Problem 1 – Sampling, Quantization (14 points):

a) There are many considerations that go into deciding how many megapixels a camera should have or how large the resolution of a computer-generated image should be. These considerations can be categorized into software, hardware, and physical factors:

Software Factors:

- Image Quality: A higher number of megapixels improves image clarity and captures more details.
- Image Transfer and Sharing: High-resolution images require more bandwidth for transfer and may slow down sharing speeds.
- Rendering Complexity: High-resolution images demand sophisticated algorithms to handle complex lighting and color rendering accurately.

Hardware Factors:

- Processing Power: Capturing and rendering high-resolution images require more powerful processors.
- Storage Space: Higher resolutions result in larger file sizes, requiring more storage capacity.

Physical Factors:

- Weight and Size: Cameras with larger megapixel sensors may be bulkier and heavier due to increased sensor size.
- Heat Dissipation: High-resolution sensors produce more heat, requiring efficient cooling mechanisms to prevent overheating.

b) There are several factors to consider when deciding how strong the quantization of an image should be. Here are some of the key considerations:

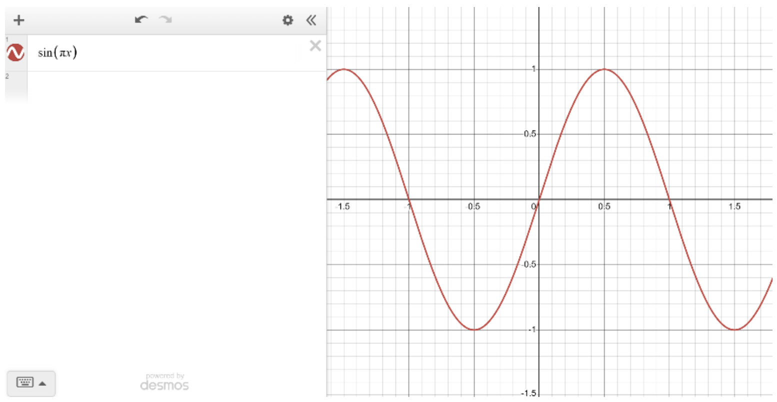
- **Color Depth**: Higher bits per pixel improve quality but require more storage, while older hardware supports lower color depths.
- **Memory Constraints**: Limited memory may require stronger quantization to reduce file size.
- **Display Technology**: Quantization is adjusted to match the capabilities of the display, as older screens support fewer colors.

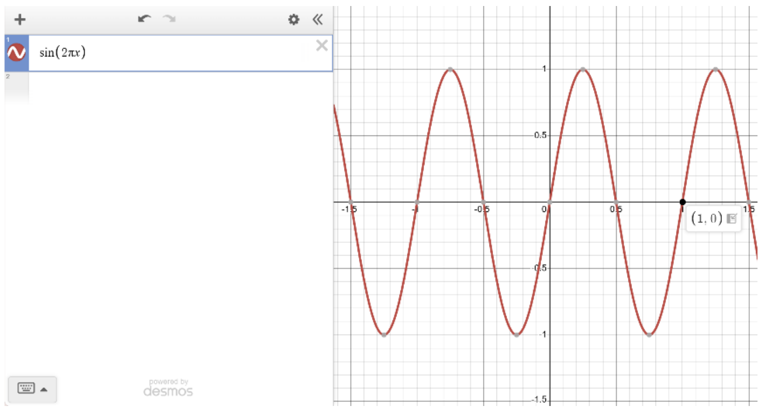
These factors balance image quality with hardware and storage limitations.

Problem 2 – Nyquist (16 points):

a) What is the wavelength of the sin image along the x-axis?

As we know, the periodic function $sin(\pi kx)$ completes half of its cycle over a unit length of 1.



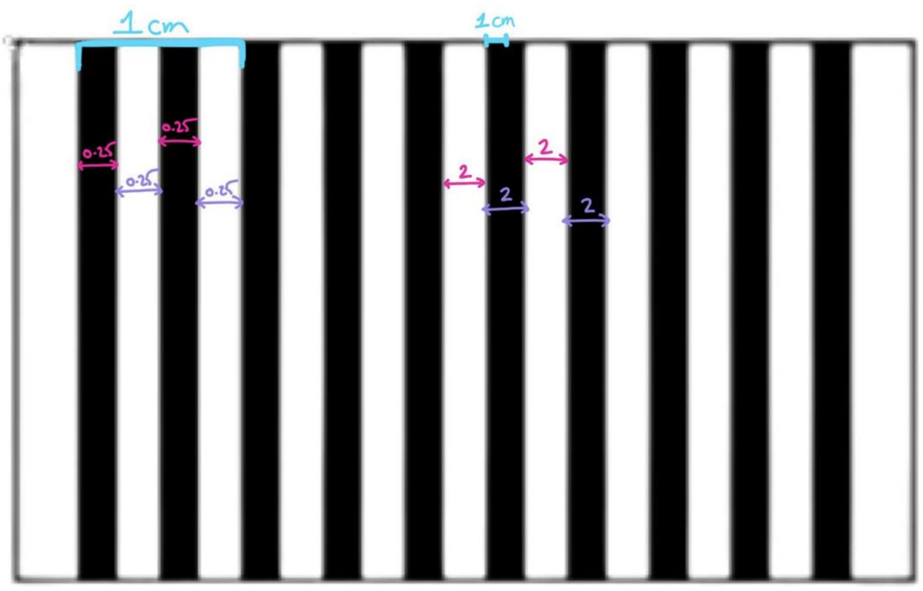


This means that the frequency of the wave along the x-axis is $\frac{k}{2}$.

From this, we can conclude that wavelength of the sin wave is $\lambda = \frac{2}{k}$.

The parameter A is not relevant to the wavelength calculation because $\lambda = \frac{2}{k}$ depends only on k. A affects the sampling process, not the wavelength itself.

b) What are the k values that will allow us to fully restore the sin image using the sampling grid given in the question?



To fully restore the sin image using the sampling grid ,the Nyquist Theorem must be satisfied. This theorem states that the sampling frequency must be at least twice the signal frequency.

As concluded in the previous section, the frequency of the image (sine wave) is $f_I = \frac{k}{2}$.

For A = 0.25:

$(2A = 0.5 \rightarrow \frac{1\,cm}{0.5} = 2)$

The frequency of the sampling grid is $f_{sample} = \frac{\#wave\ cycles}{unit} = \frac{2}{1} = 2$.

According to the Nyquist Theorem:

$$f_{sample} \geq 2 \cdot f_I$$

Substituting the values:

$$2 \geq 2 \cdot \frac{k}{2} \rightarrow 2 \geq k$$

This means we can fully restore the sine image for k values:

$$k \leq 2\,.$$

For A = 2:

$(2A \;=\; 4 \;\rightarrow \frac{1\,cm}{4} = 0.25)$

The frequency of the sampling grid is $f_{sample} = \frac{\#wave\;cycles}{unit} = \frac{0.25}{1} = 0.25$
.

According to the Nyquist Theorem:

$$f_{sample} \geq 2 \cdot f_I$$

Substituting the values:

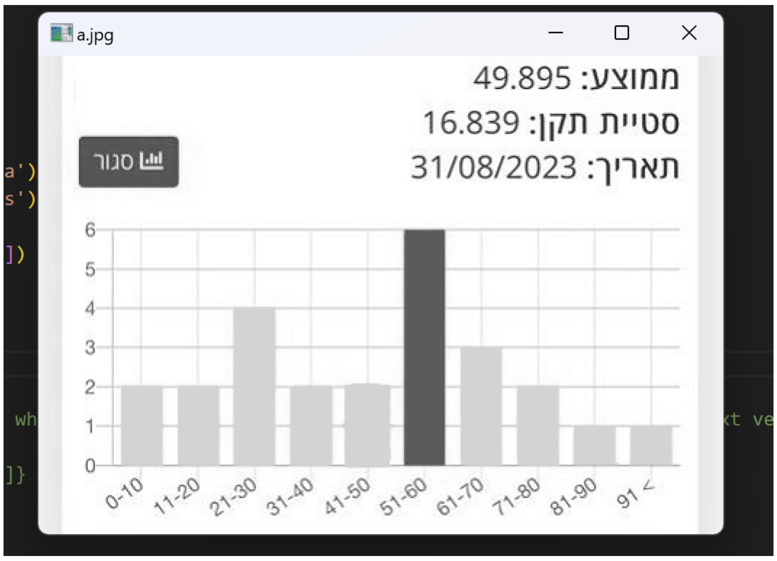
$$0.25 \;\geq 2 \cdot \frac{k}{2} \;\rightarrow 0.25 \;\geq k$$

This means we can fully restore the sine image for k values:

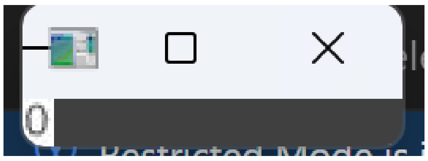
$$k \leq 0.25\,.$$

Problem 3 – Histograms, Matching, Quantization (70 points):

a) This section has already been implemented.
The function read_dir was called to read all images from the data folder.



b) The function read_dir was called to read all digits from the data folder.
To verify that the images were read properly, the first image was displayed using cv2.imshow.

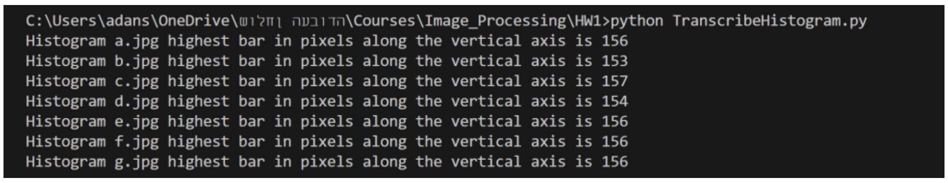


The calculate_highest_bars function was implemented to identify the tallest bar in each histogram image. This function:

- Iterates over each image.
- Computes the bar height using pixel-based calculations for all bins.

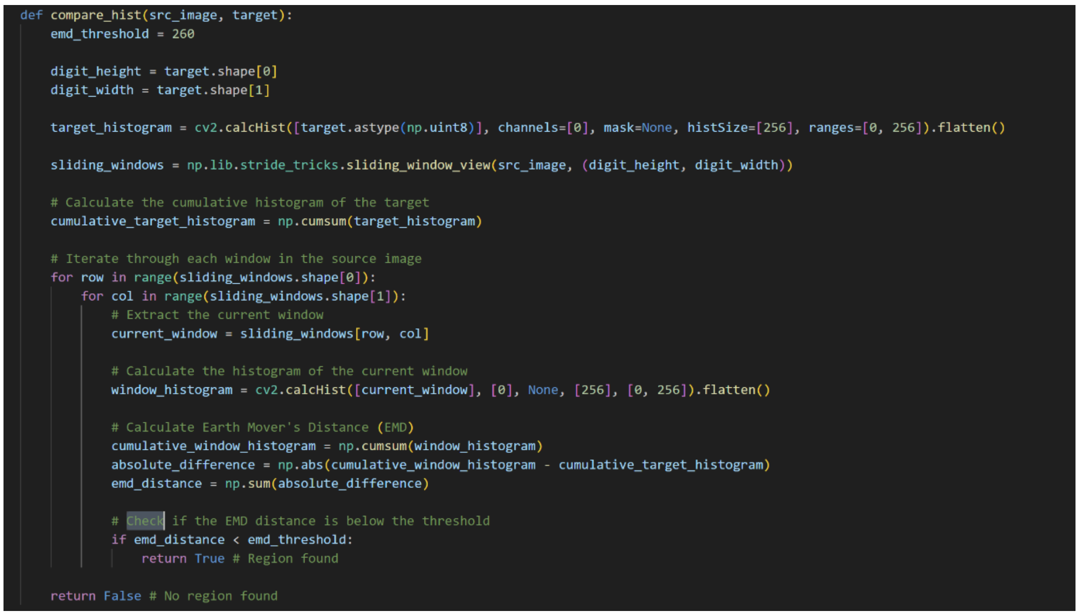
- Returns the maximum bar height in pixels for each histogram.

After running the function, the heights of the tallest bars for each histogram were calculated. The results are as follows:



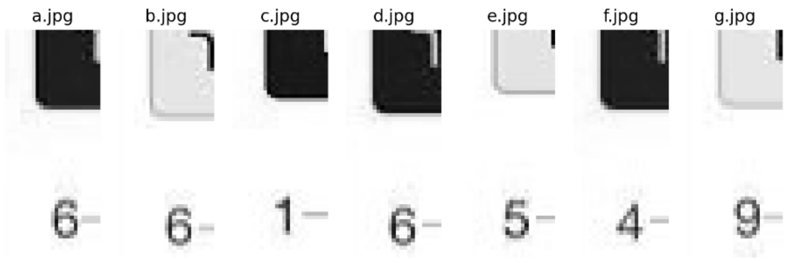
```
C:\Users\adans\OneDrive\שולחן העבודה\Courses\Image_Processing\HW1>python TranscribeHistogram.py
Histogram a.jpg highest bar in pixels along the vertical axis is 156
Histogram b.jpg highest bar in pixels along the vertical axis is 153
Histogram c.jpg highest bar in pixels along the vertical axis is 157
Histogram d.jpg highest bar in pixels along the vertical axis is 154
Histogram e.jpg highest bar in pixels along the vertical axis is 156
Histogram f.jpg highest bar in pixels along the vertical axis is 156
Histogram g.jpg highest bar in pixels along the vertical axis is 156
```

c) We followed the instructions and implemented the compare_hist(src_image, target) function, which performs histogram-based pattern matching using Earth Mover's Distance (EMD). The function utilizes np.lib.stride_tricks.sliding_window_view to create sliding windows of size equal to the target image, enabling efficient comparisons. For each window, we calculate its histogram using cv2.calcHist and compute the EMD against the target image's histogram. If the EMD is less than the specified threshold of 260, the function returns True, indicating a match. Otherwise, it returns False, ensuring precise and efficient detection of the target digit within the source image.

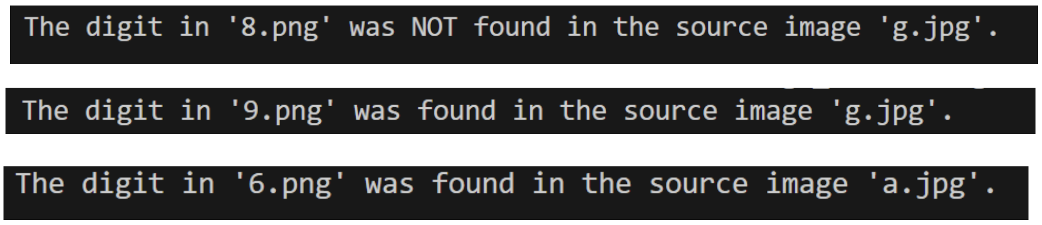


```python
def compare_hist(src_image, target):
    emd_threshold = 260

    digit_height = target.shape[0]
    digit_width = target.shape[1]

    target_histogram = cv2.calcHist([target.astype(np.uint8)], channels=[0], mask=None, histSize=[256], ranges=[0, 256]).flatten()

    sliding_windows = np.lib.stride_tricks.sliding_window_view(src_image, (digit_height, digit_width))

    # Calculate the cumulative histogram of the target
    cumulative_target_histogram = np.cumsum(target_histogram)

    # Iterate through each window in the source image
    for row in range(sliding_windows.shape[0]):
        for col in range(sliding_windows.shape[1]):
            # Extract the current window
            current_window = sliding_windows[row, col]

            # Calculate the histogram of the current window
            window_histogram = cv2.calcHist([current_window], [0], None, [256], [0, 256]).flatten()

            # Calculate Earth Mover's Distance (EMD)
            cumulative_window_histogram = np.cumsum(window_histogram)
            absolute_difference = np.abs(cumulative_window_histogram - cumulative_target_histogram)
            emd_distance = np.sum(absolute_difference)

            # Check if the EMD distance is below the threshold
            if emd_distance < emd_threshold:
                return True # Region found

    return False # No region found
```

Since we will need only the topmost number, we cropped the source image to focus on the region of interest where the digits are likely to appear. This step was achieved using the crop_regions_of_interest function, which isolates the relevant region based on predefined coordinates. All cropped images were then displayed in a grid using matplotlib to validate that the cropping was accurate.



Finally, we tested the function with multiple source and target images to confirm its performance, ensuring that it returned True for detected digits and False otherwise.



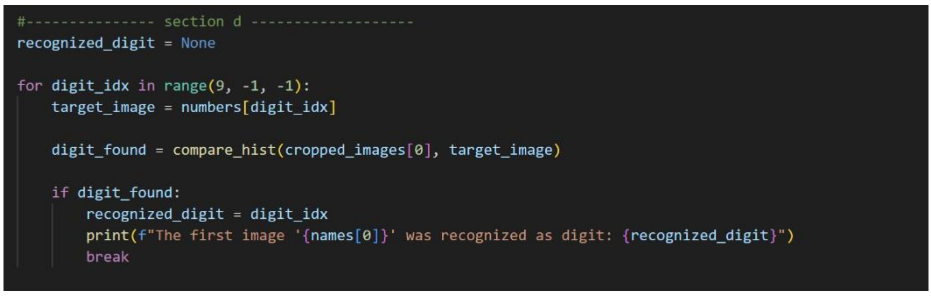
```
The digit in '8.png' was NOT found in the source image 'g.jpg'.
```

```
The digit in '9.png' was found in the source image 'g.jpg'.
```

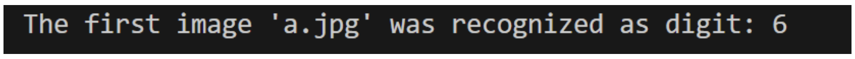
```
The digit in '6.png' was found in the source image 'a.jpg'.
```

* The validation process code is included as comments.

d) We called the function compare_hist on the first histogram, testing with the digit images in decreasing order, starting from 9 down to 0. The first number that returned True was identified as the recognized digit.

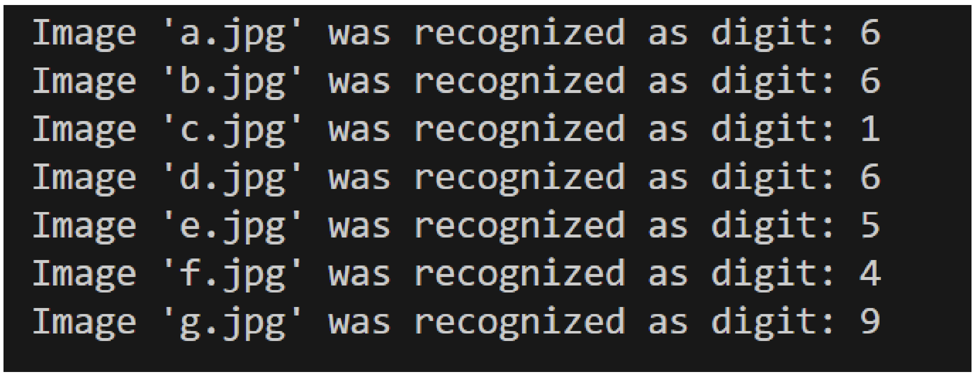

```python
#--------------- section d -------------------
recognized_digit = None

for digit_idx in range(9, -1, -1):
    target_image = numbers[digit_idx]

    digit_found = compare_hist(cropped_images[0], target_image)

    if digit_found:
        recognized_digit = digit_idx
        print(f"The first image '{names[0]}' was recognized as digit: {recognized_digit}")
        break
```

Below is the result:


```
The first image 'a.jpg' was recognized as digit: 6
```

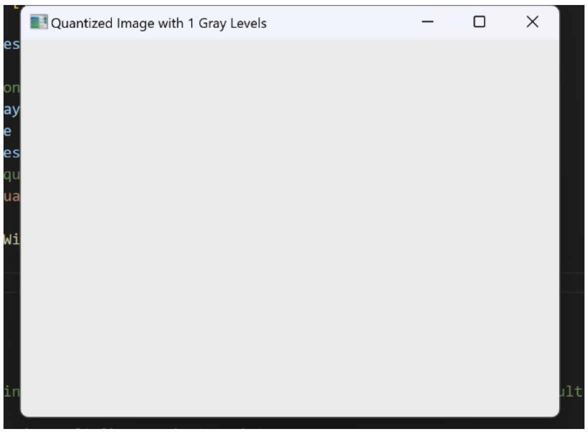
This result correctly represents the largest number in the first histogram.

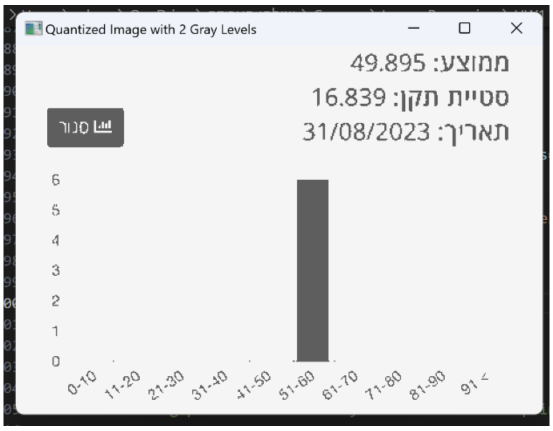
In addition, we ensured that all images produced the correct results when using compare_hist. The following results were obtained:


```
Image 'a.jpg' was recognized as digit: 6
Image 'b.jpg' was recognized as digit: 6
Image 'c.jpg' was recognized as digit: 1
Image 'd.jpg' was recognized as digit: 6
Image 'e.jpg' was recognized as digit: 5
Image 'f.jpg' was recognized as digit: 4
Image 'g.jpg' was recognized as digit: 9
```

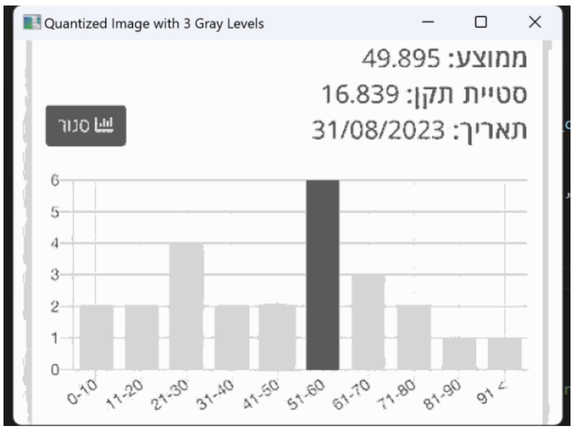
e) To distinguish the bars from the rest of the histogram image, we applied the quantization function to the first image with varying numbers of gray levels (1, 2, 3, 4). After visually evaluating the results, we determined that using 3 gray levels provided the best balance between simplicity and clarity. This level effectively separated the bars from the background without introducing unnecessary complexity.

Below are the results of applying the quantization function to the first histogram image using different numbers of gray levels (1, 2, 3, 4).

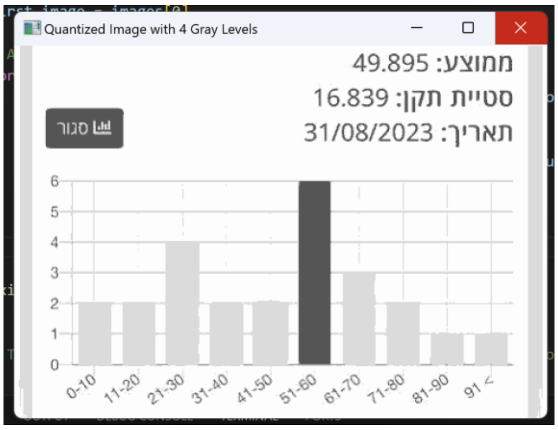




ממוצע: 49.895
סטיית תקן: 16.839
תאריך: 31/08/2023



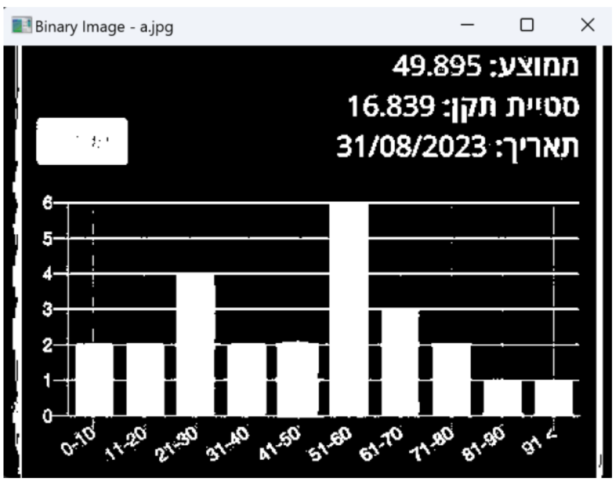
ממוצע: 49.895
סטיית תקן: 16.839
תאריך: 31/08/2023



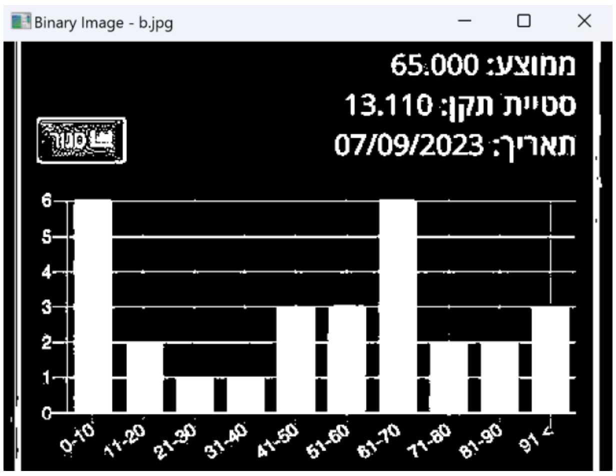
ממוצע: 49.895
סטיית תקן: 16.839
תאריך: 31/08/2023

To simplify the remaining work, we thresholded the quantized image to black and white. Observing that all bars in all images have grayscale levels lower than 220, we selected a threshold value of 220. Pixels with a grayscale values less than or equal to 220 are set to 1 (white), and the rest are set to 0 (black)..This choice ensures that all the bars are preserved without losing any relevant information.

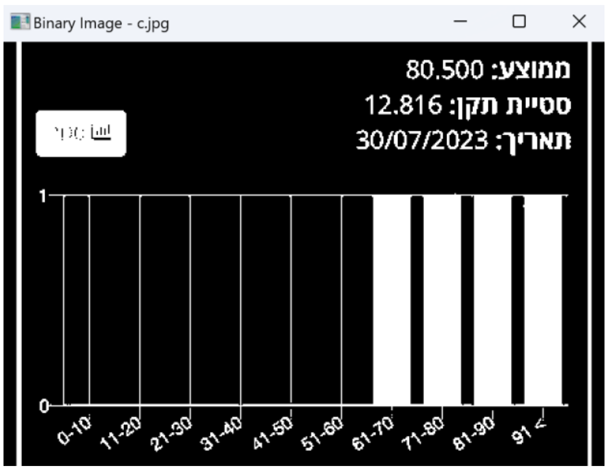
Below are the resulting images:


Binary Image - a.jpg

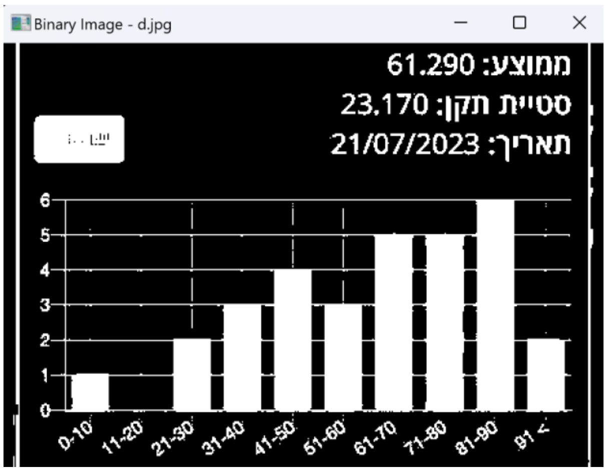
ממוצע: 49.895
סטיית תקן: 16.839
תאריך: 31/08/2023


Binary Image - b.jpg

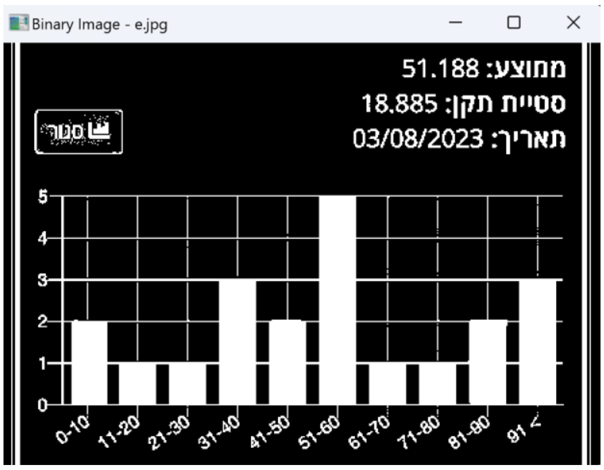
ממוצע: 65.000
סטיית תקן: 13.110
תאריך: 07/09/2023


Binary Image - c.jpg

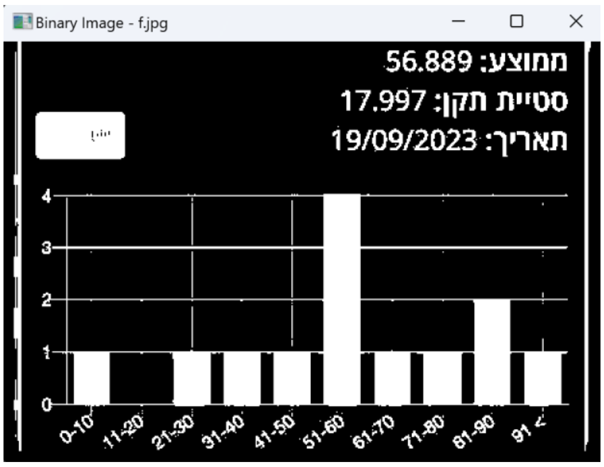
ממוצע: 80.500
סטיית תקן: 12.816
תאריך: 30/07/2023


Binary Image - d.jpg

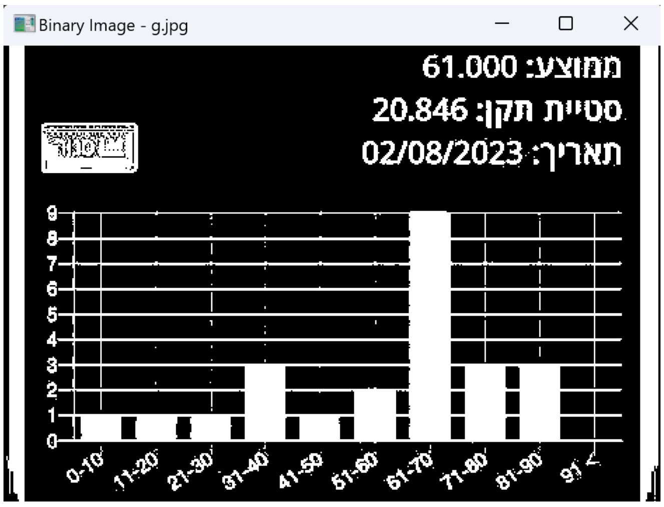
ממוצע: 61.290
סטיית תקן: 23.170
תאריך: 21/07/2023


Binary Image - e.jpg

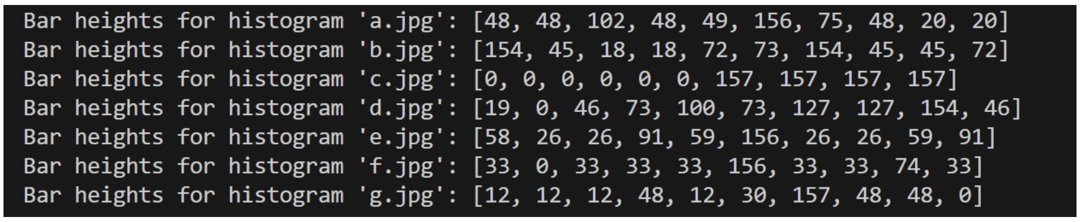
ממוצע: 51.188
סטיית תקן: 18.885
תאריך: 03/08/2023


Binary Image - f.jpg

ממוצע: 56.889
סטיית תקן: 17.997
תאריך: 19/09/2023



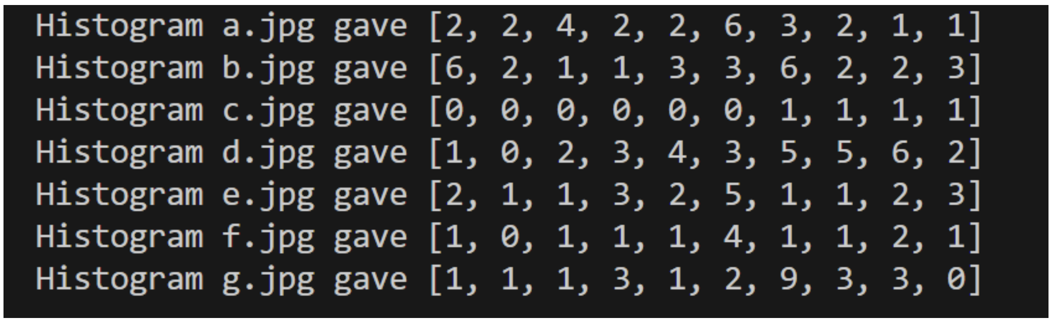
f) For each binary histogram image, we calculated the heights of all 10 bars (bins) in pixels using the get_bar_height function.
Below are the calculated bar heights for all histograms:



```
Bar heights for histogram 'a.jpg': [48, 48, 102, 48, 49, 156, 75, 48, 20, 20]
Bar heights for histogram 'b.jpg': [154, 45, 18, 18, 72, 73, 154, 45, 45, 72]
Bar heights for histogram 'c.jpg': [0, 0, 0, 0, 0, 0, 157, 157, 157, 157]
Bar heights for histogram 'd.jpg': [19, 0, 46, 73, 100, 73, 127, 127, 154, 46]
Bar heights for histogram 'e.jpg': [58, 26, 26, 91, 59, 156, 26, 26, 59, 91]
Bar heights for histogram 'f.jpg': [33, 0, 33, 33, 33, 156, 33, 33, 74, 33]
Bar heights for histogram 'g.jpg': [12, 12, 12, 48, 12, 30, 157, 48, 48, 0]
```

g) We calculate the real students’ number for each bin using the provided formula.

The results for all histograms are printed, showing the real students' number for each bin.



```
Histogram a.jpg gave [2, 2, 4, 2, 2, 6, 3, 2, 1, 1]
Histogram b.jpg gave [6, 2, 1, 1, 3, 3, 6, 2, 2, 3]
Histogram c.jpg gave [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
Histogram d.jpg gave [1, 0, 2, 3, 4, 3, 5, 5, 6, 2]
Histogram e.jpg gave [2, 1, 1, 3, 2, 5, 1, 1, 2, 3]
Histogram f.jpg gave [1, 0, 1, 1, 1, 4, 1, 1, 2, 1]
Histogram g.jpg gave [1, 1, 1, 3, 1, 2, 9, 3, 3, 0]
```