直到第一个

将列表中第一个元素与第n个元素进行置换，则列表中最后一个元素就是最大值，然后对堆进行调整

将列表中第一个元素与第n-1个元素进行置换，则列表中最后一个元素就是最大值，然后对堆进行调整

...

:return:

"""

size = len(array)

# 建立最大堆，i表示第几个数字

i = size/2

while i > 0:

# 不停调整堆的过程

self.heapdown(array, i, size)

i = i-1

i = size

while i >= 1:

# 堆的顶点元素改变了，需要重新调整堆，满足最大堆

array[0], array[i-1] = array[i-1], array[0]

# 调整最大堆

self.heapdown(array, 1, i-1)

i = i-1

def heapdown(self, array, i, size):

"""

最大堆调整，size表示要调整的最大堆范围

:param heap:

:return:

"""

# 由于i<=size/2，任何一个节点i都有左子节点，

# 将子节点和根节点进行比较，如果子节点大于根节点，则将子节点和根节点进行置换。

while i <= size/2:

mc = self.minchild(array, i, size)

# 若子节点元素大于根节点，则替换

if array[mc-1] > array[i-1]:

array[mc-1], array[i-1] = array[i-1], array[mc-1]

i = mc

def minchild(self, array, i, size):

"""

寻找根节点中最大的元素

:param array:

:param i:

:return:

"""

if i\*2 + 1 > size:

return i\*2

else:

if array[i\*2-1] > array[i\*2]:

return i\*2

else:

return i\*2 + 1

# 快速排序

def quick\_sort(self, array, left, right):

if left >= right:

return

mid = self.part(array, left, right)

self.quick\_sort(array, left, mid)

self.quick\_sort(array, mid+1, right)

# 快排非递归实现

def quick\_sort\_no\_rec(self,array):

"""

快排的非递归实现，使用堆栈存储左右节点

:param array:

:return:

"""

left = 0

right = len(array)-1

stack = []

if left < right:

stack.append([left,right])

while len(stack)>0:

res = stack.pop()

mid = self.part(array,res[0],res[1])

if mid>res[0]:

stack.append([left,mid])

if mid+1<res[1]:

stack.append([mid+1,right])

def part(self, array, left, right):

tmp = array[left]

while left < right:

if left < right and array[right] > tmp:

right -= 1

array[left] = array[right]

if left<right and array[left] < tmp:

left += 1

array[right] = array[left]

array[left] = tmp

return left

# 归并排序

def merge\_sort\_place(self, array, left, right):

"""

:param array:

:param left:

:param right:

:return:

"""

if left >= right:

return

mid = (left + right)/2

self.merge\_sort\_place(array, left, mid)

self.merge\_sort\_place(array, mid+1, right)

i = left

j = mid+1

# 判断left和right的第一个数，若left要大些，则有可能right中存在多个数比left数小，。将两者进行交换，right指针向后移一个

# 若left要小写，则将left指针向后移一个，继续判断

# while i<= mid and j <= right:

# while array[i] < array[j]:

# i += 1

# while array[i] > array[j]:

# j += 1

# # 此时array[mid+1: j]中的数字全部小于array[i]，将两者进行交换

# res = array[i:mid+1]

# # 第一部分的长度

# # length\_left = mid - i + 1

# # 将两者进行交换

# length\_right = j - mid - 1

# array[i:i+length\_right] = array[mid+1:j]

# array[i+length\_right:j] = res

#

# # 第二种方式，通过数组反转

# array[i:mid+1].reverse()

# array[mid+1:j].reverse()

# array[i:j].reverse()

while (i < j) and (j <= right):

while (i < j) and (array[i] < array[j]):

i += 1

old\_j = j

while (j <= right) and (array[j] < array[i]):

j += 1

# 将两块进行交换

array[i:old\_j] = array[i:old\_j][::-1]

array[old\_j:j]=array[old\_j:j][::-1]

array[i:j]=array[i:j][::-1]

i += j-old\_j

# 归并排序，创建新的

def merge\_sort\_create(self, array):

"""

归并排序，不需要原地替换数组

:param array:

:param left:

:param right:

:return:

"""

if len(array) == 1:

return array

mid = len(array)/2

num1 = self.merge\_sort\_create(array[:mid])

num2 = self.merge\_sort\_create(array[mid:])

return self.combine(num1, num2)

def combine(self, num1, num2):

"""

将两个数组合并到一起

:param num1:

:param num2:

:return:

"""

new\_num = []

i = 0

j = 0

while i < len(num1) and j < len(num2):

if num1[i] < num2[j]:

new\_num.append(num1[i])

i += 1

else:

new\_num.append(num2[j])

j += 1

if i!=len(num1):

new\_num.extend(num1[i:])

else:

new\_num.extend(num2[j:])

return new\_num

def shell\_sort(self, array):

"""

希尔排序

:param array:

:return:

"""

dk = len(array)/2

while dk >= 1:

self.shell\_insert\_sort(array, dk)

dk = dk/2

return array

def shell\_insert\_sort(self, array, dk):

"""

插入排序

:param array:

:param i:

:param dk:

:return:

"""

# 从下标为dx的数进行插入排序

for i in range(dk, len(array)):

position = i

# 为间隔为dk的简单插入排序,在position之前的数字都是排好序，若后面数字更小，则不停的进行交换，position必须要大于等于dk

while position-dk >= 0 and array[position] < array[position-dk]:

array[position], array[position-dk] = array[position-dk], array[position]

position = position - dk

def select\_sort(self,array):

"""

简单选择排序

:return:

"""

for i in range(len(array)-1):

# 表示最小的元素下标，找到最小元素，则将两者进行交换

min = i

tmp = array[i]

for j in range(i+1, len(array)):

if array[j] < tmp:

min = j

tmp = array[j]

# 进行交换

array[i], array[min] = array[min], array[i]

return array

def bubble(self, array):

"""

冒泡排序

:param array:

:return:

"""

for i in range(len(array)-1):

flag = True

for j in range(len(array)-i-1):

if array[j] > array[j+1]:

array[j], array[j+1] = array[j+1], array[j]

flag = False

if flag:

break

return array

def insert\_sort(self, array):

"""

插入排序，将数组的前一部分排好序，然后将后面的第一个数字插入已经排好序的数组中

:param array:

:return:

"""

for i in range(1, len(array)):

j = i

while (j > 0) and array[j] < array[j-1]:

array[j], array[j-1] = array[j-1], array[j]

j = j-1

return array

test = Array().main([-1,26,5,77,1,61,11,59,15,48,19])