Q1. Write a program to load a .csv file as a NumPy 1-D array. Find the maximum and minimum elements in the array.

```python
import numpy as np


data = np.loadtxt("book1.csv", delimiter="\t", skiprows=1,
usecols=(1))  # load csv

data = data.astype(int)
max_point = np.max(data)  # numpy methods
min_point = np.min(data)

print("Maximum point in the array :", max_point)
print("Minimum point in the array :", min_point)
```

Q2. For the Numpy 1-D array as obtained in Q.1, sort the elements in ascending order.

```python
import numpy as np


data = np.loadtxt("book1.csv", delimiter="\t", skiprows=1,
usecols=(1))

data = data.astype(int)

data.sort()  # sort method
print(data)
```

Q3. For the sorted Numpy 1-D array as obtained in Q.2, reverse the array and print

```python
import numpy as np


data = np.loadtxt("book1.csv", delimiter="\t", skiprows=1,
usecols=(1))

data = data.astype(int)

data.sort()
ascending_data = np.flip(data, axis=0)  # flips list
print(ascending_data)
```

Q4. Write a program to load three .csv files (Book1.csv, Book2.csv, and Book3.csv) as a list of Numpy 1-D arrays. Print the means of all arrays as a list.

```python
import numpy as np

data1 = np.loadtxt("book1.csv", delimiter="\t", skiprows=1,
usecols=(1))
```

```
data2 = np.loadtxt("book2.csv", delimiter="\t", skiprows=1,
usecols=(1))
data3 = np.loadtxt("book3.csv", delimiter="\t", skiprows=1,
usecols=(1))

data = [data1, data2, data3]
means = []
for d in data:  # list comprehension
    means.append(np.mean(d))  # mean method

print(f"Means of the datasets 1, 2, 3 respectively are: {means}")
```

Q5. Write a program to read an image, store the image in NumPy 3-D array. For the image, consider a.png. Display the image. Let the image stored in the NumPy array be X.

```
import cv2
import matplotlib.pyplot as plt

image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.imshow(X)
plt.title("Image loaded and stored in NumPy array")
plt.show()
```

Q6. Write a program to convert a color image (say a.PNG) into a grescale image. Let the greysacle image stored in the Numpy 2-D array be X. Display the grayscale iamge on the screen.

```
import cv2
import matplotlib.pyplot as plt

image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  ## change color space
X = cv2.cvtColor(X, cv2.COLOR_BGR2GRAY)
```

```python
X = X.astype("uint8")

plt.imshow(X, cmap="gray")
plt.title("Image loaded and stored in NumPy array")
plt.show()
```

Q7. Let Y be the transpose matrix of X. Write a program to obtain Z = X×Y.

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

image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
X = cv2.cvtColor(X, cv2.COLOR_BGR2GRAY)
X = X.astype("uint8")

X_MATRIX = np.array(X)
Z = np.dot(X_MATRIX.tolist(), X_MATRIX.T.tolist())  # numpy method to
calculate product

print(f"Z : {Z}")
```

Q8. For the problem in Q. 7, write your program without using NumPy library. Compare the computation times doing the same with NumPy and basic programming in Python.

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


image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
X = X.astype("uint8")
```

```python
X_MATRIX = np.array(X)

start_time = time.time()
Z = np.dot(X_MATRIX.tolist(), X_MATRIX.T.tolist())
end_time = time.time()

print(f"Z : {Z}")

print(f"[Numpy TIME ] : {end_time - start_time} secs")

print("======================================================================
==========")


def matrix_multiplication(X, Y):
    """
    Performs matrix multiplication using nested loops.

    Args:
      X: A 2D list representing the first matrix.
      Y: A 2D list representing the second matrix.

    Returns:
      A 2D list representing the product matrix Z.
    """
    m, n = len(X), len(Y[0])  # Get dimensions of X and Y
    Z = [[0 for _ in range(n)] for _ in range(m)]  # Initialize result
matrix

    for i in range(m):
        for j in range(n):
            for k in range(len(Y)):
                Z[i][j] += X[i][k] * Y[k][j]

    return Z


X = X_MATRIX.tolist()
Y = X_MATRIX.T.tolist()

# Assuming X is a 2D list
Y = [[row[i] for row in X] for i in range(len(X[0]))]  # Transpose X
manually


start_time = time.time()
Z = matrix_multiplication(X, Y)
end_time = time.time()
# Print the result
print(Z)


print(f"[NORMAL TIME] : {end_time - start_time} secs")
```

Q9. Plot the pixel intensity histogram of the grescale image stored in X.

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

# Assuming your grayscale image data is stored in the variable `X`

# Calculate the histogram directly from the image data


image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
X = cv2.cvtColor(X, cv2.COLOR_BGR2GRAY)
X = X.astype("uint8")

counts, bins = np.histogram(
    X.flatten(), bins=256
)  # Adjust the number of bins as needed

# Plot the histogram using matplotlib
plt.bar(bins[:-1], counts, width=bins[1] - bins[0])

# Customize the plot (optional)
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.title("Pixel Intensity Histogram")
plt.grid(True)

# Display the histogram
plt.show()
```

Q10. Create a black rectangle at the position [(40,100) top right, (70, 200) bottom left] in the grayscale image. Display the image.

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

```


# Assuming your grayscale image data is stored in the variable `X`

image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
X = cv2.cvtColor(X, cv2.COLOR_BGR2GRAY)
X = X.astype("uint8")

# Calculate rectangle coordinates based on provided points and image
dimensions
rect_top_left = (40, 100)
rect_bottom_right = (70, 200)

# Convert coordinates to array slicing indices
rect_left, rect_top = rect_top_left
rect_right, rect_bottom = rect_bottom_right

# Determine black color value based on data type and preference
black_value = 0

X[rect_top:rect_bottom, rect_left:rect_right] = black_value


# Display the modified image
plt.imshow(X, cmap="gray")  # Use grayscale colormap
plt.show()
```

Q11. Using the grayscale image stored in X, transform it into the binarized image with thresholds: [50, 70, 100, 150]. Let the binarized images are stored in Z50, Z70, Z100, and Z150, respectively

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

image_path = "a.png"

image = cv2.imread(image_path)

if image is None:
    print("[ERROR] IMAGE NOT FOUND", image_path)
    exit()

```

```
X = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
X = cv2.cvtColor(X, cv2.COLOR_BGR2GRAY)
X = X.astype("uint8")

X = np.array(X)

# Assuming X is a NumPy array with grayscale values
Z50 = np.copy(X)  # Create a copy to avoid modifying the original
image
Z70 = np.copy(X)
Z100 = np.copy(X)
Z150 = np.copy(X)

# Apply thresholding for each image and store the results
Z50[Z50 > 50] = 255
Z50[Z50 <= 50] = 0

Z70[Z70 > 70] = 255
Z70[Z70 <= 70] = 0

Z100[Z100 > 100] = 255
Z100[Z100 <= 100] = 0

Z150[Z150 > 150] = 255
Z150[Z150 <= 150] = 0

# Now Z50, Z70, Z100, and Z150 contain the binarized images with
respective thresholds, show each

plt.imshow(Z50, cmap="gray")
plt.title("Image loaded threshold as 50")
plt.show()

plt.imshow(Z70, cmap="gray")
plt.title("Image loaded threshold as 70")
plt.show()

plt.imshow(Z100, cmap="gray")
plt.title("Image loaded threshold as 100")
plt.show()

plt.imshow(Z150, cmap="gray")
plt.title("Image loaded threshold as 150")
plt.show()
```

Q12. Consider the color image stored in a.png. Create a filter of [[-1,-1,-1][0,0,0][1,1,1]], and multiply this filter to each pixel value in the image. Display the image after filtering.

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

# Replace "a.png" with the actual path to your image
image = cv2.imread("a.png")

# Check if image is loaded successfully
if image is None:
    print("Error: Failed to load image. Please check the file path and
try again.")
    exit()

filter = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])

# grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply the filter using convolution
filtered_image = cv2.filter2D(image, -1, filter)

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
filtered_image = cv2.cvtColor(filtered_image, cv2.COLOR_BGR2RGB)

plt.imshow(filtered_image)
plt.title("Filtered")
plt.show()
```