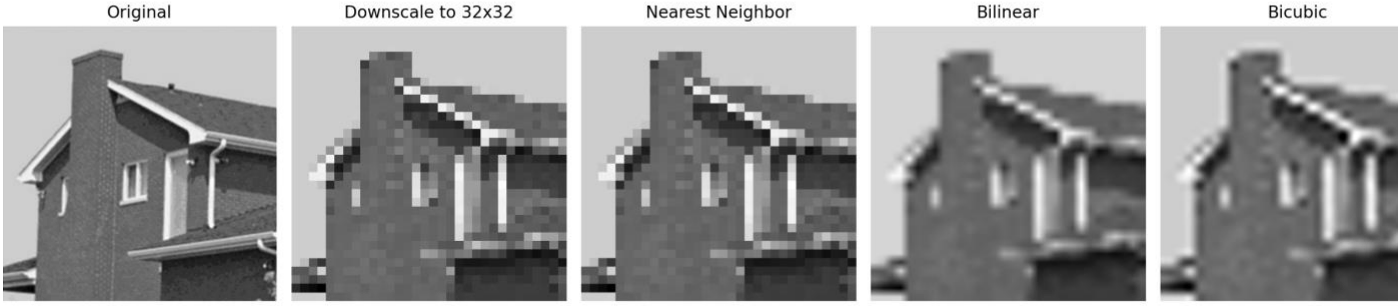
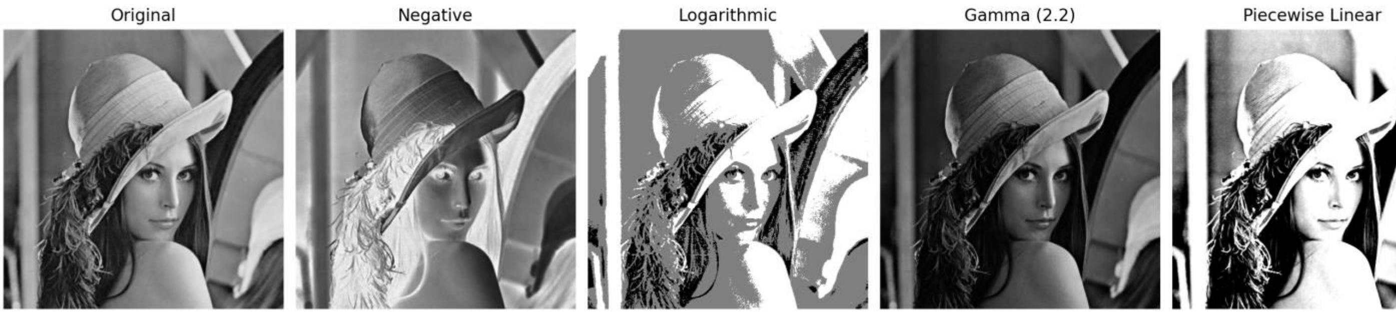
Q1> 1.



(a) Original: This is the original grayscale image of a house.
(b) Downscale to 32×32: The image is reduced dramatically in size to only 32×32 pixels. As a result, it's very pixelated, with visible blocky squares replacing fine details.
(c) Nearest Neighbor Interpolation (Upscaled): After upscaling the downscaled image using nearest neighbor interpolation, the pixelation remains.
(d) Bilinear Interpolation (Upscaled): Upscaling using bilinear interpolation creates a smoother appearance compared to nearest neighbor. Each new pixel is calculated as a weighted average of nearby original pixels.
(e) Bicubic Interpolation (Upscaled): Bicubic interpolation produces the smoothest and softest upscaled image among the three. Here we are accounting for 16 neighboring pixels when calculating each new pixel.

Q1> 2.



(a) Original: This is the standard grayscale image.
(b) Negative: The negative transformation inverts the pixel values, turning dark areas light and vice versa. Formula: 255 – r.
(c) Logarithmic: The logarithmic transformation brightens dark regions and compresses bright ones.
Formula: clog(1+r)
(d) Gamma (2.2): Gamma adjustment with y=2.2. It darkens the image overall. Formula: c*r**y
(e) Piecewise Linear: Here the value of (r1, s1) = (70,100) and (r2, s2) = (140,180).

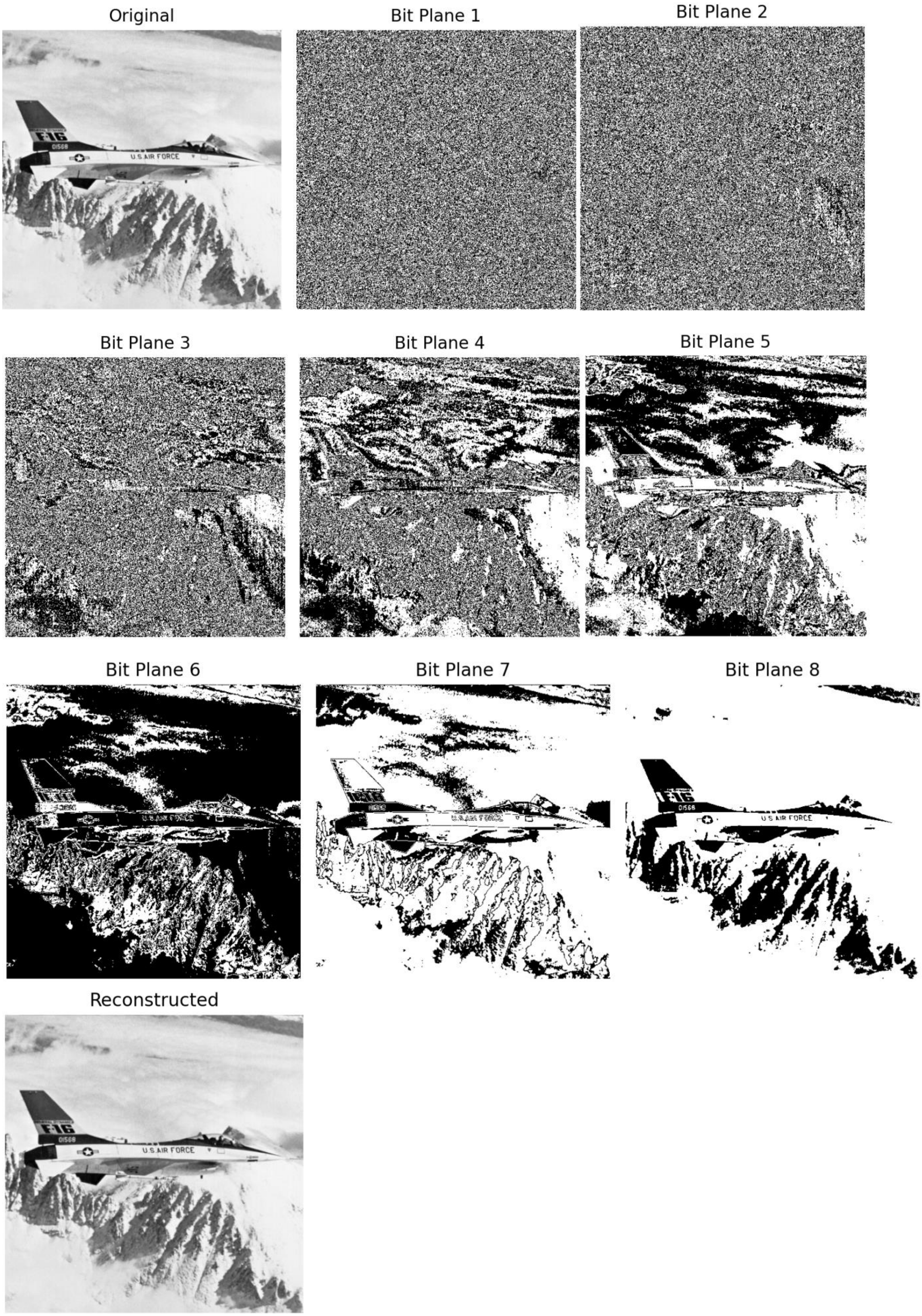
imageInterpolation.py

```python
import cv2
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('house.tif', cv2.IMREAD_GRAYSCALE)

# Downscale to 32x32
small = cv2.resize(img, (32, 32))

# Upscale using different interpolation methods
nearest = cv2.resize(small, (256, 256), interpolation=cv2.INTER_NEAREST)
linear = cv2.resize(small, (256, 256), interpolation=cv2.INTER_LINEAR)
cubic = cv2.resize(small, (256, 256), interpolation=cv2.INTER_CUBIC)


# Display results
plt.figure(figsize=(15,4))
plt.subplot(1,5,1)
plt.title('Original')
plt.imshow(img, cmap='gray')
plt.axis('off')

plt.subplot(1,5,2)
plt.title('Downscale to 32x32')
plt.imshow(small, cmap='gray')
plt.axis('off')

plt.subplot(1,5,3)
plt.title('Nearest Neighbor')
plt.imshow(nearest, cmap='gray')
plt.axis('off')

plt.subplot(1,5,4)
plt.title('Bilinear')
plt.imshow(linear, cmap='gray')
plt.axis('off')

plt.subplot(1,5,5)
plt.title('Bicubic')
plt.imshow(cubic, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

contrastStreching.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the grayscale image
img = cv2.imread('lena_gray_512.tif', cv2.IMREAD_GRAYSCALE)

# Negative transformation
negative = 255 - img

# Logarithmic transformation
c_log = 1
log_trans = c_log * np.log(1 + img.astype(np.float32))
log_trans = np.array(log_trans, dtype=np.uint8)

# Gamma transformation (gamma = 2.2)
gamma = 2.2
c_gamma = 255 / (np.max(img) ** gamma)
gamma_trans = c_gamma * (img.astype(np.float32) ** gamma)
gamma_trans = np.array(gamma_trans, dtype=np.uint8)

# Piecewise linear transformation
r1, s1 = 70, 100
r2, s2 = 140, 180
piecewise = np.zeros_like(img)

# Apply piecewise linear mapping
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        r = img[i, j]
        if r < r1:
            piecewise[i, j] = s1
        elif r < r2:
            piecewise[i, j] = ((s2 - s1) / (r2 - r1)) * (r - r1) + s1
        else:
            piecewise[i, j] = s2
piecewise = piecewise.astype(np.uint8)

# Plotting all transformations
fig, axes = plt.subplots(1, 5, figsize=(18, 4))
axes[0].set_title('Original')
axes[0].imshow(img, cmap='gray')
axes[0].axis('off')

axes[1].set_title('Negative')
axes[1].imshow(negative, cmap='gray')
axes[1].axis('off')

axes[2].set_title('Logarithmic')
axes[2].imshow(log_trans, cmap='gray')
axes[2].axis('off')
```

```python

axes[3].set_title('Gamma (2.2)')
axes[3].imshow(gamma_trans, cmap='gray')
axes[3].axis('off')

axes[4].set_title('Piecewise Linear')
axes[4].imshow(piecewise, cmap='gray')
axes[4].axis('off')

plt.tight_layout()
plt.show()
```


Original
Bit Plane 1
Bit Plane 2
Bit Plane 3
Bit Plane 4
Bit Plane 5
Bit Plane 6
Bit Plane 7
Bit Plane 8
Reconstructed

bitwise.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read image in grayscale
img = cv2.imread('jetplane.tif', cv2.IMREAD_GRAYSCALE)

# Prepare a list
bit_planes = []
for i in range(8):
    # Extract bit planes
    plane = np.bitwise_and(img, 1 << i) >> i
    bit_planes.append(plane)

# Reconstruct image
reconstructed = np.zeros_like(img, dtype=np.uint8)
for i in range(8):
    reconstructed += (bit_planes[i] << i)

# Plot original and 8 planes
fig, axes = plt.subplots(2, 5, figsize=(15, 6))

# Show original image
axes[0, 0].imshow(img, cmap='gray')
axes[0, 0].set_title('Original')
axes[0, 0].axis('off')

# Show bit planes
for i in range(8):
    row = 0 if i < 4 else 1
    col = i + 1 if i < 4 else i - 3
    axes[row, col].imshow(bit_planes[i], cmap='gray')
    axes[row, col].set_title(f'Bit Plane {i+1}')
    axes[row, col].axis('off')


axes[1, 0].imshow(reconstructed, cmap='gray')
axes[1, 0].set_title('Reconstructed')
axes[1, 0].axis('off')
plt.tight_layout()
plt.show()
```