**Performance**

[Here](https://github.com/adnanaziz/publicadnan/blob/master/python/pda/insertionsort.py) is a simple sorting algorithm. Use %timeit to compute the largest array of uniform-[0,1] random floats that it can sort in 100ms, 1sec, 10sec. Report the same for the built in sorting algorithm. Include the commands you used.

**Solution**

**Code:**

def insertionsort(A):

for i in range(len(A)):

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

if A[i] > A[j]:

(A[i], A[j]) = (A[j], A[i])

B = []

import random

for i in range(0,1000000):

B.append(random.random())

Bsorted = sorted(B)

insertionsort(B)

assert B == Bsorted

%timeit [insertionsort(B)]

%timeit [sorted(B)]

# 100 ms - Insertion Sort

insrt100 = []

for i in range(0,912):

insrt100.append(random.random())

%timeit [insertionsort(insrt100)]

# 1 s - Insertion Sort

insrt1 = []

for i in range(0,2750):

insrt1.append(random.random())

%timeit [insertionsort(insrt1)]

# 10 s - Insertion Sort

insrt10 = []

for i in range(0,8000):

insrt10.append(random.random())

%timeit [insertionsort(insrt10)]

# 100 ms - Built in Sort

srt100 = []

for i in range(0,77000):

srt100.append(random.random())

%timeit [sorted(srt100)]

# 1 s - Built in Sort

srt1 = []

for i in range(0,600000):

srt1.append(random.random())

%timeit [sorted(srt1)]

# 10 s - Built in Sort

srt10 = []

for i in range(0,4500000):

srt10.append(random.random())

%timeit [sorted(srt10)]

**Output:**

1000 loops, best of 3: 1.29 ms per loop

100000 loops, best of 3: 8.16 µs per loop

10 loops, best of 3: 98.5 ms per loop (100 ms – Insertion Sort)

1 loops, best of 3: 1.01 s per loop (1 s – Insertion Sort)

1 loops, best of 3: 8.79 s per loop (10 s – Insertion Sort)

10 loops, best of 3: 96.7 ms per loop (100 ms – Built in Sort)

1 loops, best of 3: 1.02 s per loop (1 s – Built in Sort)

1 loops, best of 3: 9.91 s per loop (10 s – Built in Sort)

**Graphing**

The Mandelbrot set is the set of complex numbers c for which the sequence (c, c^2 + c, (c^2 + c)^2 + c, ... does not diverge. (Here x^2 denotes the square of x.)

Use NumPy to compute the Mandelbrot set over real part in [-1.0, 2.0] and imaginary part in [-1.5, 1.5].

Divide the region into a 1000x1000 grid, and perform 1000 iterations of the iteration for each point. Take points whose magnitude is less than 1.0 to be part of the set.

Show both your program and the plot it generated inline with your writeup.

Page 97 of PDA should help you get started. Note that Python has a built-in complex type, which you can write for example as 1.4 + 3.1j.

**Solution**

**Code:**

from numpy import \*

from pylab import \*

iterations = 1000

density = 1000

x\_min, x\_max = -1, 2

y\_min, y\_max = -1.5, 1.5

x, y = meshgrid(linspace(x\_min, x\_max, density),

linspace(y\_min, y\_max, density))

c = x + 1j\*y

z = c.copy()

m = zeros((density, density))

for n in xrange(iterations):

print "Iteration No." , n

indices = (abs(z) <= 1)

z[indices] = z[indices]\*\*2 + c[indices]

m[indices] = n

imshow(m,

extent=(x\_min, x\_max, y\_min, y\_max))

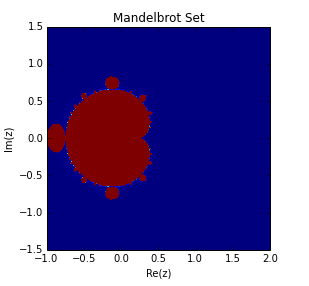
title('Mandelbrot Set')

xlabel('Re(z)')

ylabel('Im(z)')

show()

**Output:**



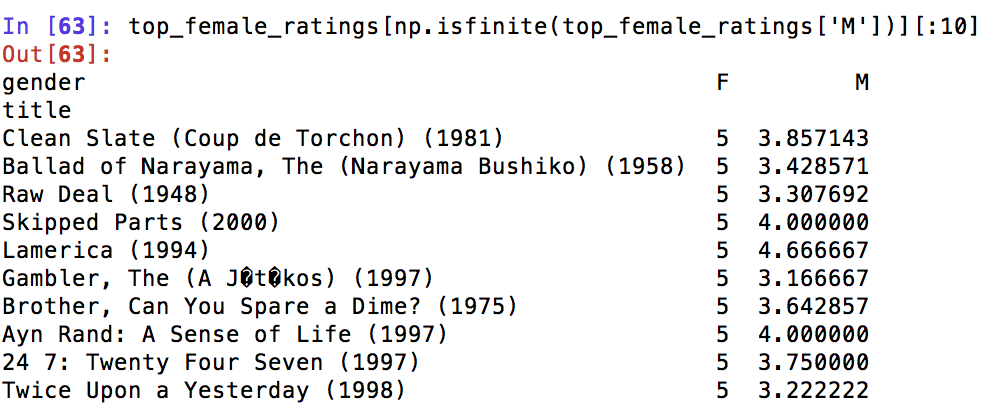
**Pandas**

Work through the MovieLens 1M DataSet example on pages 26-31. Specifically, through all the steps in that example, and also find the 10 movies that have the least disagreement among viewers, using the same metrics and methodology as in the example.

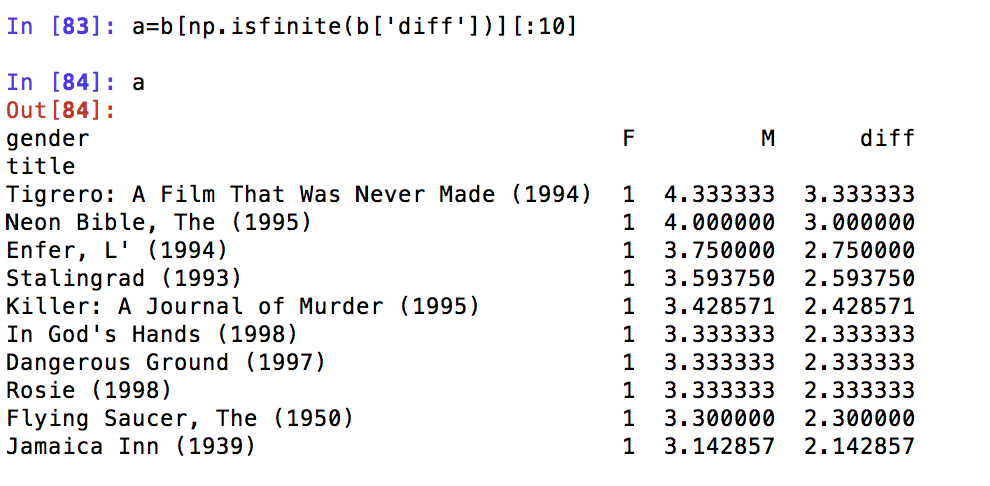
Turn in screenshots of the top 10 films among female viewers, the top 10 movies that were preferred by men which were not as highly rated by women, and the 10 movies with the least disagreement.

**Solution**

Top 10 films among female viewers:



The top 10 movies that were preferred by men which were not as highly rated by women:



10 movies with the least disagreement:

