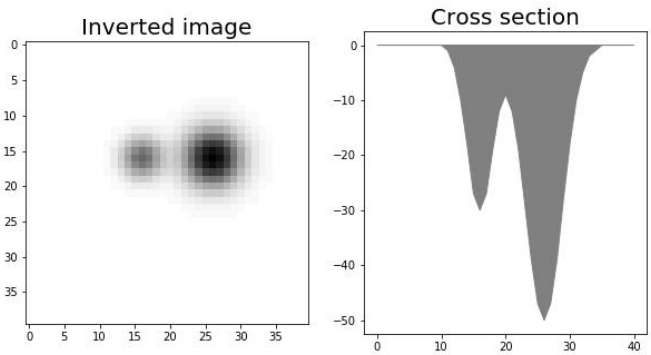
9. Watershed algorithm

In a number of cases, one is able to detect the positions of multiple objects on an image, but it might be difficult to segment them because they are close together or very irregular. This is where the wahtersehd algorithm is very practical. It takes as input an image, and a series of seeds and expands each region centered around a seed as if it was filling a topographic map.



In [1]:
```
from skimage.morphology import watershed
from skimage.measure import regionprops
```

In [2]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
plt.gray()
from skimage.external.tifffile import TiffFile
import skimage.io as io
from skimage.morphology import label

import course_functions
```

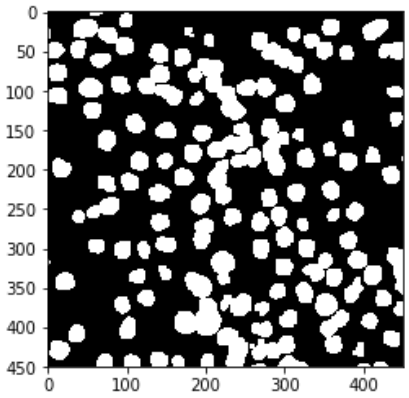
In [3]:
```
#load the image to process
image = io.imread('Data/BBBC007_v1_images/A9/A9 p9d.tif')
```

9.1 Create seeds

We can use the code of the last chapter to produce the seeds. We added the necessary code in our course module called course_functions

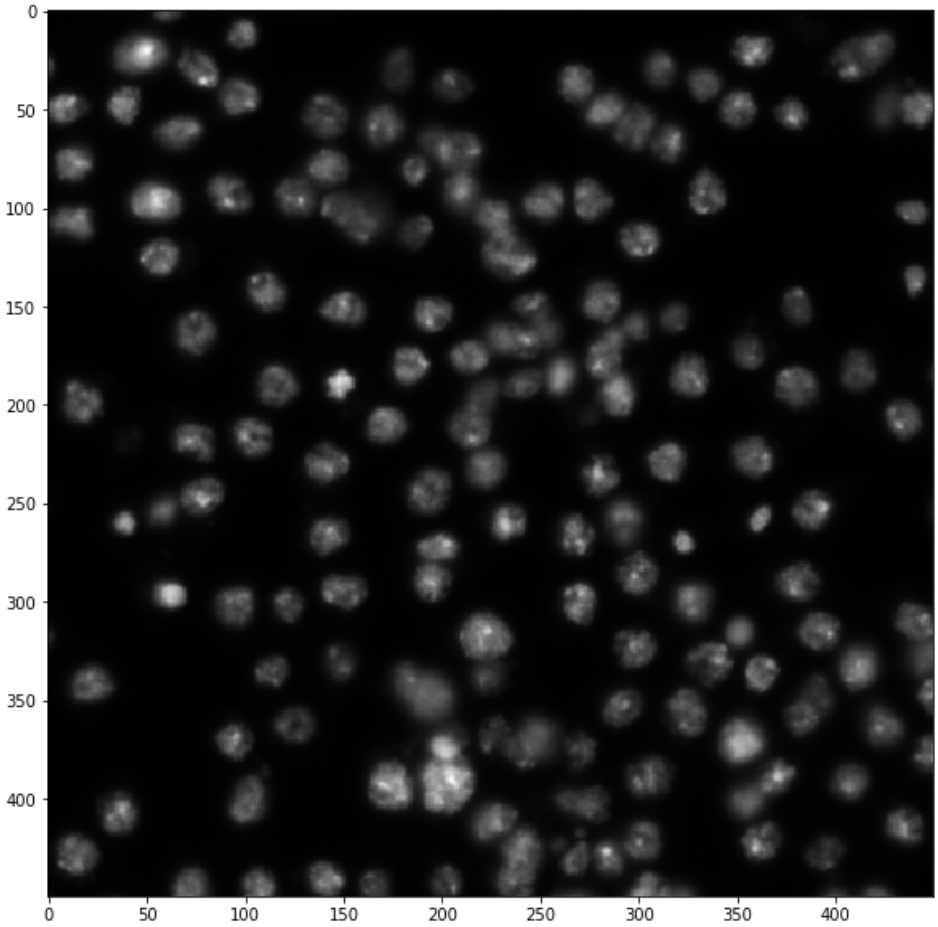
In [4]:
```
#generate template
template = course_functions.create_disk_template(10)
#generate seed map
seed_map, global_mask = course_functions.detect_nuclei_template(image, t
emplate)
```

In [5]:
```
plt.imshow(global_mask)
plt.show()
```

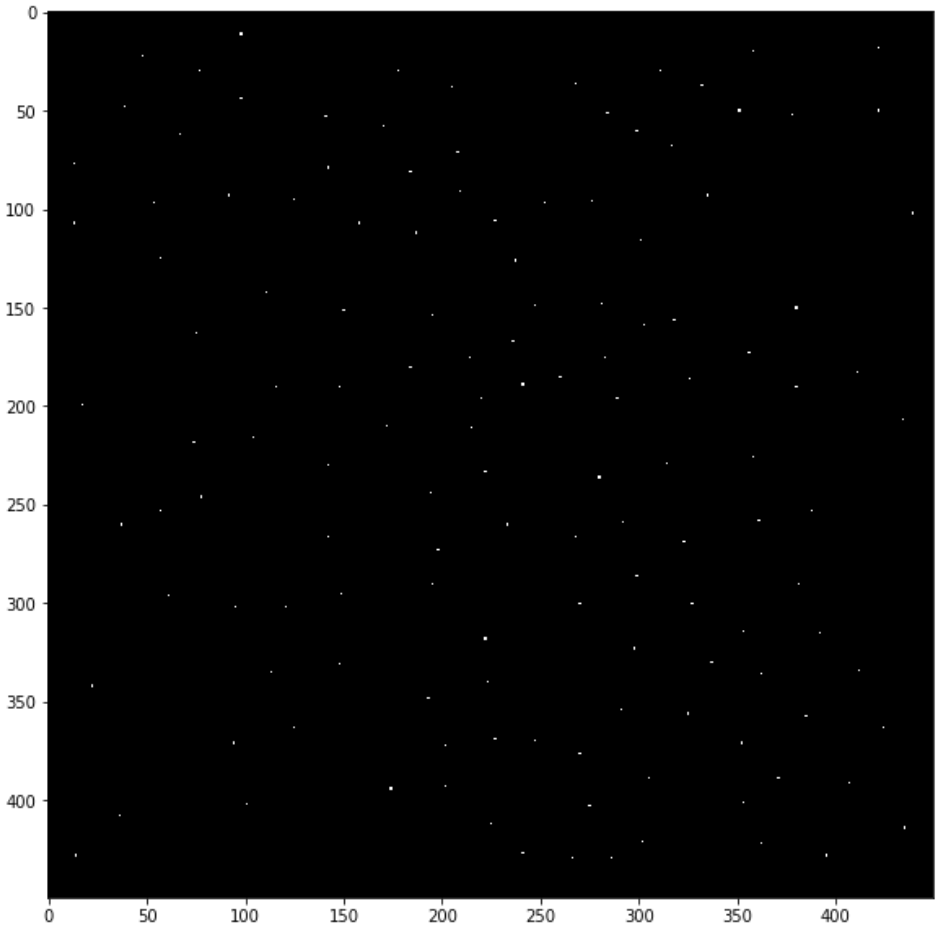


We need to create a labeled image, so that the watershed algorithm creates regions with different labels:

In [6]:
```
plt.figure(figsize=(10,10))
plt.imshow(image);
```



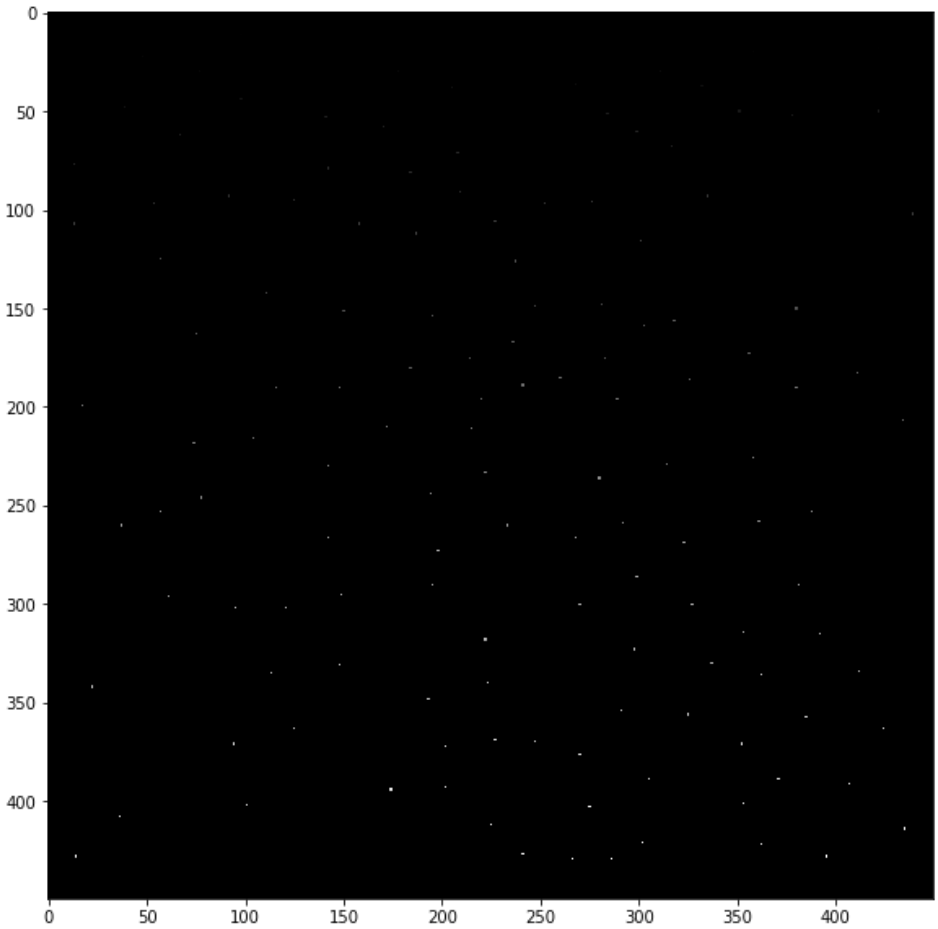
In [7]:
```
plt.figure(figsize=(10,10))
plt.imshow(seed_map);
```



In [8]:
```
seed_label = label(seed_map)
```

In [9]:
```
plt.figure(figsize=(10,10))
plt.imshow(seed_label)
```

Out[9]: <matplotlib.image.AxesImage at 0x7fb84e9f8ac8>



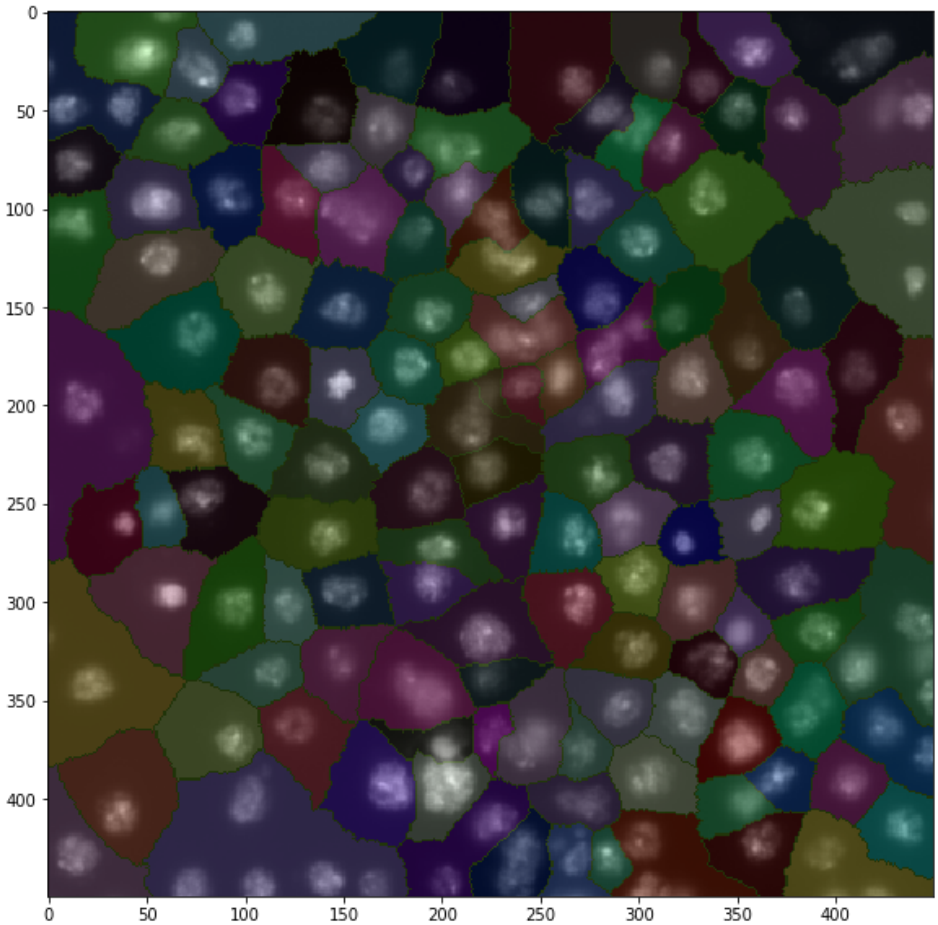
Now we can use the image and the labeled seed map to run the watershed algorithm. However, remember the analogy of filling a topographic map: our nuclei should be "deep" regions, so we need to invert the image. Finally we also require that a thin line separates regions (watershed_line option).

In [10]:
```
watershed_labels = watershed(image = -image, markers = seed_label, water
shed_line=True)
```

In [11]:
```
watershed_labels.max()
```

Out[11]: 136

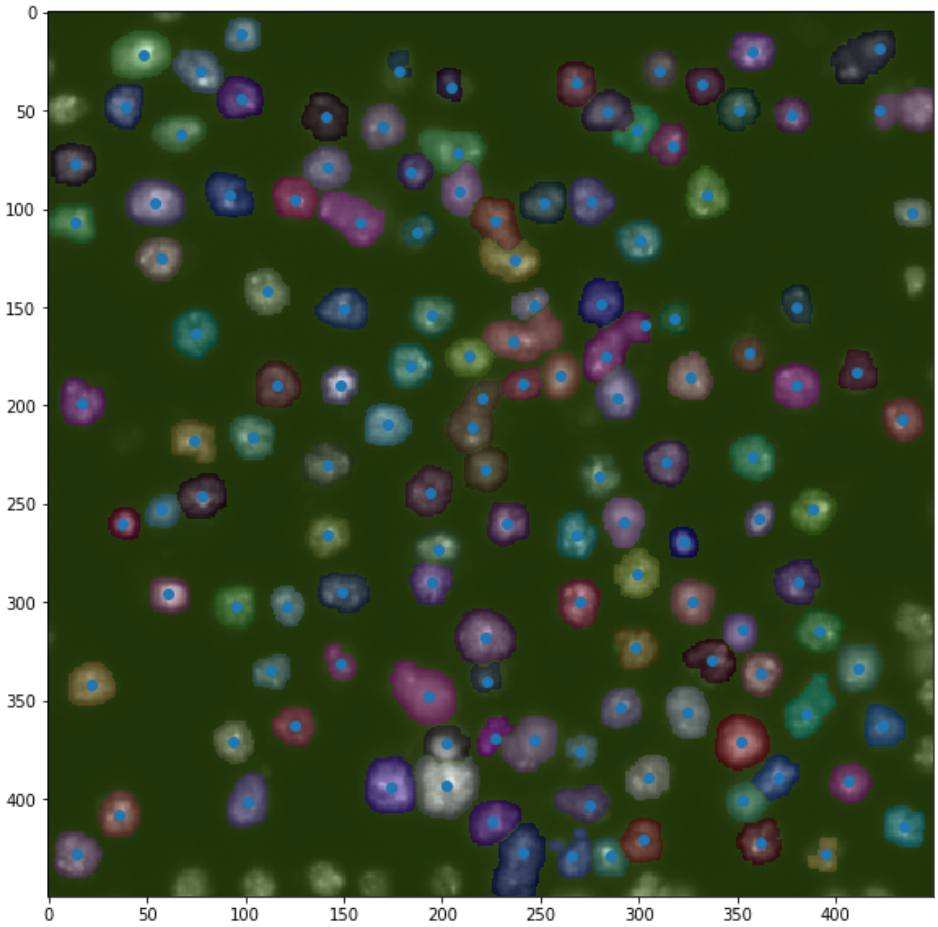
In [12]:
```
#create a random map
plt.figure(figsize = (10,10))

cmap = matplotlib.colors.ListedColormap ( np.random.rand ( 256,3))

plt.imshow(image)
plt.imshow(watershed_labels, cmap = cmap, alpha = 0.3);
```



The algorithm worked well and created regions around each nucleus. However we are only interested in the actual nuclei properties. So let's use our global masks to limit ourselves to those regions:

In [13]:
```
watershed_labels = watershed(image = -image, markers = seed_label, mask
= global_mask, watershed_line=True)
```

In [14]:
```
plt.figure(figsize = (10,10))
plt.imshow(image)
plt.imshow(watershed_labels, cmap = cmap, alpha = 0.3)
plt.plot(np.argwhere(seed_map)[:,1],np.argwhere(seed_map)[:,0],'o');
```



Finally, now that you have all the nuclei segmented you can proceed to do actual measurements e.g. by using the previously seen `regionprops` function.

In [15]:
```
myregions = regionprops(watershed_labels)
```

In [18]:
```
shape  = [x.area for x in myregions]
```

In [17]:
```
plt.hist(shape);
```