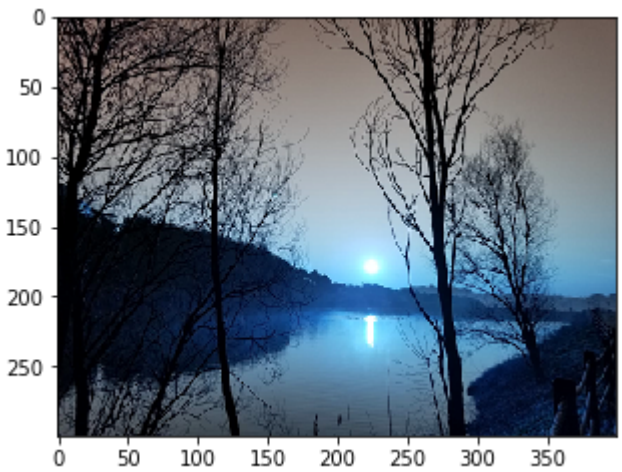
1. Download the “Sunny Lake” image.

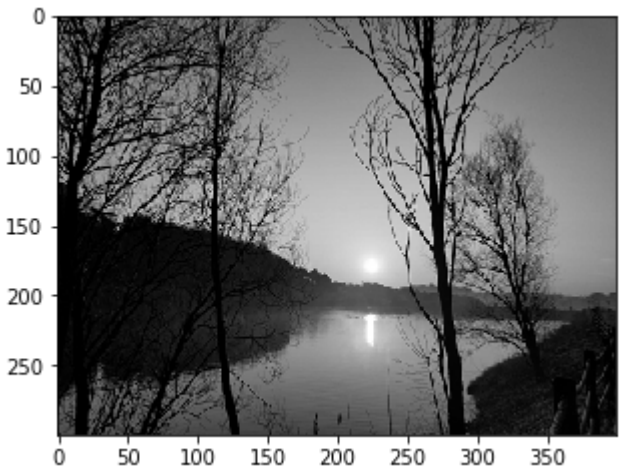
In [45]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import numpy as np

#%matplotlib qt
pathname = '/Users/erkamozturk/Desktop/hw1/SunnyLake.bmp'
pic = cv2.imread(pathname)
plt.figure(figsize = (5,5))

plt.imshow(pic)
```

Out[45]:
```
<matplotlib.image.AxesImage at 0x11a5f7c90>
```



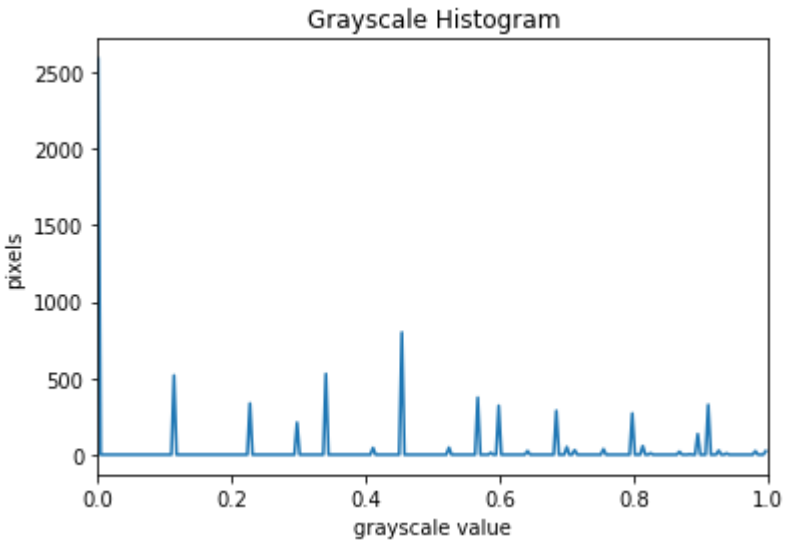
2. Obtain the gray scale image, I, by taking the average values of R, G, B channels.

In [46]:
```
gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
I = gray(pic)

plt.figure( figsize = (5,5))
plt.imshow(I, cmap = plt.get_cmap(name = 'gray'))
plt.show()
```



3. Obtain the histogram, h, of the gray scale image, I.

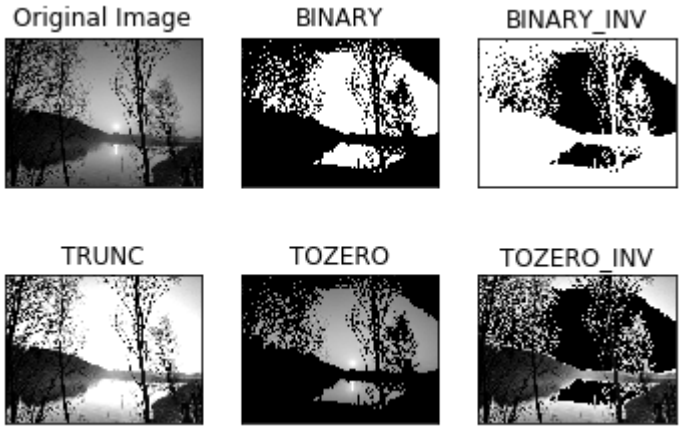
In [47]:
```
H, bin_edges = np.histogram(I, bins=256, range=(0, 1))

plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("grayscale value")
plt.ylabel("pixels")
plt.xlim([0.0, 1.0])  # <- named arguments do not work here

plt.plot(bin_edges[0:-1], H)  # <- or here
plt.show()
```



4&5. Inspect h and propose a threshold value, T, to segment the image into two parts and hence obtain a binary image, B:

In [48]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

pathname = '/Users/erkamozturk/Desktop/hw1/SunnyLake.bmp'
img = cv2.imread(pathname,0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV'
]
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
#%%  5. Present the output image B.
B = thresh1
```

In [49]:
```
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
```



6. Add the following zero mean Gaussian noises, separately to red, green and blue channels of 256x256 colored "Sunny Lake" image, with standard deviations of 1, 5, 10, 20. Show resulting images.

7. Obtain gray scale images, I_1, I_5, I_10 and I_20 by taking the average values of R, G, B channels corresponding to different noise levels.

In [60]:
```
noise_sigma = [1, 5, 10, 20]
pathname = '/Users/erkamozturk/Desktop/hw1/SunnyLake.bmp'
#lake_filename = "SunnyLake.bmp"
lake_image = cv2.imread(pathname, cv2.IMREAD_UNCHANGED)
lake_image = lake_image[:256, :256] # crop to 256 x 256
lake_grayscale_image = cv2.cvtColor(lake_image, cv2.COLOR_BGR2GRAY)

# play with the standard deviation
noise_sigma = noise_sigma[2]

temp_image = np.float64(np.copy(lake_grayscale_image))

h = temp_image.shape[0]
w = temp_image.shape[1]
noise = np.random.randn(h, w) * noise_sigma

noisy_image = np.zeros(temp_image.shape, np.float64)
if len(temp_image.shape) == 2:
    noisy_image = temp_image + noise
else:
    noisy_image[:,:,0] = temp_image[:,:,0] + noise
    noisy_image[:,:,1] = temp_image[:,:,1] + noise
    noisy_image[:,:,2] = temp_image[:,:,2] + noise
```

In [61]:
```
print('noisy image shape: {0}, len of shape {1}'.format(\
    lake_image.shape, len(noisy_image.shape)))
print(' WxH: {0}x{1}'.format(noisy_image.shape[1], noisy_image.shape[0]))
print(' image size: {0} bytes'.format(noisy_image.size))
```

```
noisy image shape: (256, 256, 3), len of shape 2
 WxH: 256x256
 image size: 65536 bytes
```

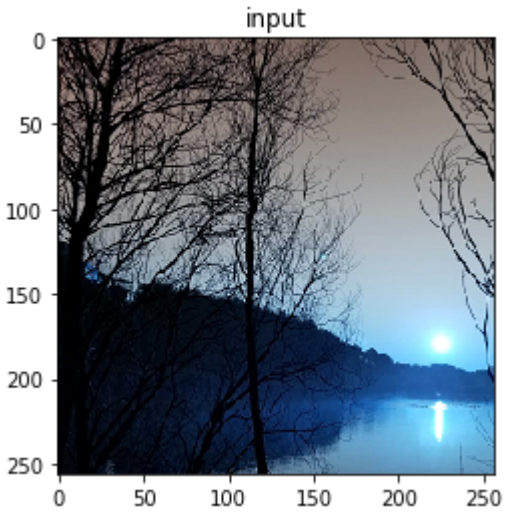
In [62]:
```
temp_image = np.float64(np.copy(noisy_image))
cv2.normalize(temp_image, temp_image, 0, 255, cv2.NORM_MINMAX, dtype=-1)
```

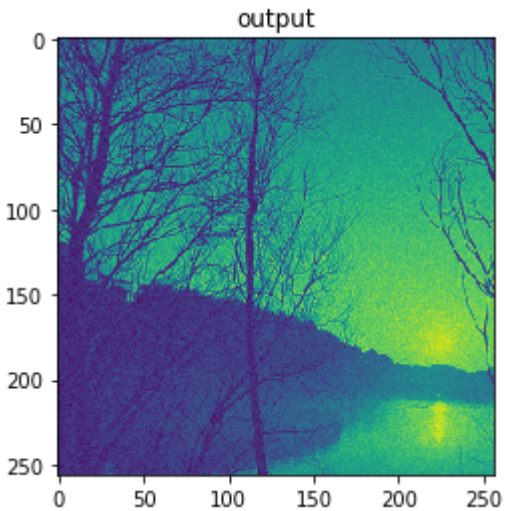
Out[62]:
```
array([[ 27.33314876,  21.36172403,  48.74402629, ..., 125.85371394,
          38.79888596,  16.84260227],
       [ 52.8163346 ,  39.12828752,  96.60798415, ...,  82.13645027,
          74.76894145,  31.86148039],
       [ 54.84117267,  57.58469581,  46.79109878, ...,  66.64983148,
         101.94769966,  28.03982939],
       ...,
       [ 42.00466504,  30.24309486,  40.76949081, ..., 162.15235816,
         147.75885684, 155.73531128],
       [ 31.95832959,  34.95110917,  35.60625267, ..., 152.71392101,
         153.04217649, 143.16502545],
       [ 28.17490344,  39.90967271,  25.6733462 , ..., 145.33975222,
         157.93074095, 150.90549231]])
```

because opencv.imshow will cause jupyter notebook crash

I used matplotlib.pyplot

In [77]:
```
plt.imshow(lake_image)
plt.title('input')
plt.show()

plt.imshow(temp_image.astype(np.uint8))
plt.title('output')
plt.show()



# Wait for user input; click X to destroy window.
#cv2.waitKey(3000)

# Destroy window and return to caller.
#cv2.destroyAllWindows()
```





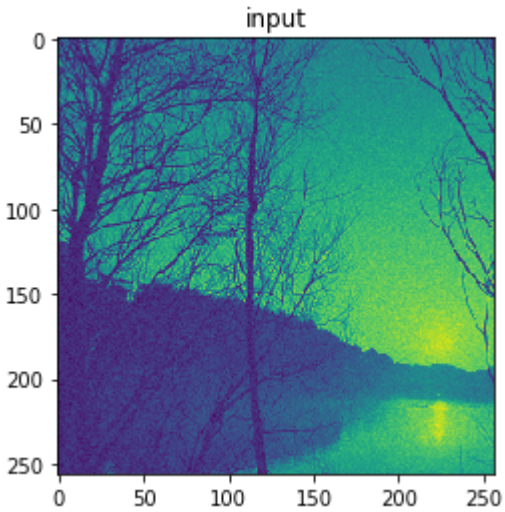
8. Filter these images using low-pass filters with kernels presented on pages 9 and 12 of “filter.pdf” document. Comment on the results.

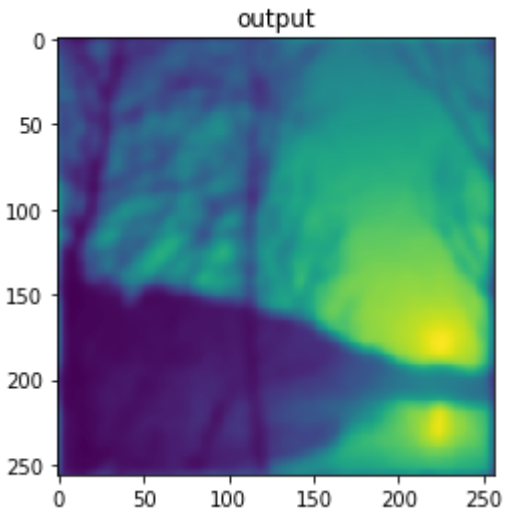
In [68]:
```
img = temp_image.astype(np.uint8)
img = img[:256, :256]
```

In [71]:
```
r = 50 # how narrower the window is
ham = np.hamming(256)[:,None] # 1D hamming
ham2d = np.sqrt(np.dot(ham, ham.T)) ** r # expand to 2D hamming

f = cv2.dft(img.astype(np.float32), flags=cv2.DFT_COMPLEX_OUTPUT)
f_shifted = np.fft.fftshift(f)
f_complex = f_shifted[:,:,0]*1j + f_shifted[:,:,1]
f_filtered = ham2d * f_complex
```

In [72]:
```
f_filtered_shifted = np.fft.fftshift(f_filtered)
inv_img = np.fft.ifft2(f_filtered_shifted) # inverse F.T.
filtered_img = np.abs(inv_img)
filtered_img -= filtered_img.min()
filtered_img = filtered_img*255 / filtered_img.max()
filtered_img = filtered_img.astype(np.uint8)
```

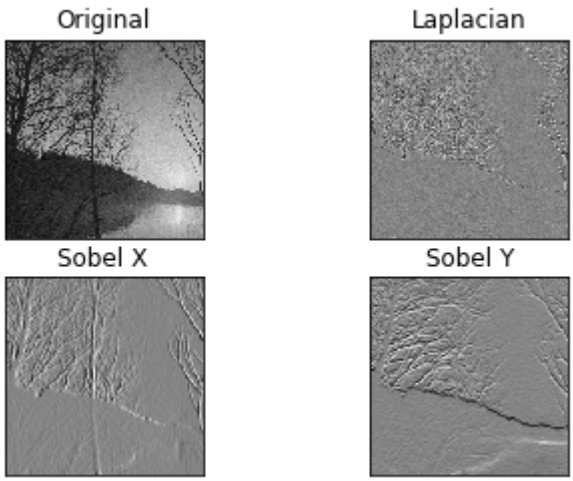
In [78]:
```
plt.imshow(img)
plt.title('input')
plt.show()

plt.imshow(filtered_img)
plt.title('output')
plt.show()
```





9. Filter images in 7) using high-pass filters with kernels presented on pages 17 and 19 of “filter.pdf” document. Comment on the results.

In [82]:
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = temp_image.astype(np.uint8)
img = img[:256, :256]
laplacian = cv.Laplacian(img,cv.CV_64F)
sobelx = cv.Sobel(img,cv.CV_64F,1,0,ksize=5)
sobely = cv.Sobel(img,cv.CV_64F,0,1,ksize=5)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

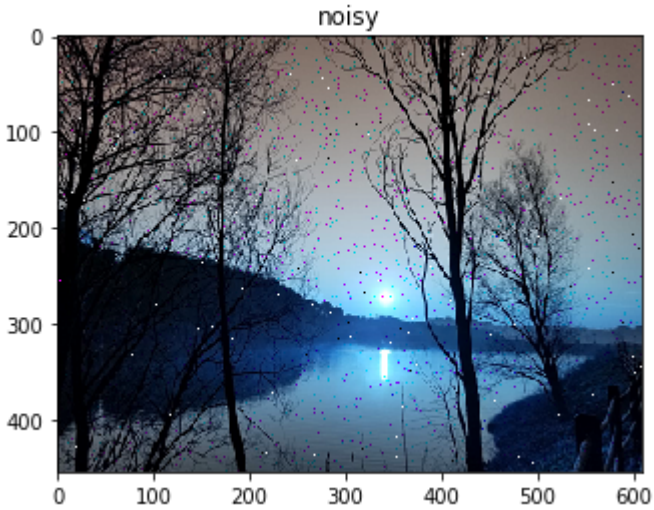


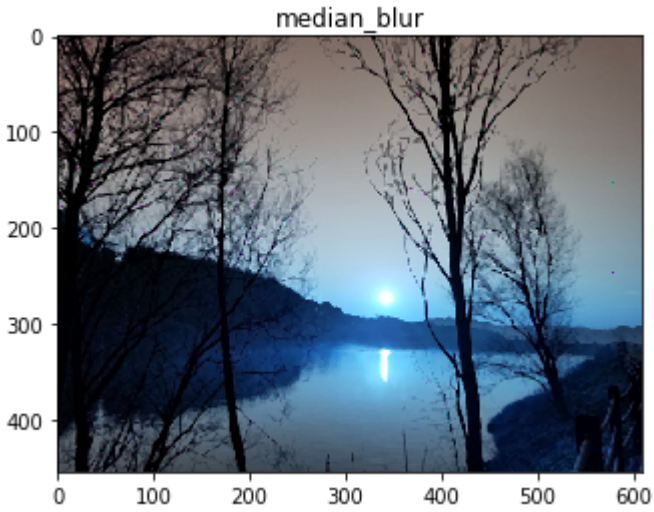
10. Inspect Figure-1. Comment on the type of noise and propose a method to de-noise the image. Implement your method and present the de-noised image.

In [81]:

```
import numpy as np
import cv2

pathname = '/Users/erkamozturk/Desktop/hw1/Figure_1.png'
img = cv2.imread(pathname, 1)
row,col,ch = img.shape
p = 0.5
a = 0.009
noisy = img

  # Salt mode
num_salt = np.ceil(a * img.size * p)
coords = [np.random.randint(0, i - 1, int(num_salt))
          for i in img.shape]
noisy[coords] = 1

  # Pepper mode
num_pepper = np.ceil(a * img.size * (1. - p))
coords = [np.random.randint(0, i - 1, int(num_pepper))
          for i in img.shape]
noisy[coords] = 0

#cv2.imshow('noisy', noisy)

plt.imshow(noisy)
plt.title('noisy')
plt.show()

#%%
median_blur= cv2.medianBlur(noisy, 3)

#cv2.imshow('median_blur', median_blur)

plt.imshow(median_blur)
plt.title('median_blur')
plt.show()
```

```
/Users/erkamozturk/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:15: FutureWarning: Using a non-tuple sequence for multidimensional inde
xing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futur
e this will be interpreted as an array index, `arr[np.array(seq)]`, which wil
l result either in an error or a different result.
  from ipykernel import kernelapp as app
/Users/erkamozturk/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launch
er.py:21: FutureWarning: Using a non-tuple sequence for multidimensional inde
xing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the futur
e this will be interpreted as an array index, `arr[np.array(seq)]`, which wil
l result either in an error or a different result.
```





In [ ]: