使用几种排序算法测试性能，其中冒泡排序性能太差，忽略

Python冒泡排序

def maopao(s):

length = len(s)

for i in range(length-1):

for j in range(i+1,length):

if s[i]>s[j] : change(s,i,j)

def change(s,i,j):

temp=s[i]

s[i]=s[j]

s[j]=temp

import numpy

mu,sigma=0,0.1 #均值与标准差

s=numpy.random.normal(mu,sigma,3000)

maopao(s)

s

输出：[1, 2, 3, 4, 5, 7, 9]

测试感觉：不应该用冒泡排序来测大数据排序的，本来是3百万数据排序，结果电脑直接卡死，无法动弹

Python插入排序

def insert(s):

length = len(s)

for i in range(1,length):

j=i-1

while j>=0:

if s[j]<=s[j+1]:

break

else:

change(s,j+1,j)

j=j-1

测试结果：插入排序性能比插入排序稍微好一点，除了思想不一样也没啥不同了。可以考虑

选择排序

def choose\_sort(s):

length = len(s)

for i in range(length):

t=i

for j in range(i+1,length):

if s[j] < s[t]:

t=j

change(s,t,i)

测试结果：与插入排序、冒泡排序一样，性能非常差，完全不能与python内置的sort()、sorted()相比，内置排序函数一次排大小为1000000正态分布随机数组，只需要3s左右，用上面自己写的排30000的电脑就挂掉了，一下子温度烧的好厉害，等5，6分钟没啥反应（算法之间的差距就是这么明显）

Shell排序

＃选用Knuth在1969年提出的步长序列：1 4 13 40 121 364 1093 3280 。。。后一个元素是前一个元素\*3+1

def shellSort(seq):

length=len(seq)

inc=0

while inc<=length/3:

inc=inc\*3+1

#print(inc)

while inc>=1:

for i in range(inc,length):

tmp=seq[i]

for j in range(i,0,-inc):

if tmp<seq[j-inc]:

seq[j]=seq[j-inc]

else:

j+=inc

break

seq[j-inc]=tmp

inc//=3

测试结果：非常不想写希尔排序，4层for循环，这是在网上找的优化后的3层，很神奇的是他的平均效率是O(n1.5)……

但是上网查，在大数据处理方面，能和快排竞争的就是希尔排序，而且希尔排序的理论要优于快排，还是要好红啊学一下希尔排序。

归并排序

def merge\_sort(s):

length=len(s)

mid\_length=length//2

if length<2:

return s

merge\_sort(s[:mid\_length])

merge\_sort(s[mid\_length:])

s\_left=[]

s\_right=[]

s\_left[:mid\_length]=s[:mid\_length]

s\_right=s[mid\_length:]

s\_right=list(reversed(s\_right))

s\_left[mid\_length:]=s\_right

begin=0

end=length-1

for i in range(length):

if s\_left[begin]>s\_left[end]:

s[i]=s\_left[end]

end-=1

else:

s[i]=s\_left[begin]

begin+=1

return s

快速排序

def quick\_sort(s,low,high):

t=choose(s,low,high)

temp\_value=s[t]

begin=low

end=high

change(s,t,high)

while low<high:

while low<high and s[low]<=temp\_value:

begin+=1

while low<high and s[high]>=temp\_value:

high-=1

change(s,low,high)

change(s,end,low)

quick\_sort(s,begin,high)

quick\_sort(s,low+1,end)

return s

def choose(s,low,hign):

return (low+high)//2

def change(s,i,j):

temp=s[i]

s[i]=s[j]

s[j]=temp

堆排序

def fixDown(a,k,n): #自顶向下堆化，从k开始堆化

N=n-1

while 2\*k<=N:

j=2\*k

if j<N and a[j]<a[j+1]: #选出左右孩子节点中更大的那个

j+=1

if a[k]<a[j]:

a[k],a[j]=a[j],a[k]

k=j

else:

break

def heapSort(l):

n=len(l)-1

for i in range(n//2,0,-1):

fixDown(l,i,len(l))

while n>1:

l[1],l[n]=l[n],l[1]

fixDown(l,1,n)

n-=1

return l[1:]