In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
```

From Wikipedia: https://en.wikipedia.org/wiki/Brownian_motion (https://en.wikipedia.org/wiki/Brownian_motion)

```
Brownian motion is the random motion of particles suspended in a fluid (a
liquid or a gas) resulting from their collision with the quick atoms or mo
lecules in the gas or liquid. The term "Brownian motion" can also refer to
the mathematical model used to describe such random movements, which is of
ten called a particle theory.
```

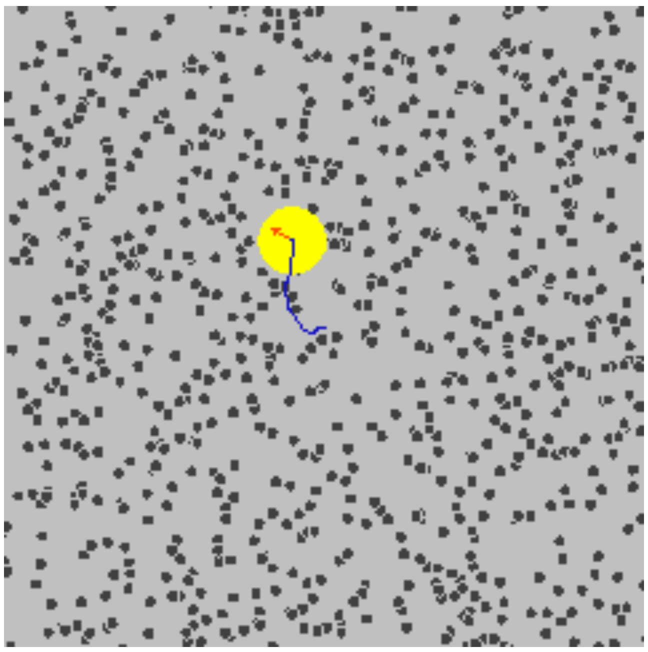
One algorithm (more of a *random walk*) is given by (http://wiki.scipy.org/Cookbook/BrownianMotion (http://wiki.scipy.org/Cookbook/BrownianMotion)):

1. $x(0) = x_0$, a starting position
2. $x(t + \Delta t) = x(t) + \mathcal{N}(0, \sigma^2 \Delta t; t, t + \Delta t)$, an update in a (normal) random direction

In [2]:

```
from IPython.display import Image
Image(url="https://upload.wikimedia.org/wikipedia/commons/c/c2/Brownian_motion_l
arge.gif")
```

Out[2]:



In [9]:

```
a=np.random.randn(2,1)
print(a)
print(a.shape)
print(a.ravel())
print(a.ravel().shape)
b=np.zeros((5,1),dtype=int)
print(b)
c=np.zeros((5,),dtype=int)
print(c)
```

```
[[-0.07006343]
 [-0.44559479]]
(2, 1)
[-0.07006343 -0.44559479]
(2,)
[[0]
 [0]
 [0]
 [0]
 [0]]
[0 0 0 0 0]
```

Let's try a an attempt at Brownian motion

In [10]:

```
def brownian1(n):
    x = np.zeros((2,n+1)) # starting position
    for k in xrange(1,n+1):
        x[:,k] = x[:,k-1] + np.random.randn(2,1).ravel()

    return x
```

Now time it!

In [11]:

```
n = 1000
%timeit x=brownian1(n)
```

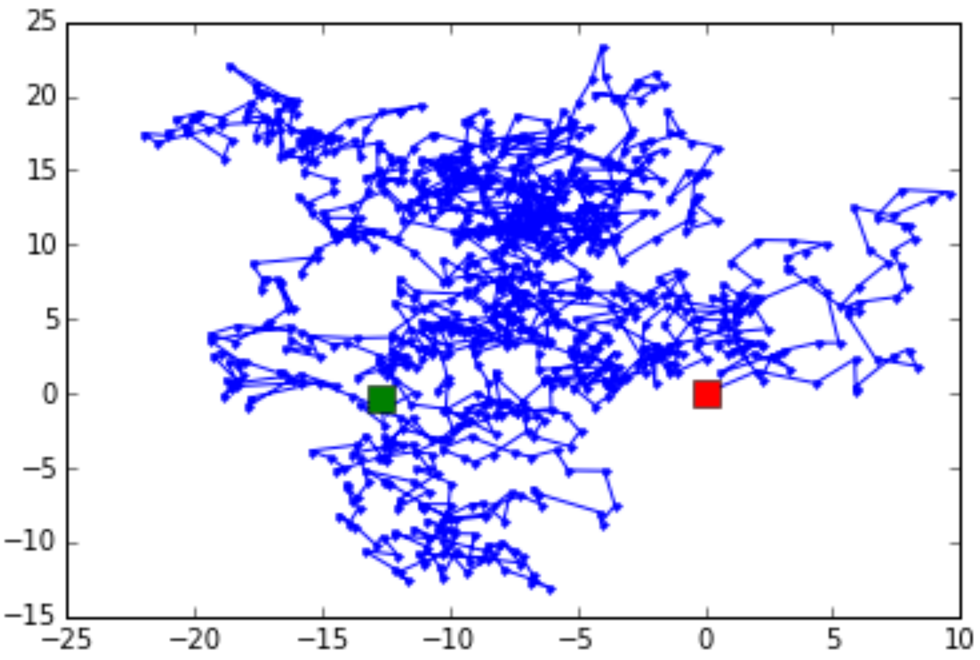
```
100 loops, best of 3: 7.23 ms per loop
```

In [26]:

```
#np.random.seed(98)
n = 1000
x = brownian1(n)
plt.plot(x[0,:], x[1,:], '.b-')
plt.plot(0,0,'rs', ms=10)
plt.plot(x[0,-1],x[1,-1],'gs', ms=10)
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x106aa8390>]
```



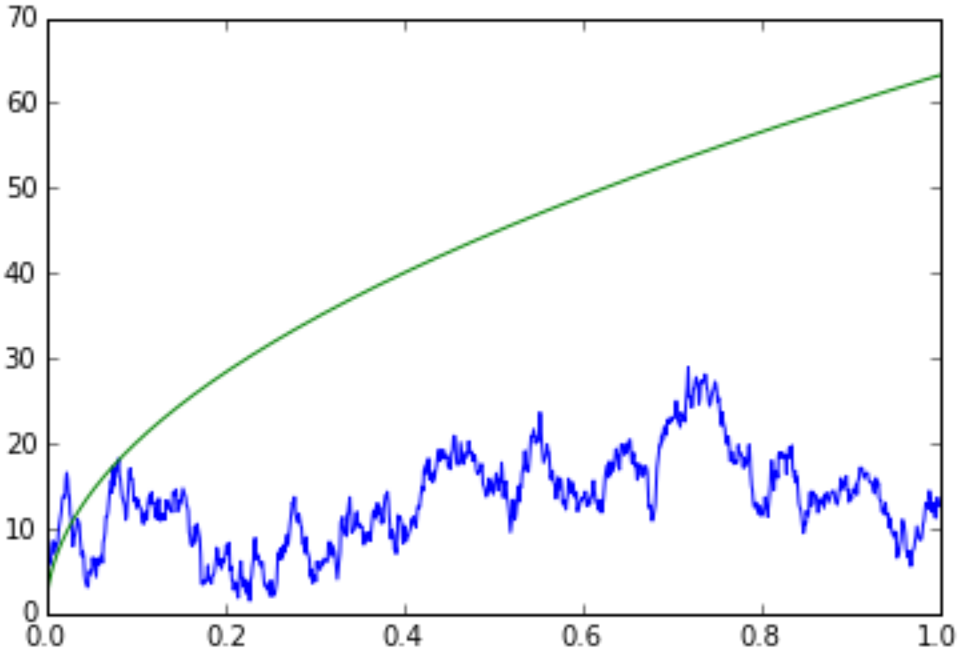
Question: Is the distance from green to red n^2 or n or $\sqrt{n}$ or something else?

In [27]:

```
r = np.sqrt(x[0,:]**2 + x[1,:]**2)
t = np.linspace(0,1,n+1)
theory = 2*np.sqrt(np.arange(n+1))
plt.plot(t,r)
plt.plot(t,theory)
```

Out[27]:

```
[<matplotlib.lines.Line2D at 0x106aca1d0>]
```



This seems reasonable, but let's try to speed up the computation by removing the loop:

In [28]:

```
def brownian2(n):
    x =  np.random.randn(2,n+1)
    x[:,0] = 0.0
    x = np.cumsum(x, axis=1)
    return x
```

Now time it!

In [29]:

```
n = 1000
%timeit x=brownian2(n)
```

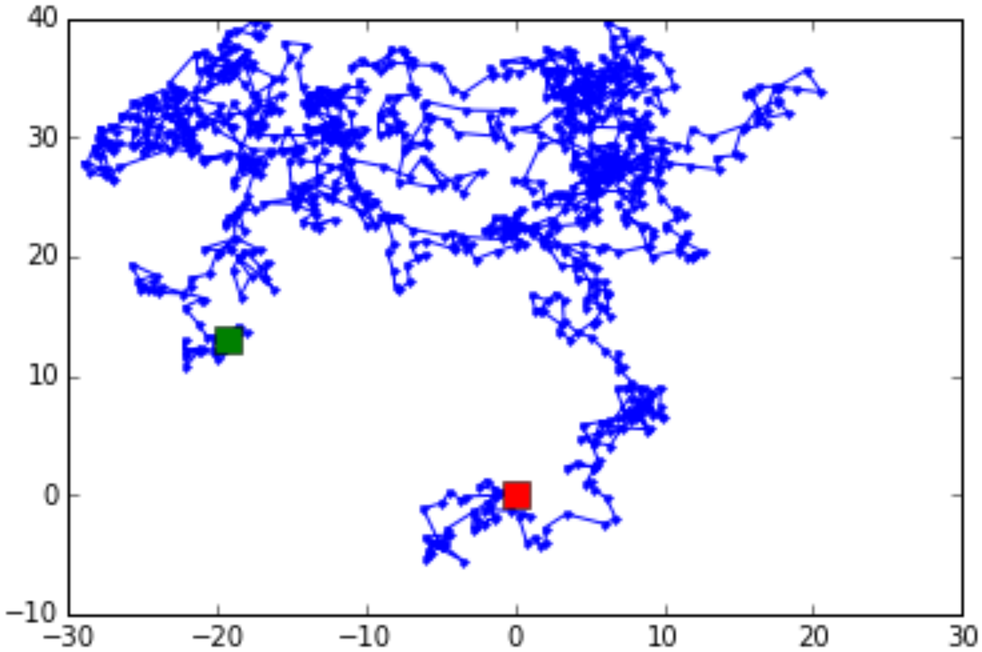
```
10000 loops, best of 3: 82 µs per loop
```

In [30]:

```
n=1000
x = brownian2(n)
plt.plot(x[0,:], x[1,:], '.b-')
plt.plot(0,0,'rs', ms=10)
plt.plot(x[0,-1],x[1,-1],'gs', ms=10)
```

Out[30]:

```
[<matplotlib.lines.Line2D at 0x106d0dd10>]
```

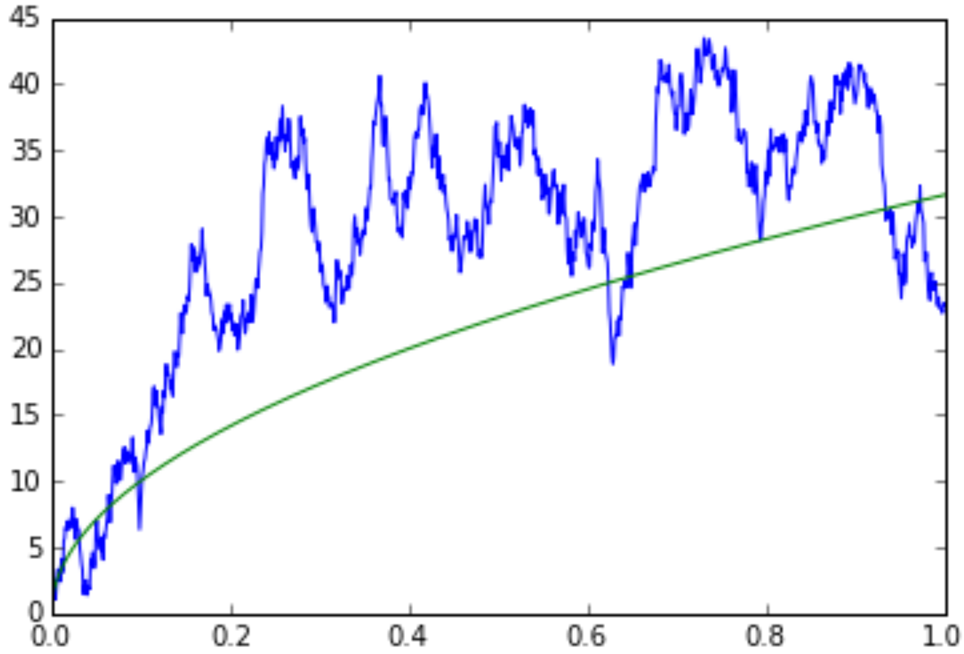


In [32]:

```
r = np.sqrt(x[0,:]**2 + x[1,:]**2)
t = np.linspace(0,1,n+1)
theory = np.sqrt(np.arange(n+1))
plt.plot(t,r)
plt.plot(t,theory)
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x106e24490>]
```



Now that we have a faster version, let's make a better "test".

Since things are random, let's try averaging over a number of random walks. Say 3 or 5 or 10. This may give a clearer picture of the dependence on n.

In [46]:

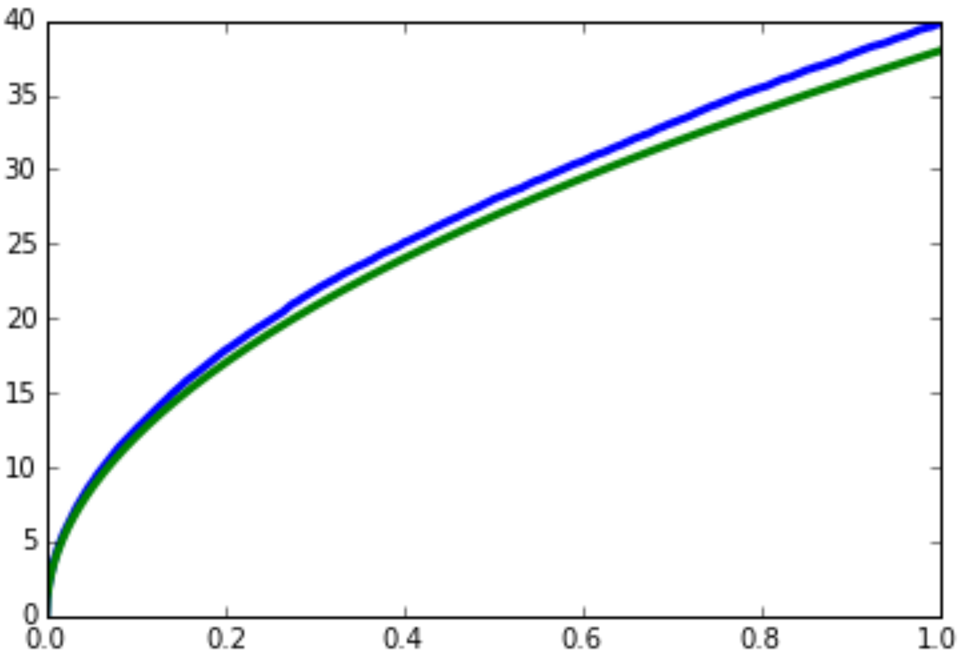
```
n=1000

m=10000
r=np.zeros((n+1,))
for i in range(m):
    x=brownian2(n)
    r += np.sqrt(x[0,:]**2 + x[1,:]**2)
r /= m
```

In [47]:

```
t = np.linspace(0,1,n+1)
theory = 1.2*np.sqrt(np.arange(n+1))
plt.plot(t,r, lw=3)
plt.plot(t,theory, lw=3)
```

Out[47]:

```
[<matplotlib.lines.Line2D at 0x10777a590>]
```



In [ ]:

```
```