Test1：

* **设*X*[0:n-1]和*Y*[0:n-1]为两个数组，每个数组中含有*n*个已排好序的数。试设计一个O(log*n*)时间的分治算法，找出*X*和*Y*的2*n*个数的中位数，并证明算法的时间复杂性为O(log*n*)**
  + 个数为奇数,则处于最中间位置的数
  + 个数为偶数,则中间两个数据的平均数

由中位数定义可知,中位数即将有序数组二等分的那个数，即中位数左边有n个比中位数小的数右边有n个比中位数大的数。

由于有两个有序数组，考虑中位数定义，假设在数组X中取i个数将X分为两个部分，在数组Y中取j个将其分为两部分，只要i+j=n-i+n-j时即j=(2n+1)/2时X[0:i-1]UY[0:j-1]和X[i:n-1]UY[j:n-1]为大数组的两个相等大小部分。只要X[i-1]<Y[j]且Y[j-1]<X[i]那么中位数即为

由于时间复杂度要求为O(logn)所以考虑使用二分查找的方法找到对应的i。

While imin<imax:

i=(imin+imax)/2

j=n-i

若Y[n-i-1]≤X[i] 且 X[i−1]≤Y[[j] 则循环结束

若X[i]<Y[n-i-1]则代表i过小 imin=i+1

若 Y[j]<X[i-1],则代表i过大 imax=i-1

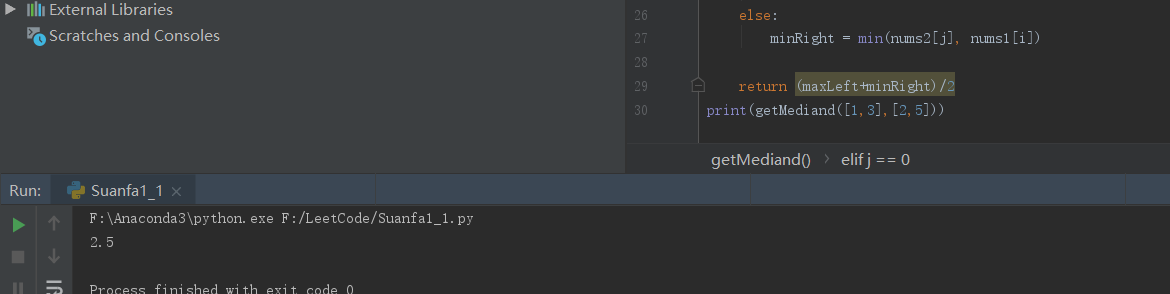
Return

由于该算法相当于在n长数组中用二分法寻找一个特定数，所以该算法时间复杂度为O(logn)

Code：

from typing import List  
def getMediand(nums1:List[int],nums2:List[int])->int:  
 #传入两个相同大小的有序数组找到其中位数  
 lenth=len(nums1)  
 imax=lenth  
 imin=0  
 while imin<imax:  
 i=(imin+imax)//2  
 j=lenth-i  
 if nums1[i-1]>nums2[j] and i>0:  
 imin=i+1  
 elif nums2[j-1]>nums2[i] and i<lenth:  
 imax=i-1  
 else:  
 break  
 if i == 0 :  
 maxLeft = nums2[j-1]  
 elif j == 0:  
 maxLeft = nums1[i-1];  
 else :  
 maxLeft = max(nums1[i-1], nums2[j-1])  
 if (i == lenth):  
 minRight = nums2[j]  
 elif (j == lenth):  
 minRight = nums1[i]  
 else:  
 minRight = min(nums2[j], nums1[i])  
  
 return (maxLeft+minRight)/2

实验截图：



Test2：

* **有一实数序列，若 且 ，则构成了一个逆序对，请使用分治方法求整个序列中逆序对个数，并分析算法的时间复杂性。**
  + 例如：序列(4,3,2)逆序对有(4,3)，(4,2)，(3,2)共3个

题目使用分治算法，采用类似归并排序的思想，使用递归解决问题。

将整个数组的逆序问题看成两个有序子数组合并产生逆序数的问题。

此算法使用了类归并排序的方法，实际上就是在归并排序操作中添加了逆序计数的操作，故时间复杂度与归并排序相同为O(nlogn)

复杂度分析：

T(n)=O(1) n=1；T(n)=T(n/2)+1 n>1；

使用Master定理，f(n)=nlog21 =1，所以T(n)=O(nlogn)。

Sum=0

Inversion（List）{

Left=Inversion（List（0:len/2））

Right=Inversion(List(len/2:))

{nlist=sort(Left,Right)(采用指针移动的方法对子数组排序，并累计逆序数个数sum)}  
return nlist

}

Print’ sum

Code：

from typing import List

sum=0

def inversion(nums:List[int]):

lenth = len(nums)

global sum

if lenth == 1:

return nums

else:

left=inversion(nums[0:lenth//2])

right= inversion(nums[lenth//2:])

rlist=[]

i=0

j=0

while i<len(left)and j<len(right):

if(left[i]<right[j]):

rlist.append(left[i])

i+=1

else:

rlist.append(right[j])

sum+=len(left)-i

j+=1

for i in range(i,len(left)):

rlist.append(left[i])

for j in range(j,len(right)):

rlist.append(right[j])

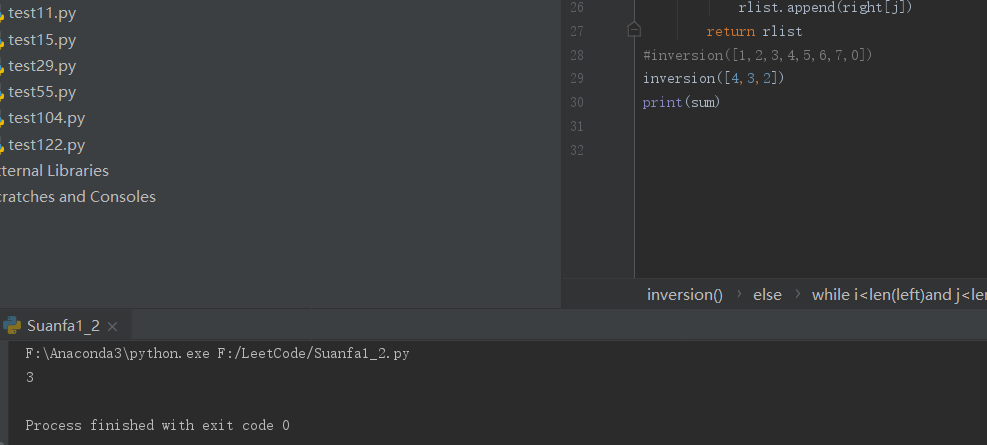
return rlist

#inversion([1,2,3,4,5,6,7,0])

inversion([4,3,2])

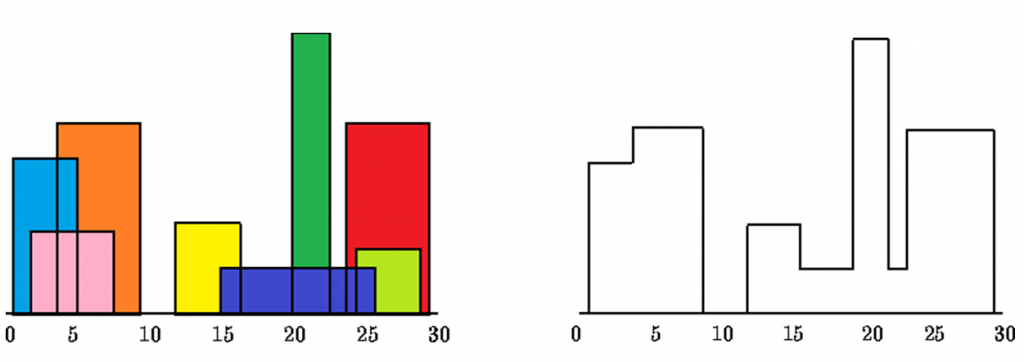
print(sum)

实验截图：



Test3：

* **给定座建筑物，每个建筑物表示为一个矩形，用三元组表示，其中表示建筑左下顶点，表示建筑的右下顶点，表示建筑的高，请设计一个的算法求出这座建筑物的天际轮廓。例如，左下图所示中8座建筑的表示分别为(1,5,11), (2,7,6), (3,9,13), (12,16,7), (14,25,3), (19,22,18), (23,29,13)和(24,28,4)，其天际轮廓如右下图所示可用9个高度的变化(1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13)和(29,0)表示。另举一个例子，假定只有一个建筑物(1, 5, 11)，其天际轮廓输出为2个高度的变化(1, 11), (5, 0)。**



使用分治思想

一整个大区域的天际线可以看作左右两个区域天际线的子问题。合并天际线列表时操作，天际线是左右两个天际线较高的那个。

算法：

If lenth==1:

Return [buildings[0][0][2],buildings[0][1][0]]

Left=skyline(buildings[:lenth/2])

Right=skyline(building[lenth/2:])

Return mix(left,right)

。5

Code：

def getSkyline(buildings):

if not buildings: return []

if len(buildings) == 1:

return [[buildings[0][0], buildings[0][2]], [buildings[0][1], 0]]

mid = len(buildings) // 2

left = getSkyline(buildings[:mid])

right = getSkyline(buildings[mid:])

return merge(left, right)

def merge(left, right):

# 记录目前左右建筑物的高度

lheight = rheight = 0

# 位置

l = r = 0

res = []

while l < len(left) and r < len(right):

if left[l][0] < right[r][0]:

cp = [left[l][0], max(left[l][1], rheight)]

lheight = left[l][1]

l += 1

elif left[l][0] > right[r][0]:

cp = [right[r][0], max(right[r][1], lheight)]

rheight = right[r][1]

r += 1

else:

cp = [left[l][0], max(left[l][1], right[r][1])]

lheight = left[l][1]

rheight = right[r][1]

l += 1

r += 1

# 和前面高度比较，不一样才加入

if len(res) == 0 or res[-1][1] != cp[1]:

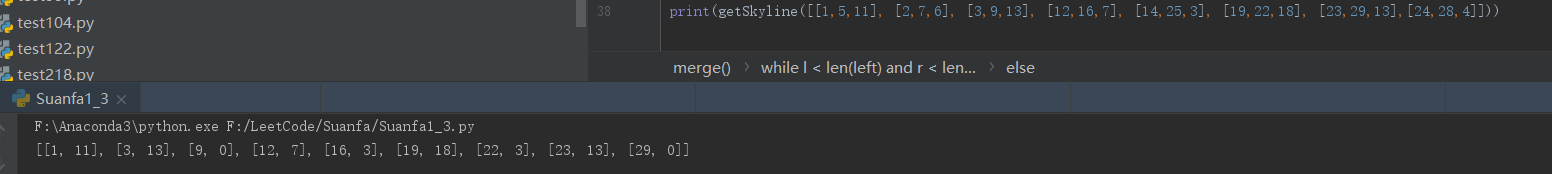
res.append(cp)

# 剩余部分添加进去

res.extend(left[l:] or right[r:])

return res

实验截图：



Test4：

* **最大子数组问题。一个包含n个整数（有正有负）的数组A，设计一O(nlogn)算法找出和最大的非空连续子数组。对于此问题你还能设计出O(n)的算法吗？**
  + **例如：[0, -2, 3, 5, -1, 2]应返回9，**
  + **[-9, -2, -3, -5, -3]应返回-2**

为了采用分治策略得到最大子数组，将一个大数组的最大子数组问题分解为，左子数组的最大子数组，右子数组的最大子数组和跨越中心的最大子数组三个子问题。其中跨过中心的最大子数组和可以通过分别求左右两边子数组最大值再相加的方式得到时间复杂度为O(nlogn)。

Code:

from typing import List

def getmMaxSum1(nums:List[int])->int:

lenth=len(nums)

if lenth==1:

return nums[0]

left=getmMaxSum1(nums[:lenth//2])

right=getmMaxSum1(nums[lenth//2:])

#开始求跨越中心点的最大值

lsum=nums[lenth//2-1]

sum=0

for i in range(lenth//2-1,-1,-1):

sum=sum+nums[i]

if sum>lsum:

lsum=sum

rsum=nums[lenth//2]

sum=0

for i in nums[lenth//2:]:

sum=sum+i

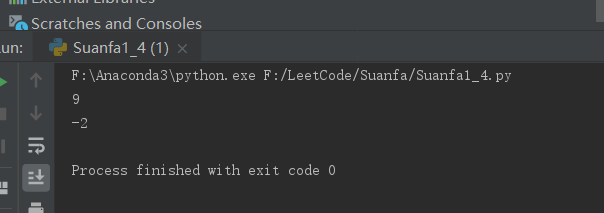
if sum>rsum:

rsum=sum

mid=rsum+lsum

return max(mid,right,left)

实验截图:



考虑问题特性，设置maxsum和sum分别储存目前最大的连续和和当前和。由于问题特性，当sum<0时对sum重新赋值。

遍历数组，当sum>maxsum时更新maxsum

由于该算法只遍历了一次数组，所以时间复杂度为O(n)

Code：

from typing import List

def getmMaxSum(nums:List[int])->int:

lenth = len(nums)

sum = 0

maxsum = nums[0]

for i in range(0, lenth):

if sum > 0:

sum = sum + nums[i]

else:

sum = nums[i]

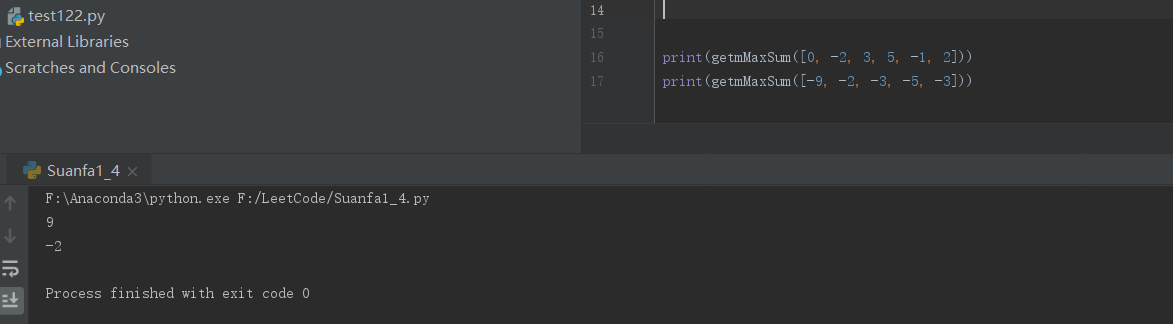
maxsum = max(maxsum, sum)

return maxsum

print(getmMaxSum([0, -2, 3, 5, -1, 2]))

print(getmMaxSum([-9, -2, -3, -5, -3]))

实验截图：



Test5:

* **循环移位问题。给定一个数组，数组中元素按从小到大排好序，现将数组中元素循环右移若干位，请设计一算法，计算出循环右移了多少位。**

由题可知，遍历数组，当第一次出现nums[i]<nums[i+1]时此时i+1（也可能是（i+1）+n\*len）即为移位位数。由于此算法只遍历了一次数组所以时间复杂度为O(n)

Code：

from typing import List

def get\_shiftnum(nums:List[int]):

lenth=len(nums)

if lenth==1:

return 0

i=0

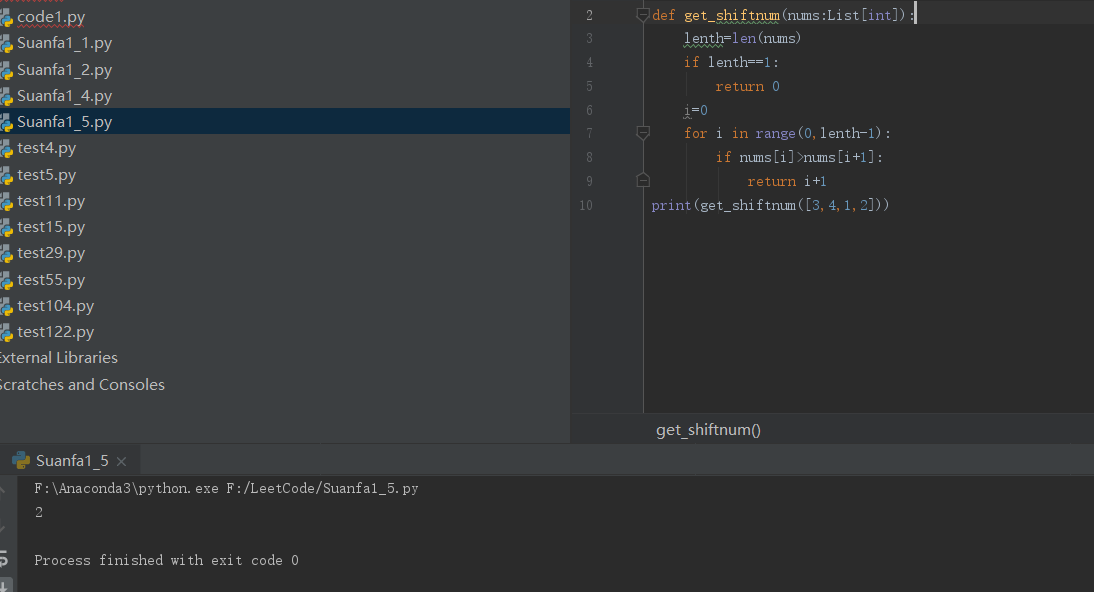
for i in range(0,lenth-1):

if nums[i]>nums[i+1]:

return i+1

print(get\_shiftnum([3,4,1,2]))

实验截图：



Test6:

* **两元素和为X。给定一个由n 个实数构成的集合S 和另一个实数x，判断S 中是否有两个元素的和为x。试设计一个分治算法求解上述问题，并分析算法的时间复杂度。**

先将数组由小到大排序，使用双指针法。

I=0

Flag=0

j=len-1

While(i<j):

If nums[i]+nums[j]>x:

i+=1

else if nums[i]+nums[j]<x:

j-=1

else:

flag=1

break

由于该算法使用了排序算法，时间复杂度为O(nlogn)，后双指针法时间复杂度为O(n)。故综合下来该算法时间复杂度为O(nlogn)

Code:

from typing import List

def get\_max(nums:List[int],x)-> bool:

flag=False

lenth=len(nums)

nums=sorted(nums)

i=0

j=lenth-1

while i<j:

if nums[i]+nums[j]>x:

j-=1

elif nums[i]+nums[j]<x:

i+=1

else:

flag=True

break

return flag

print(get\_max([9,1,2,3,4,5],14))

实验截图：

