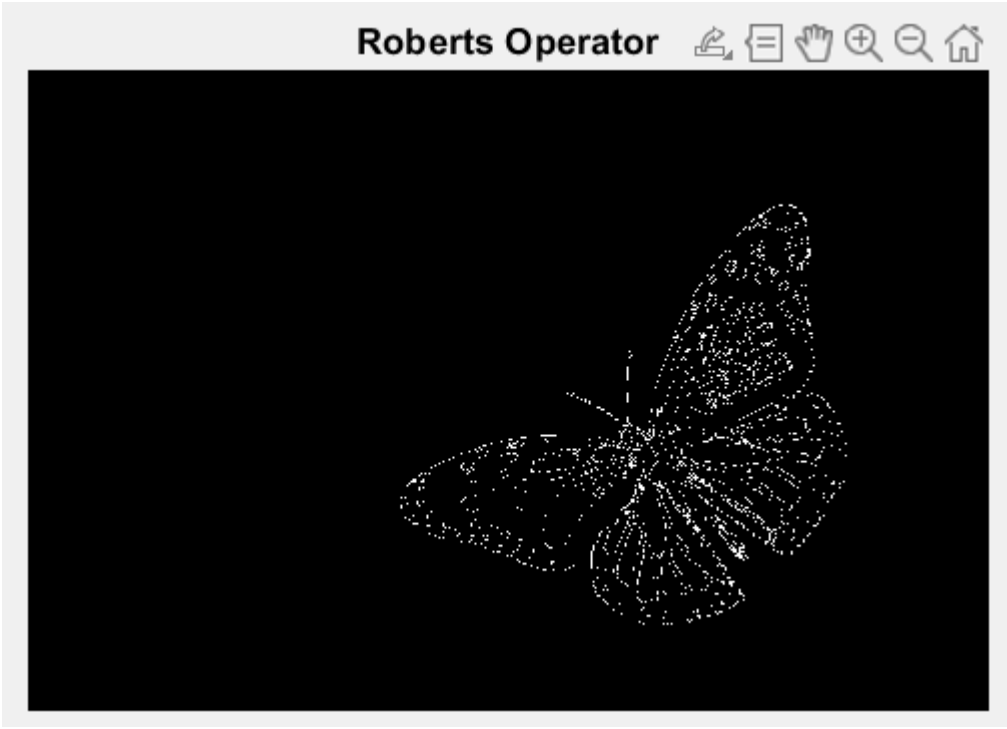
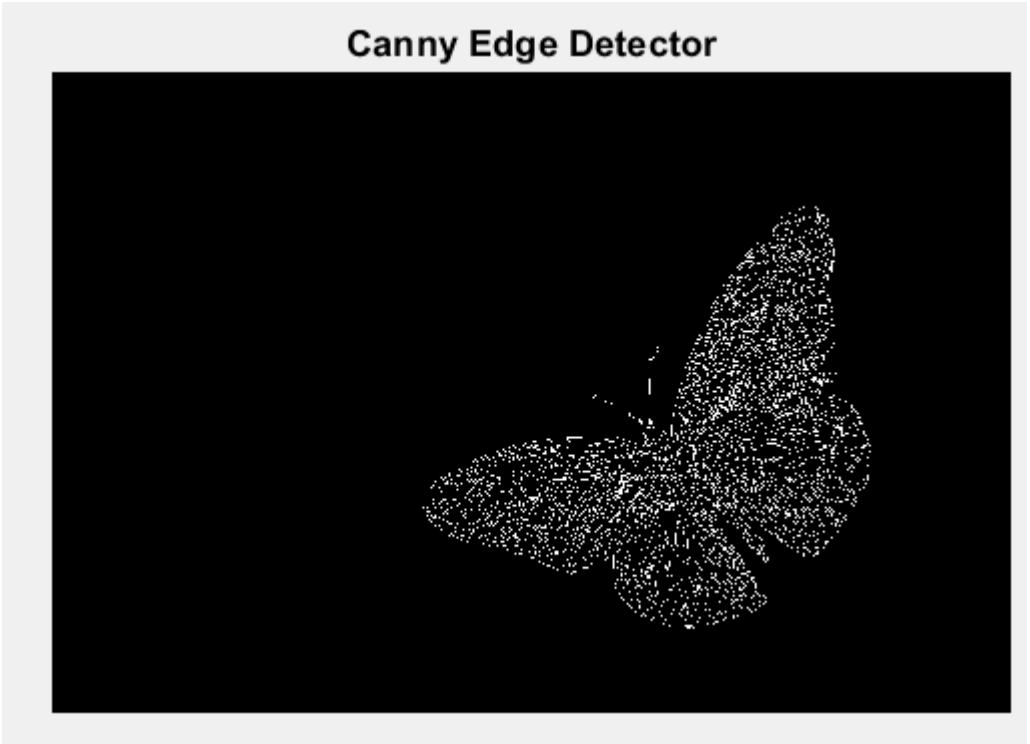
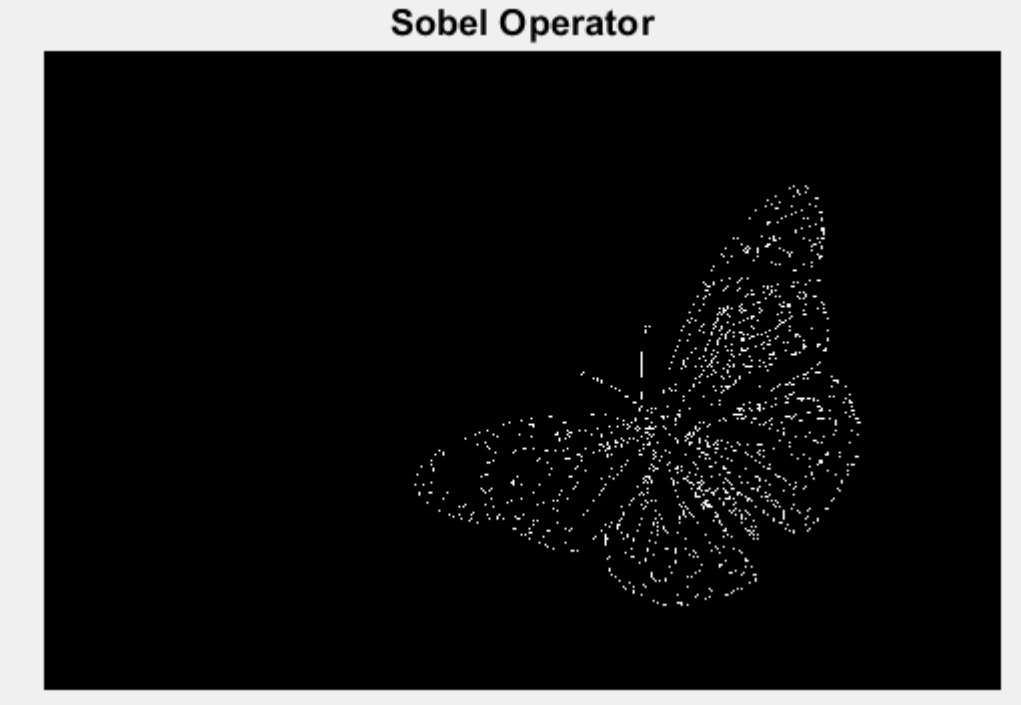
Edge Detection (10)

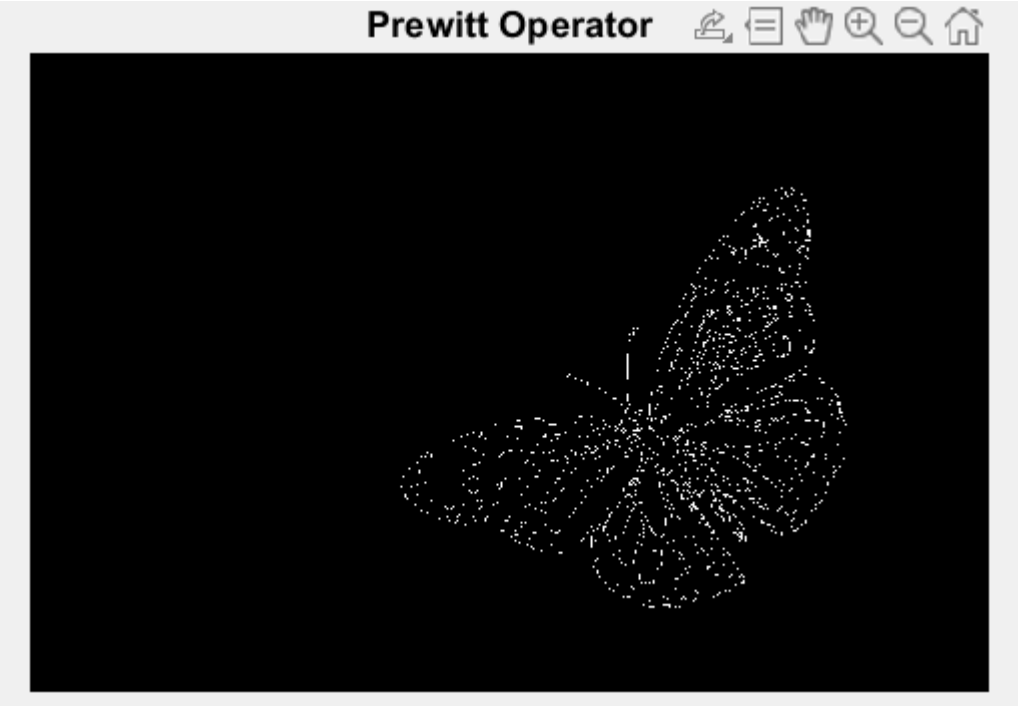
1) Use the `edge` function to generate results for Roberts, Canny, Sobel, and Prewitt operators on an image of your choice

```
% Load an image
img = imread('Butterfly 1.jpg');

% Convert the image to grayscale
img = rgb2gray(img);

% Apply Roberts operator
roberts_edge = edge(img, 'Roberts');

% Apply Canny edge detector
canny_edge = edge(img, 'Canny');

% Apply Prewitt operator
prewitt_edge = edge(img, 'Prewitt');

% Apply Sobel operator
sobel_edge = edge(img, 'Sobel');

% Display the results
subplot(2, 2, 1), imshow(roberts_edge), title('Roberts Operator');
subplot(2, 2, 2), imshow(canny_edge), title('Canny Edge Detector');
subplot(2, 2, 3), imshow(sobel_edge), title('Sobel Operator');
subplot(2, 2, 4), imshow(prewitt_edge), title('Prewitt Operator');
```


Roberts Operator


Canny Edge Detector


Sobel Operator


Prewitt Operator

Canny edge detector provided you with the best results. Canny is known for its robust edge detection capabilities, especially when it comes to reducing noise and accurately localizing edges.

Sobel's ability to provide detailed directional information can be very useful, especially understanding the orientation of edges is essential.

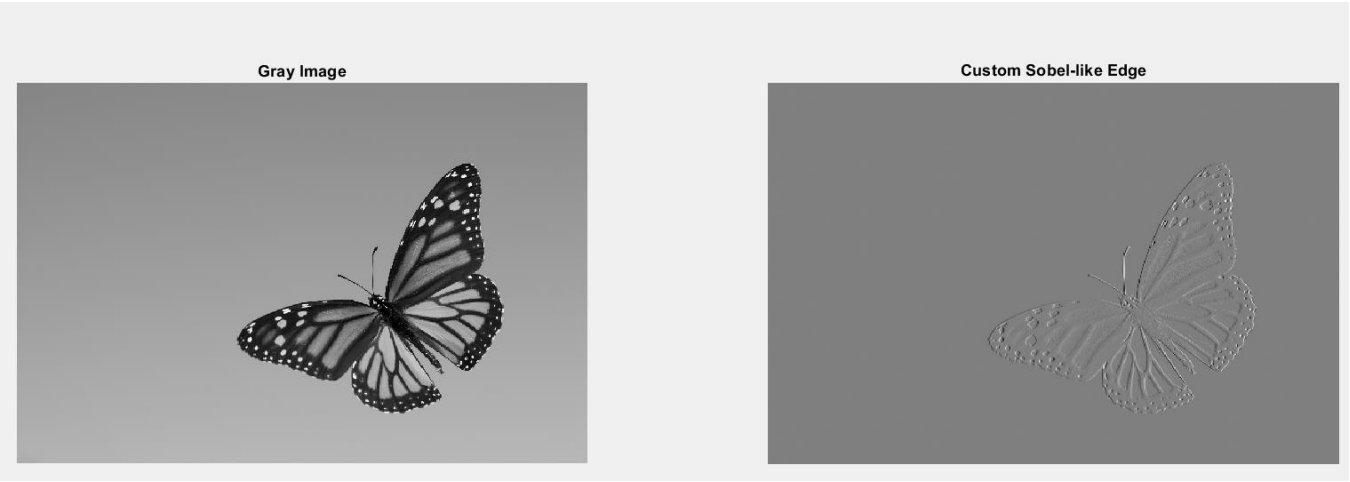
The combination of both Canny and Sobel operators seems to have worked well for my butterfly image, balancing accuracy and detail.

2)

Edge Filter (10)

Design a 7x7 “Sobel” operator and filter your image from the “Edge Detection” task above with your filter. The main idea behind the design of a proper Sobel-ish operator is to model the Gaussian derivate in one direction and the Gaussian in the perpendicular direction:

$$\frac{\partial G(x,y,\sigma)}{\partial x} = -\frac{x}{2\pi\sigma^4}e^{-\frac{\left(x^2+y^2\right)}{2\sigma^2}}$$

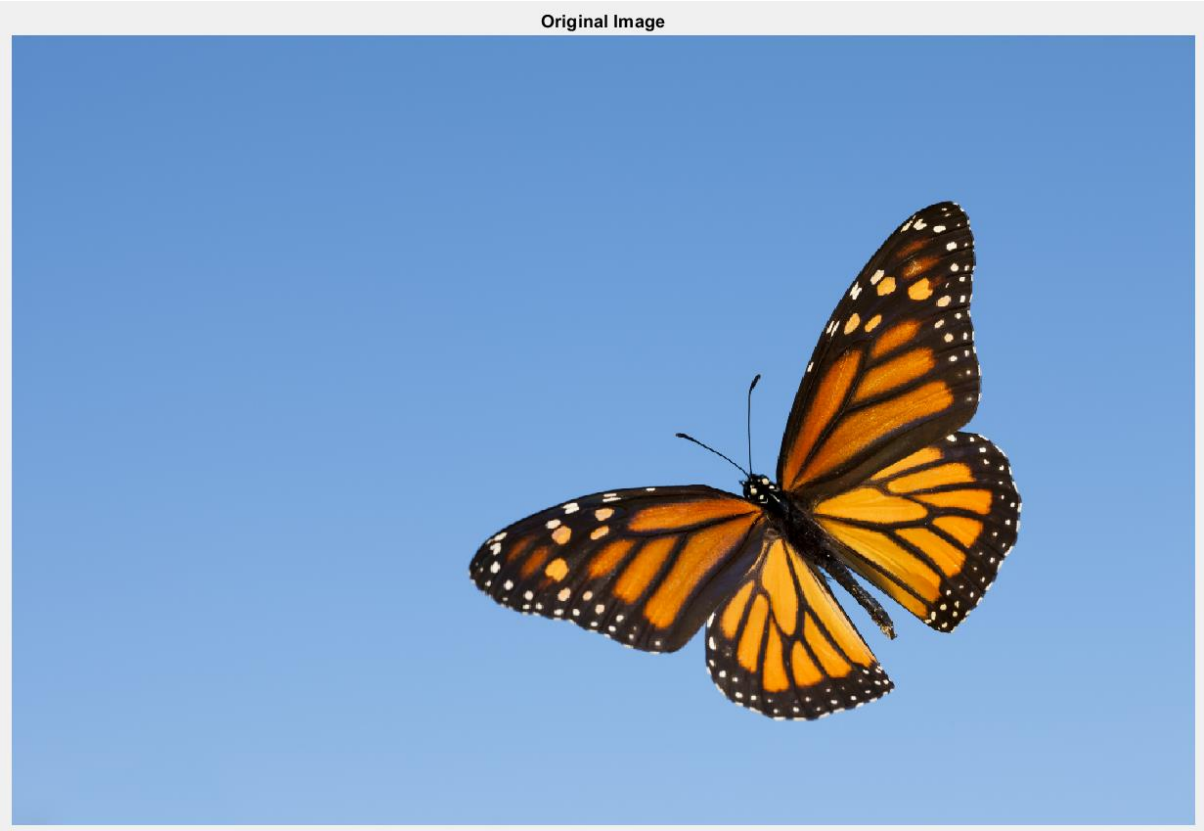



```
grayImage = imread('Butterfly 1.jpg');
grayImage = rgb2gray(grayImage); % Convert to grayscale

% Define the filter parameters
sigma = 1;
fltrS = 7;

% Create the custom Sobel like filter
[x, y] = meshgrid(-(fltrS-1)/2:(fltrS-1)/2, -(fltrS-1)/2:(fltrS-1)/2);
custSobel = (-x./(2*pi*sigma^4)) .* exp(-((x.^2 + y.^2) / (2*sigma^2)));
custSobel = custSobel - mean(custSobel(:));

% Apply the custom Sobel like filter to the image
customSobelEdge = conv2(double(grayImage), custSobel, 'same');

% Display the original image and the custom Sobel like edge
figure;
subplot(1, 2, 1);
imshow(grayImage);
```

```
title('Gray Image');


subplot(1, 2, 2);
imshow(customSobelEdge, []);
title('Custom Sobel-like Edge');
```

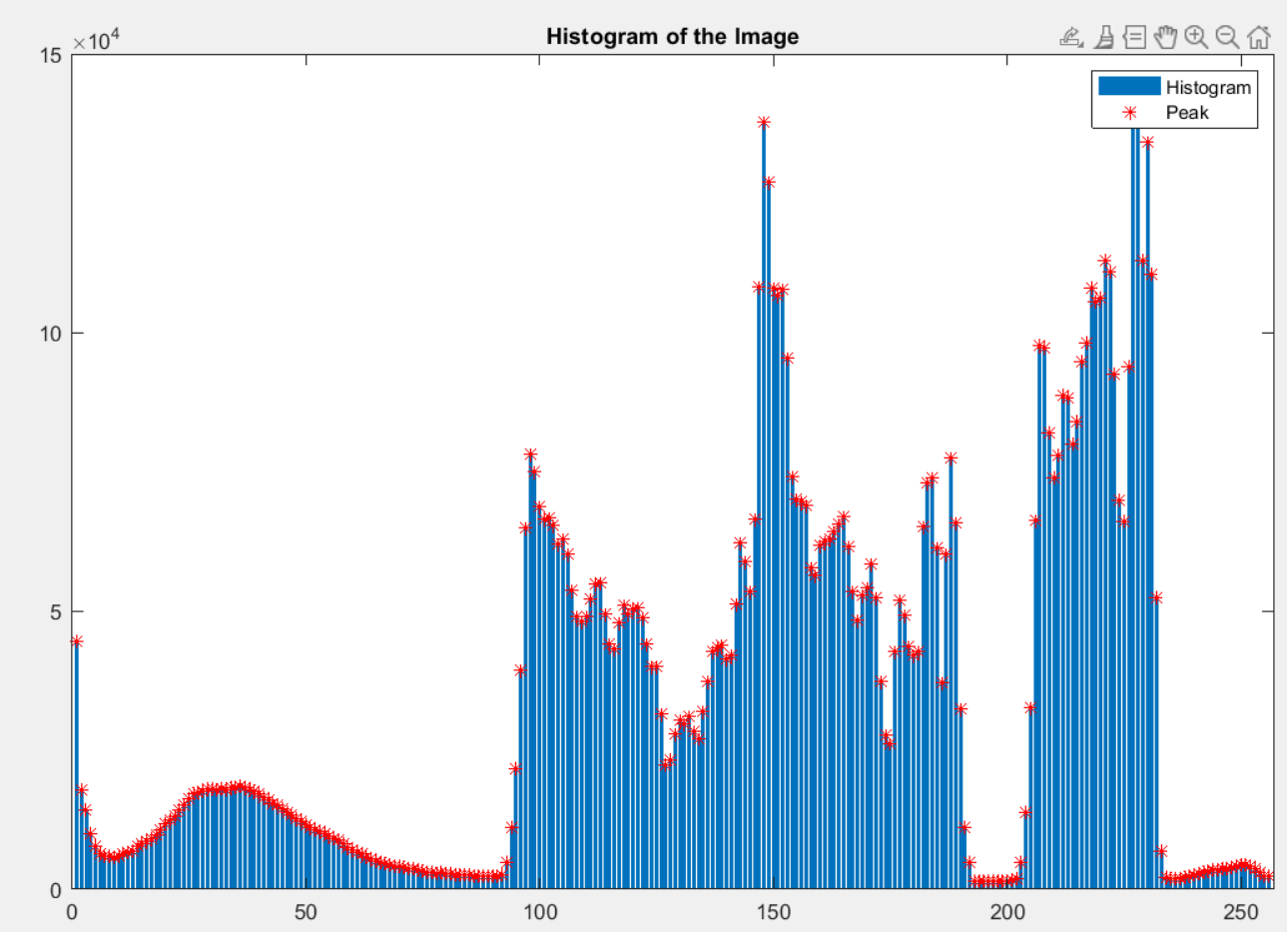
3)

a) Show your image

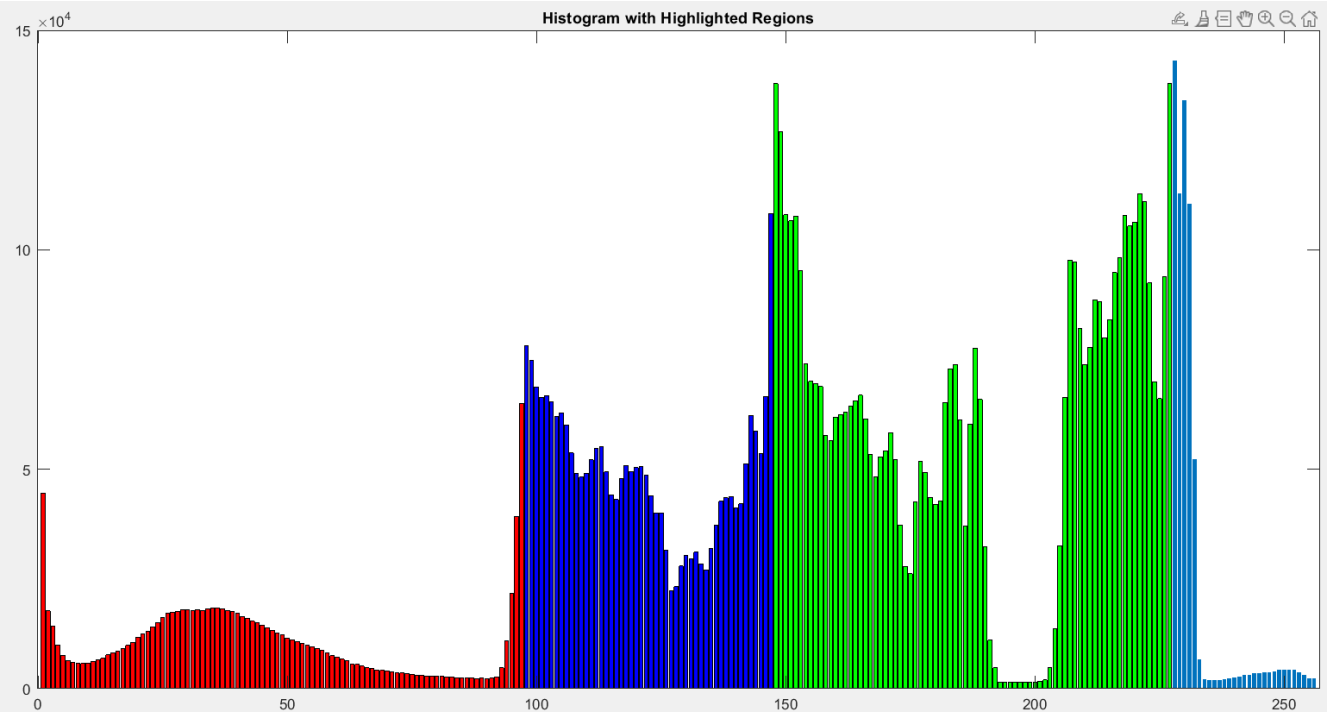


b)

Display the histogram and identify the peaks of your histogram with the “objects” that they correspond to



c)




```
Object Ranges:
     1    98

    98   148

   148   227
```

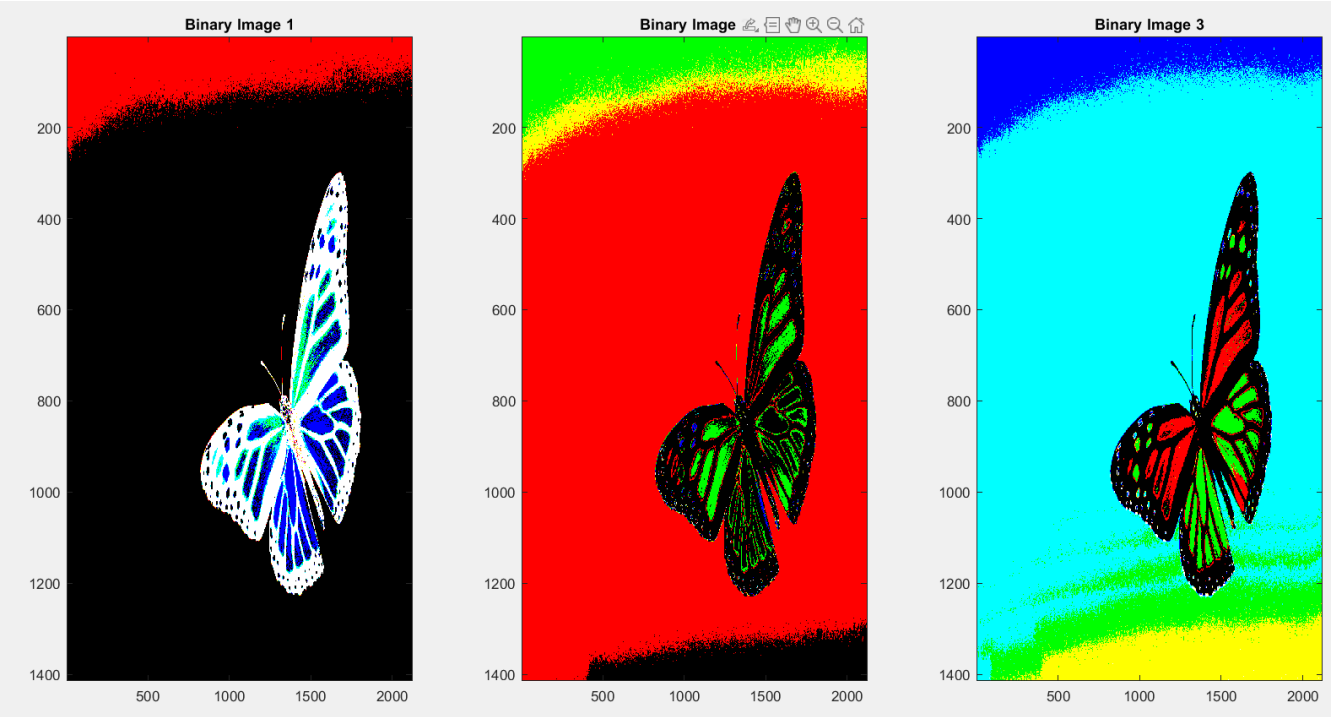
```
% Load and display the original image
img = imread('your_image.jpg');
imshow(img);
title('Original Image');

% Compute and display the histogram
histogram = imhist(img);
```

```
% Identify peaks in the histogram (you may need to adjust the threshold)
threshold = 100; % Adjust the threshold to detect peaks
peaks = find(histogram > threshold);

% Manually specify the ranges for the highlighted regions in the histogram
highlighted_region1 = [1, 98];   % Range for the first object (1 to 98)
highlighted_region2 = [98, 148];  % Range for the second object (98 to 148)
highlighted_region3 = [148, 227]; % Range for the third object (148 to 227)

% Create a histogram with the specified regions highlighted
figure;
bar(histogram); % Plot the full histogram
hold on;
bar(highlighted_region1(1):highlighted_region1(2), histogram(highlighted_region1(1):highlighted_region1(2)), 'r'); % Plot the first highlighted region in red
bar(highlighted_region2(1):highlighted_region2(2), histogram(highlighted_region2(1):highlighted_region2(2)), 'b'); % Plot the second highlighted region in blue
bar(highlighted_region3(1):highlighted_region3(2), histogram(highlighted_region3(1):highlighted_region3(2)), 'g'); % Plot the third highlighted region in green
title('Histogram with Highlighted Regions');
hold off;

% Display peak values and specified object ranges
disp('Peak Values:');
disp(histogram(peaks));

disp('Object Ranges:');
disp(highlighted_region1);
```

disp(highlighted_region2);

disp(highlighted_region3);

D)



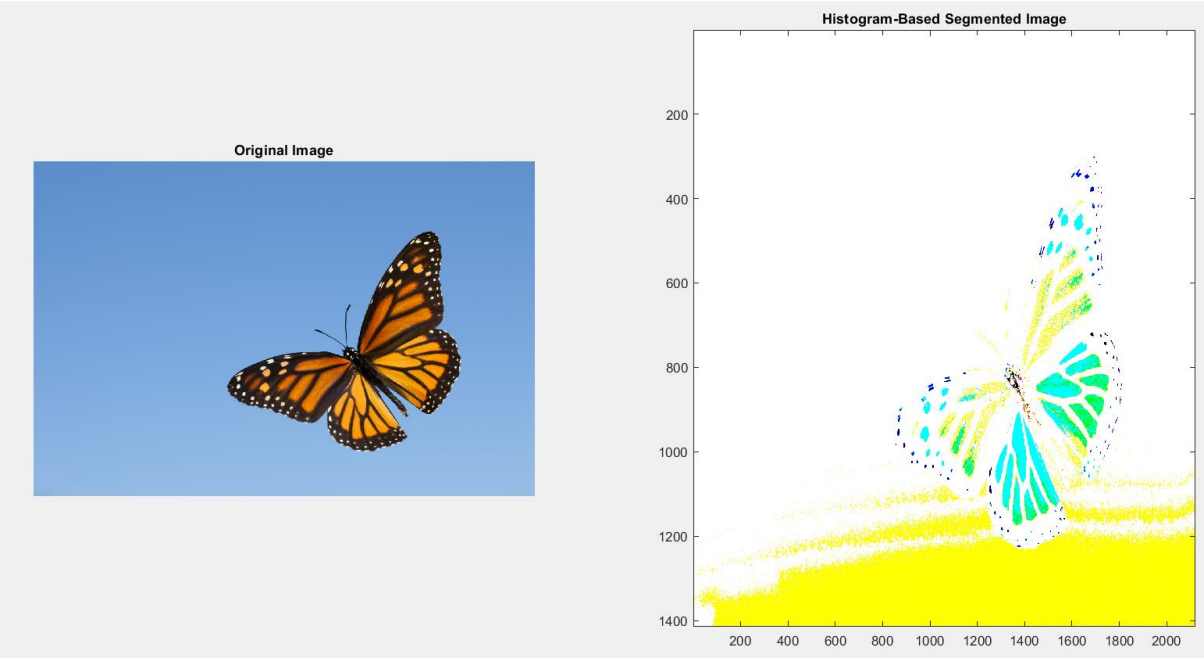
% Load and display the original image

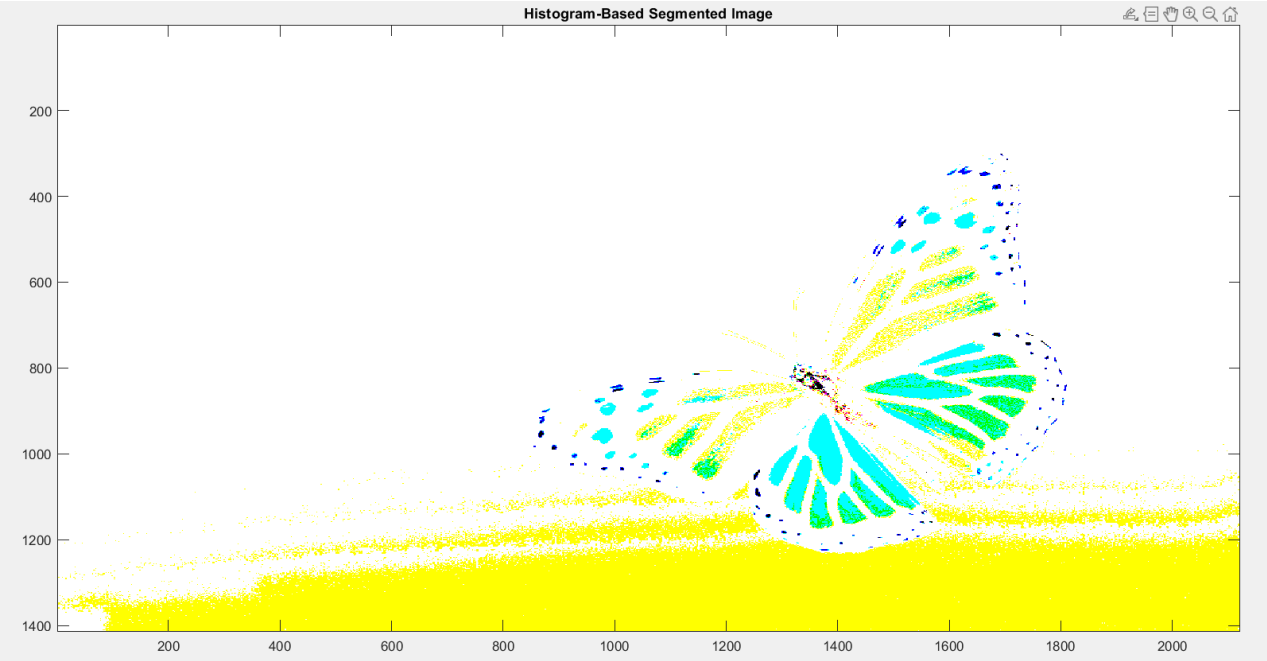
img = imread('Butterfly 1.jpg');

```
imshow(img);
title('Original Image');

% Compute and display the histogram
histogram = imhist(img);

% Identify peaks in the histogram (you may need to adjust the threshold)
threshold = 100; % Adjust the threshold to detect peaks
peaks = find(histogram > threshold);

% Manually specify the ranges for the highlighted regions in the histogram
highlighted_region1 = [1, 98];   % Range for the first object (1 to 98)
highlighted_region2 = [98, 148];  % Range for the second object (98 to 148)
highlighted_region3 = [148, 227]; % Range for the third object (148 to 227)
% Create binary images for each highlighted region
binary_image1 = img >= highlighted_region1(1) & img <= highlighted_region1(2);
binary_image2 = img >= highlighted_region2(1) & img <= highlighted_region2(2);
binary_image3 = img >= highlighted_region3(1) & img <= highlighted_region3(2);

% Display the binary images with proper scaling
figure;

% Scale the binary images for better visibility using imagesc
subplot(1, 3, 1);
imagesc(binary_image1);
colormap gray;
title('Binary Image 1');

subplot(1, 3, 2);
```

```
imagesc(binary_image2);
colormap gray;
title('Binary Image 2');

subplot(1, 3, 3);
imagesc(binary_image3);
colormap gray;
title('Binary Image 3');
```

a) e) Finally construct the histogram-based segmented image, by combining the binary images.






```
img = imread('Butterfly 1.jpg');
imshow(img);


% Compute and display the histogram
histogram = imhist(img);

% Identify peaks in the histogram (you may need to adjust the threshold)
threshold = 100; % Adjust the threshold to detect peaks
peaks = find(histogram > threshold);

% Manually specify the ranges for the highlighted regions in the histogram
highlighted_region1 = [1, 98];   % Range for the first object (1 to 98)
highlighted_region2 = [98, 148];  % Range for the second object (98 to 148)
highlighted_region3 = [148, 227]; % Range for the third object (148 to 227)
```

```
% Create a histogram with the specified regions highlighted

figure;

bar(histogram); % Plot the full histogram

hold on;

bar(highlighted_region1(1):highlighted_region1(2),
histogram(highlighted_region1(1):highlighted_region1(2)), 'r'); % Plot the first highlighted
region in red

bar(highlighted_region2(1):highlighted_region2(2),
histogram(highlighted_region2(1):highlighted_region2(2)), 'b'); % Plot the second
highlighted region in blue

bar(highlighted_region3(1):highlighted_region3(2),
histogram(highlighted_region3(1):highlighted_region3(2)), 'g'); % Plot the third highlighted
region in green

title('Histogram with Highlighted Regions');

hold off;


% Display peak values and specified object ranges

disp('Peak Values:');

disp(histogram(peaks));


disp('Object Ranges:');

disp(highlighted_region1);

disp(highlighted_region2);

disp(highlighted_region3);
```

e) e) Finally construct the histogram-based segmented image, by combining the binary images.

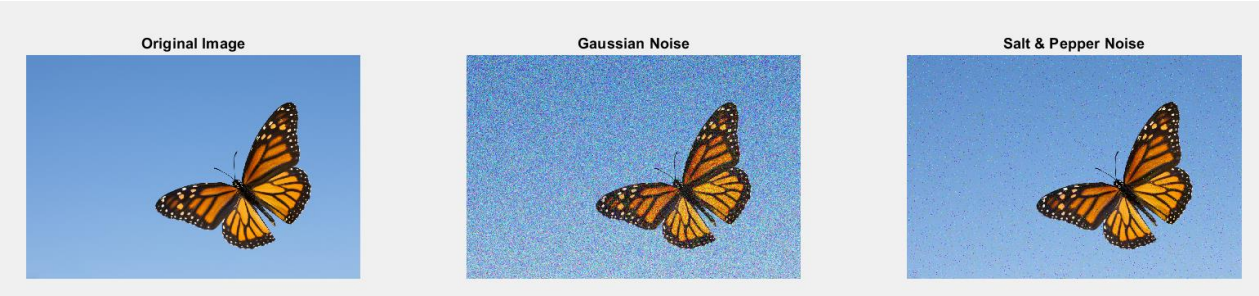
```
% Load and display the original image
```

```
img = imread('Butterfly 1.jpg');

% Compute and display the histogram
histogram = imhist(img);

% Identify peaks in the histogram (you may need to adjust the threshold)
threshold = 100; % Adjust the threshold to detect peaks
peaks = find(histogram > threshold);

% Manually specify the ranges for the highlighted regions in the histogram
highlighted_region1 = [1, 98];   % Range for the first object (1 to 98)
highlighted_region2 = [98, 148];  % Range for the second object (98 to 148)
highlighted_region3 = [148, 227]; % Range for the third object (148 to 227)

% Create binary images for each highlighted region
binary_image1 = img >= highlighted_region1(1) & img <= highlighted_region1(2);
binary_image2 = img >= highlighted_region2(1) & img <= highlighted_region2(2);
binary_image3 = img >= highlighted_region3(1) & img <= highlighted_region3(2);

% Merge the binary images with logical OR operation
segmented_image = binary_image1 | binary_image2 | binary_image3;

% Resize the segmented image to match the original image size
segmented_image = imresize(segmented_image, size(img));

% Display the original image and the segmented image side by side
figure;

subplot(1, 2, 1);
```

```
imshow(img);
title('Original Image');

subplot(1, 2, 2);
imshow(segmented_image);
title('Histogram-Based Segmented Image');
```

4)

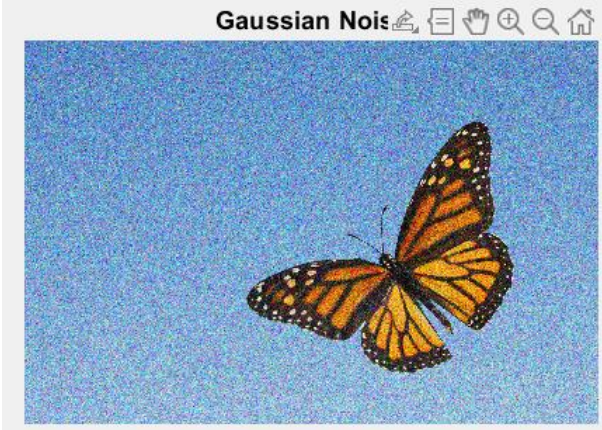
a)

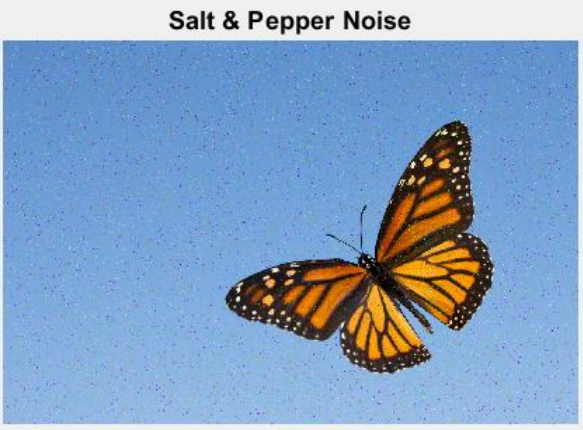



```
% Load an image of your choice
originalImage = imread('your_image.jpg');

% Generate a Gaussian noise-corrupted image
noise_gaussian = imnoise(originalImage, 'gaussian', 0, 0.05);
```

```
% Generate a salt and pepper noise-corrupted image
noise_saltAndpepper = imnoise(originalImage, 'salt & pepper', 0.02);

% Display the images:
% 1. Original Image
% 2. Gaussian Noise
% 3. Salt & Pepper Noise

figure;

subplot(1, 3, 1);
imshow(originalImage);
title('Original Image');

subplot(1, 3, 2);
imshow(noise_gaussian);
title('Gaussian Noise');

subplot(1, 3, 3);
imshow(noise_saltAndpepper);
title('Salt & Pepper Noise');
```

b) Use `subplot` to display the original image and the two noise corrupted images

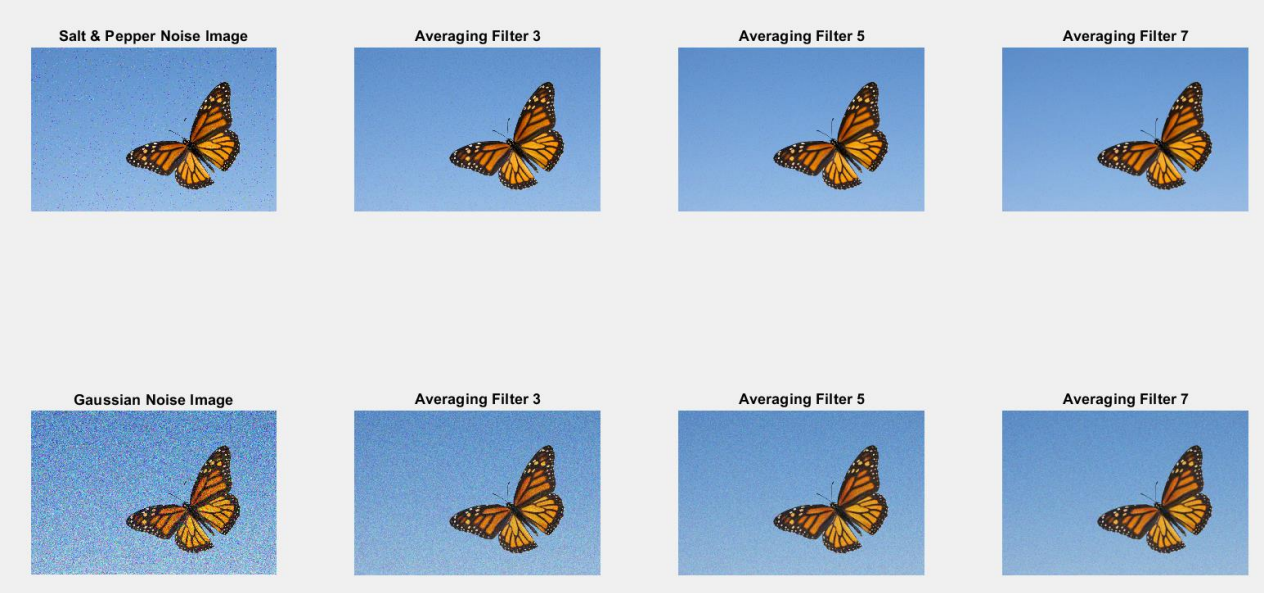





```
% Load an image of your choice
originalImage = imread('Butterfly 1.jpg');

% Generate a Gaussian noise-corrupted image
noise_gaussian = imnoise(originalImage, 'gaussian', 0, 0.05);

% Generate a salt and pepper noise-corrupted image
noise_saltAndpepper = imnoise(originalImage, 'salt & pepper', 0.02);

% Display the images using subplot:
% Original Image, Gaussian Noise, Salt & Pepper Noise

figure;

subplot(1, 3, 1);
imshow(originalImage);
title('Original Image');

subplot(1, 3, 2);
```

```
imshow(noise_gaussian);
title('Gaussian Noise');

subplot(1, 3, 3);
imshow(noise_saltAndpepper);
title('Salt & Pepper Noise');
```

c) Use the function `fspecial` to design averaging filters of size (3x3), (5,5), and (7x7). Use `subplot` to display the `noise_saltAndpepper` image and the three averaged filtered results. Do the same for the `noise_gaussian` image.

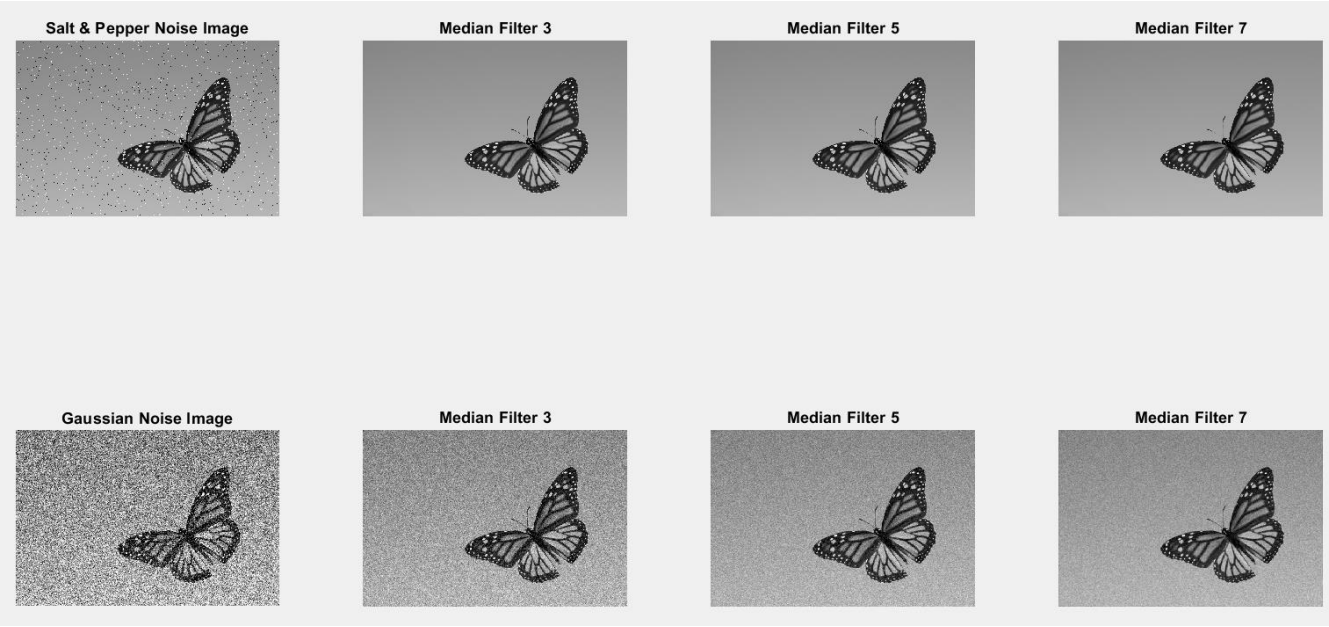



```
% Load an image of your choice (if not already loaded)
originalImage = imread('your_image.jpg');

% Generate a Gaussian noise-corrupted image
noise_gaussian = imnoise(originalImage, 'gaussian', 0, 0.05);
```

```
% Generate a salt and pepper noise-corrupted image
noise_saltAndpepper = imnoise(originalImage, 'salt & pepper', 0.02);

% Define filter sizes
filter_sizes = [3, 5, 7];

% Create a figure for salt and pepper noise image and filtered results
figure;
subplot(2, 4, 1);
imshow(noise_saltAndpepper);
title('Salt & Pepper Noise Image');

% Create a figure for Gaussian noise image and filtered results
subplot(2, 4, 5);
imshow(noise_gaussian);
title('Gaussian Noise Image');

% Loop through filter sizes
for i = 1:numel(filter_sizes)
    % Create an averaging filter of the specified size
    filter = fspecial('average', filter_sizes(i));

    % Apply the filter to the salt and pepper noise image
    filtered_image = imfilter(noise_saltAndpepper, filter, 'replicate');

    % Display the filtered image
    subplot(2, 4, i + 1);
    imshow(filtered_image);
    title(['Averaging Filter ' num2str(filter_sizes(i))]);

    % Apply the filter to the Gaussian noise image
    filtered_image = imfilter(noise_gaussian, filter, 'replicate');
```

```
    % Display the filtered image
    subplot(2, 4, i + 5);
    imshow(filtered_image);
    title(['Averaging Filter ' num2str(filter_sizes(i))]);
end
```

d)




```
% Load an image of your choice (if not already loaded)
originalImage = imread('Butterfly 1.jpg');

% Convert the original image to grayscale
originalImage = rgb2gray(originalImage);

% Generate a Gaussian noise-corrupted image
```

```
noise_gaussian = imnoise(originalImage, 'gaussian', 0, 0.05);

% Generate a salt and pepper noise-corrupted image
noise_saltAndpepper = imnoise(originalImage, 'salt & pepper', 0.02);

% Define window sizes for median filtering
window_sizes = [3, 5, 7];

% Create a figure for salt and pepper noise image and filtered results
figure;
subplot(2, 4, 1);
imshow(noise_saltAndpepper);
title('Salt & Pepper Noise Image');

% Create a figure for Gaussian noise image and filtered results
subplot(2, 4, 5);
imshow(noise_gaussian);
title('Gaussian Noise Image');

% Loop through window sizes for median filtering
for i = 1:numel(window_sizes)
    % Perform median filtering on the salt and pepper noise image
    filtered_image_saltAndPepper = medfilt2(noise_saltAndpepper, [window_sizes(i), window_sizes(i)]);

    % Display the filtered image
    subplot(2, 4, i + 1);
    imshow(filtered_image_saltAndPepper);
    title(['Median Filter ' num2str(window_sizes(i))]);

    % Perform median filtering on the Gaussian noise image
    filtered_image_gaussian = medfilt2(noise_gaussian, [window_sizes(i), window_sizes(i)]);

    % Display the filtered image
```

```
    subplot(2, 4, i + 5);
    imshow(filtered_image_gaussian);
    title(['Median Filter ' num2str(window_sizes(i))]);
end
```