IMPORT LIBRARIES

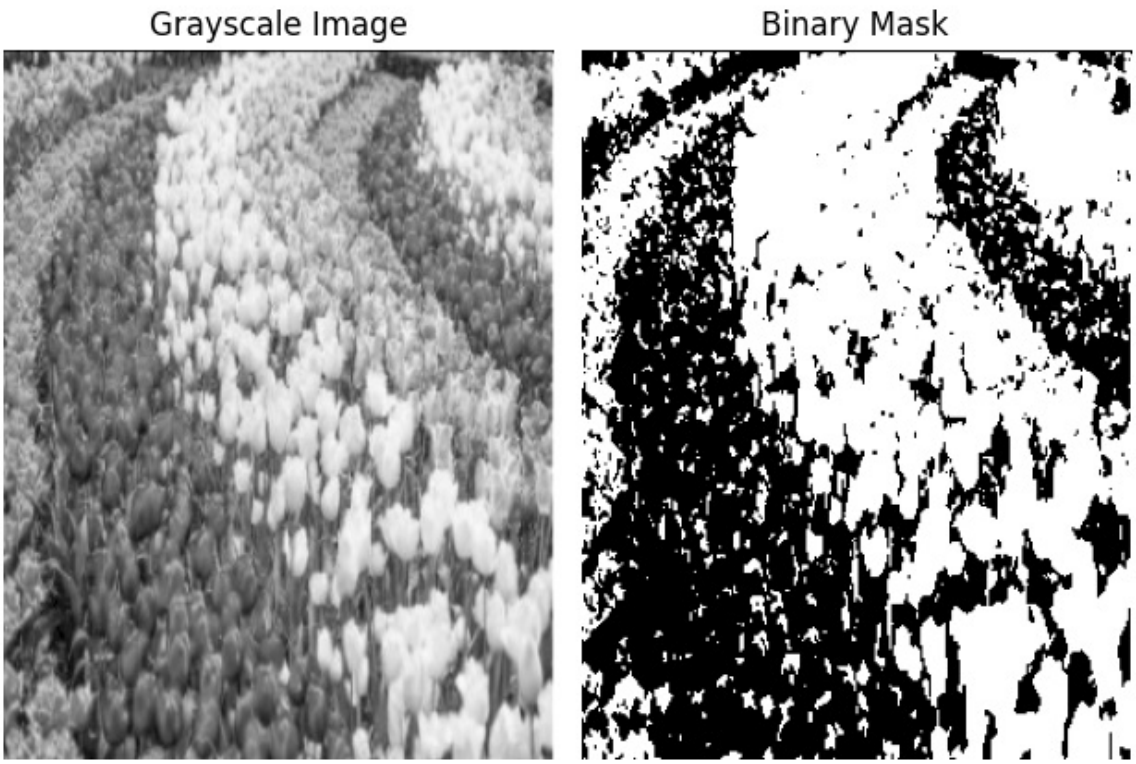
In [ ]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

Global Thresholding

In [6]:

```
image = cv2.imread('/content/sample_data/4.png', cv2.IMREAD_GRAYSCALE)

_, binary_mask = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(binary_mask, cmap='gray')
plt.title('Binary Mask')
plt.axis('off')


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

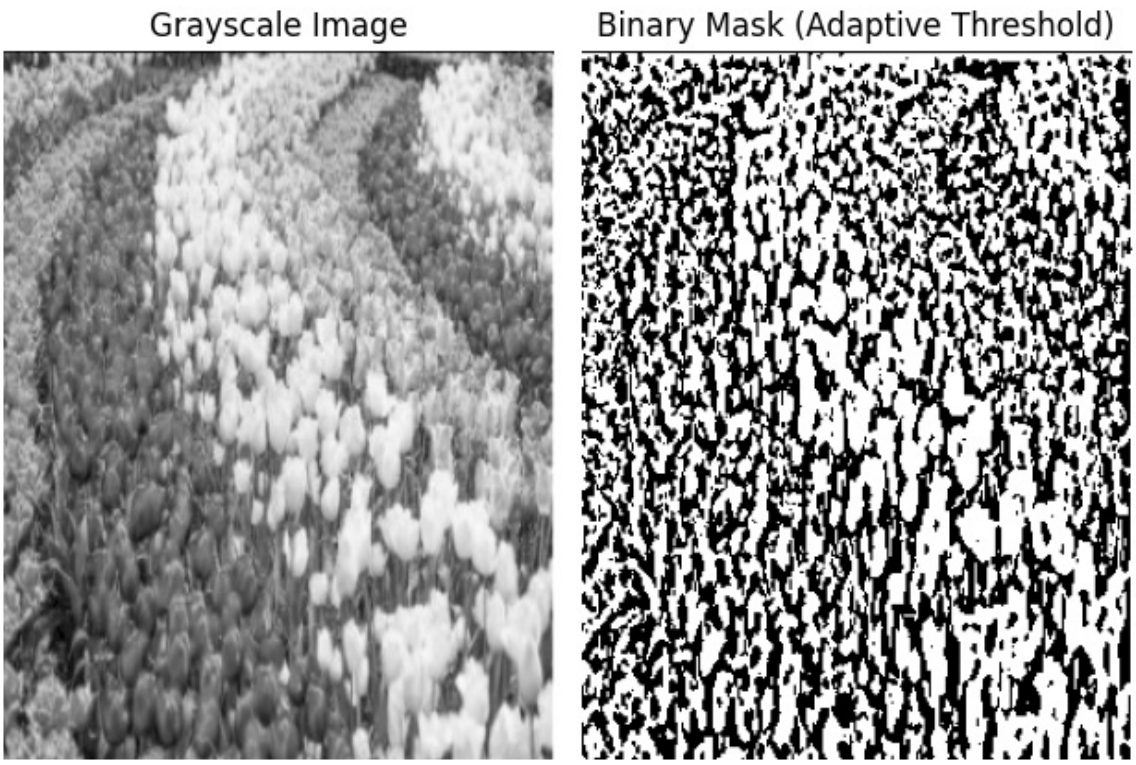


Adaptive Thresholding

In [2]:

```
image = cv2.imread('/content/sample_data/4.png', cv2.IMREAD_GRAYSCALE)

binary_mask = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BI
NARY, 11, 2)
```

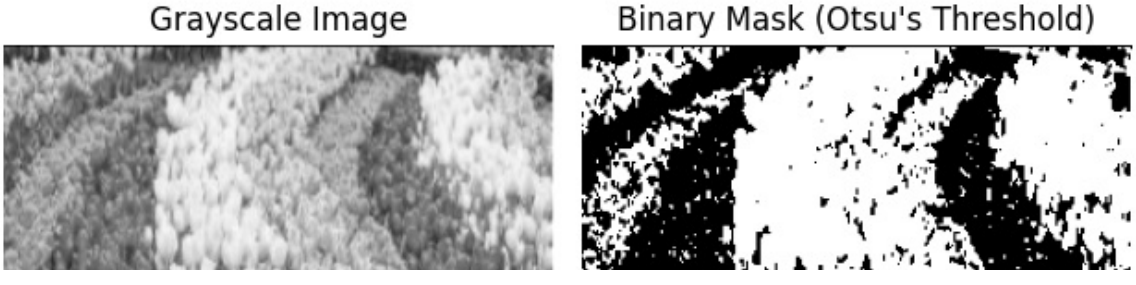
```
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(binary_mask, cmap='gray')
plt.title('Binary Mask (Adaptive Threshold)')
plt.axis('off')


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

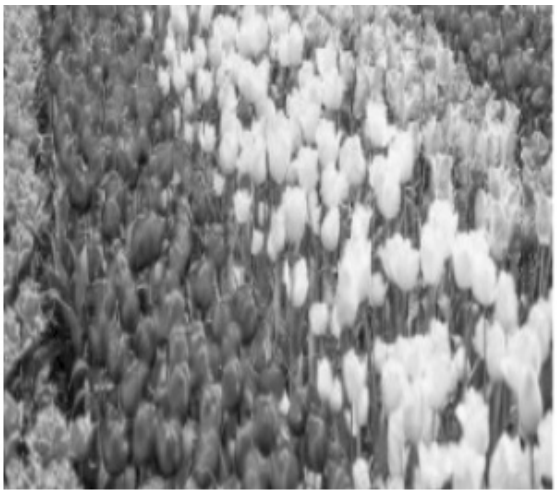


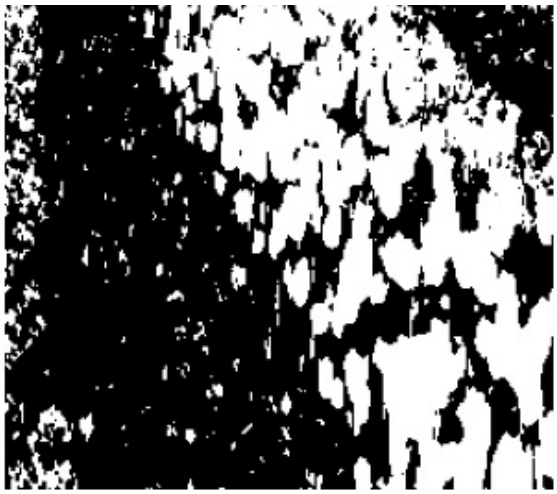
OTSU THRESHOLDING

In [3]:

```
image = cv2.imread('/content/sample_data/4.png', cv2.IMREAD_GRAYSCALE)

_, binary_mask = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(binary_mask, cmap='gray')
plt.title("Binary Mask (Otsu's Threshold)")
plt.axis('off')


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



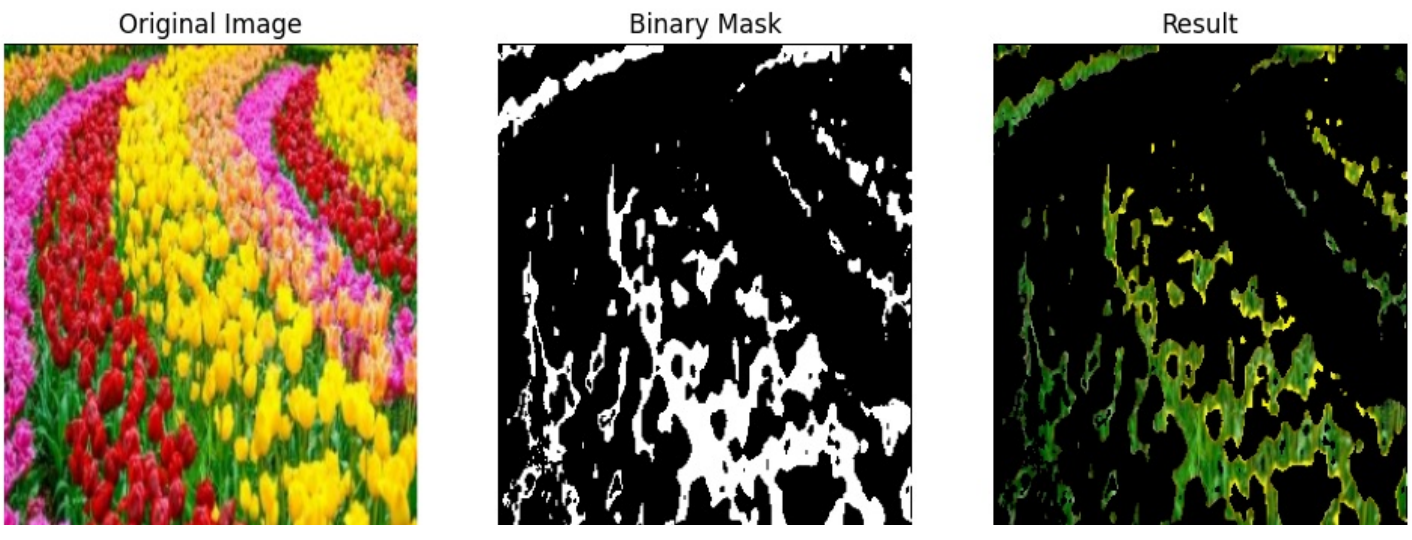




Color-Based Thresholding

In [4]:

```
image = cv2.imread('/content/sample_data/4.png')

hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

lower_color = np.array([30, 50, 50])  # Lower HSV values for the color (adjust as needed)
upper_color = np.array([60, 255, 255])  # Upper HSV values for the color (adjust as needed)

binary_mask = cv2.inRange(hsv_image, lower_color, upper_color)

result = cv2.bitwise_and(image, image, mask=binary_mask)

plt.figure(figsize=(10, 4))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(binary_mask, cmap='gray')
plt.title('Binary Mask')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title('Result')
plt.axis('off')

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

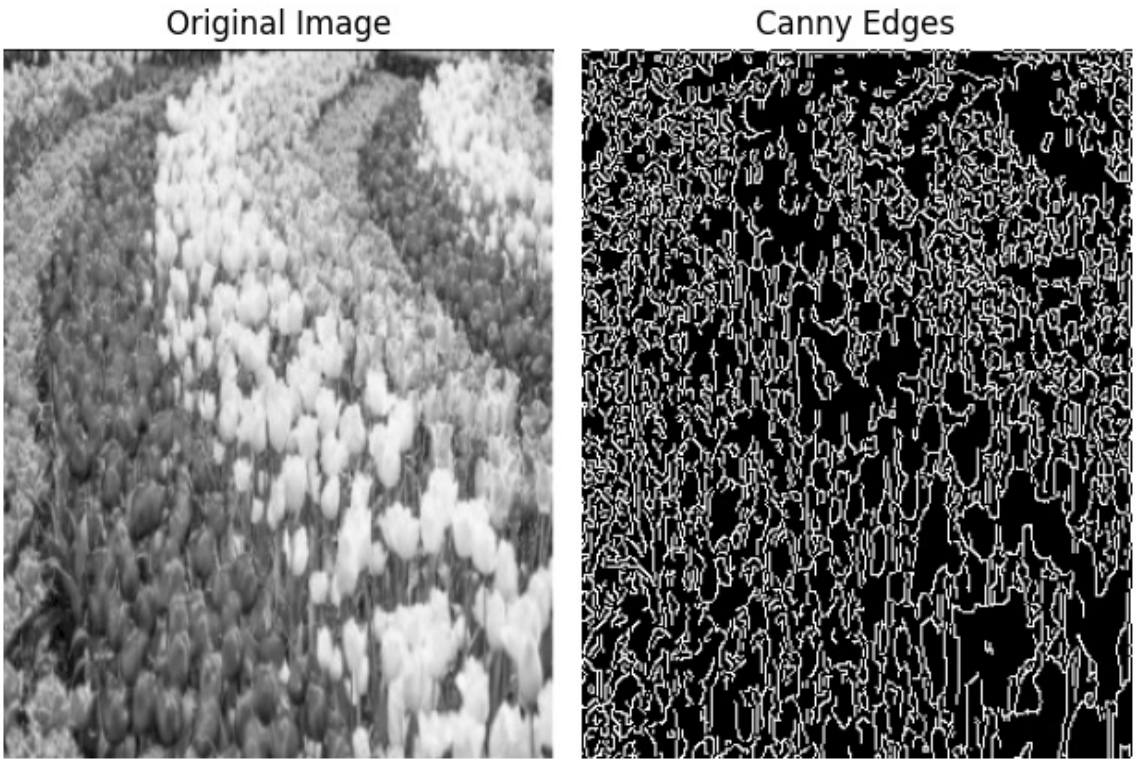




Hysteresis Thresholding (Canny Edge Detector)

In [5]:

```
image = cv2.imread('/content/sample_data/4.png', cv2.IMREAD_GRAYSCALE)

edges = cv2.Canny(image, threshold1=100, threshold2=200)  # Adjust thresholds as needed

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edges')
plt.axis('off')

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

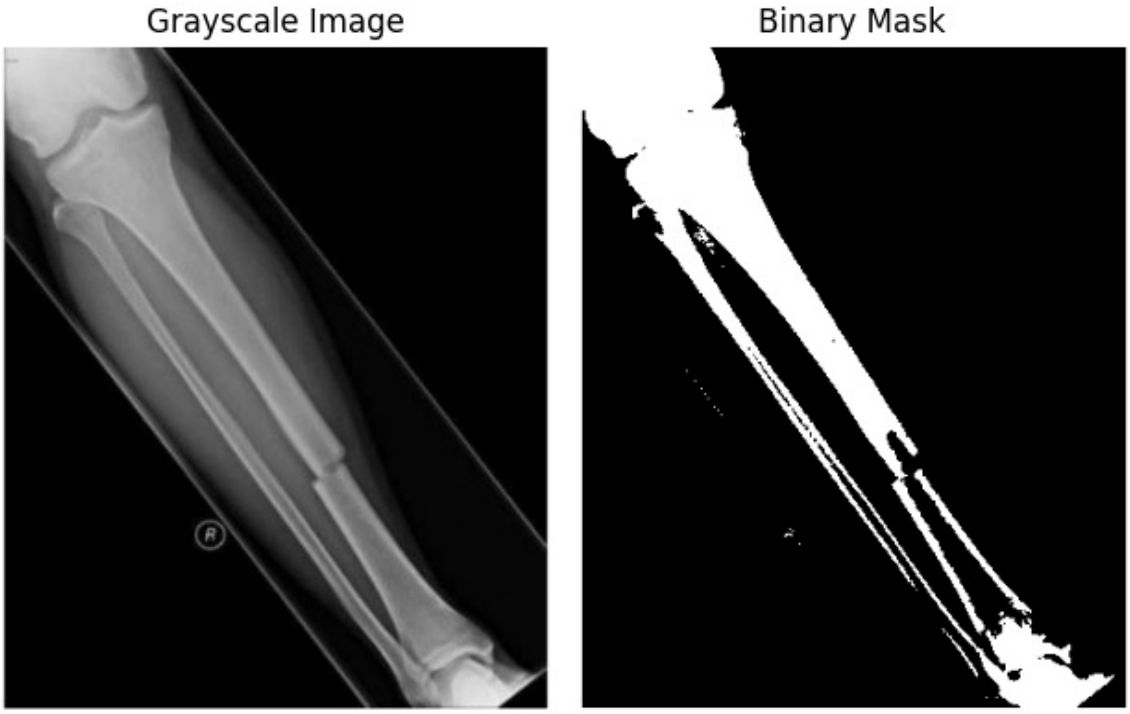


TASK 1:Thresholding-Based Segmentation

In [ ]:

```
image = cv2.imread('/content/sample_data/1.JPG', cv2.IMREAD_GRAYSCALE)

if image is None:
    print("Error: Could not load the image.")
else:

    _, binary_mask = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)


    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Grayscale Image')
    plt.axis('off')
```

```
    plt.subplot(1, 2, 2)
    plt.imshow(binary_mask, cmap='gray')
    plt.title('Binary Mask')
    plt.axis('off')

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

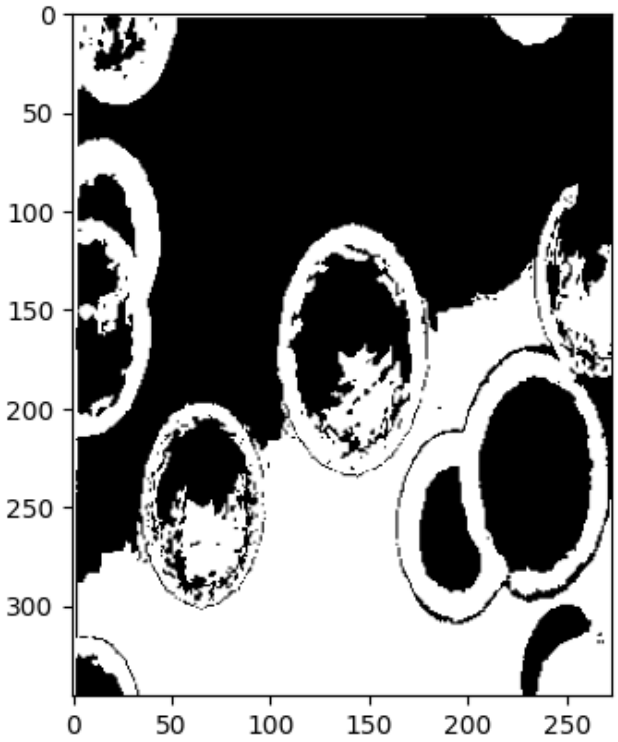


TASK 2:Region Growing Intensity-Based Segmentation

In [ ]:

```
image = cv2.imread('/content/sample_data/2.png', cv2.IMREAD_GRAYSCALE)

seed_point = (18, 16)

def region_growing(image, seed, threshold):
    mask = np.zeros_like(image, dtype=np.uint8)


    stack = [seed]
    seed_intensity = image[seed]

    while stack:
        x, y = stack.pop()

        if x < 0 or x >= image.shape[0] or y < 0 or y >= image.shape[1]:
            continue

        if mask[x, y] == 255:
            continue

        if abs(image[x, y] - seed_intensity) <= threshold:
            mask[x, y] = 255
            stack.extend([(x+1, y), (x-1, y), (x, y+1), (x, y-1)])

    return mask

threshold = 200

segmented_image = region_growing(image, seed_point, threshold)

plt.imshow(segmented_image, cmap='gray')
plt.show()
```

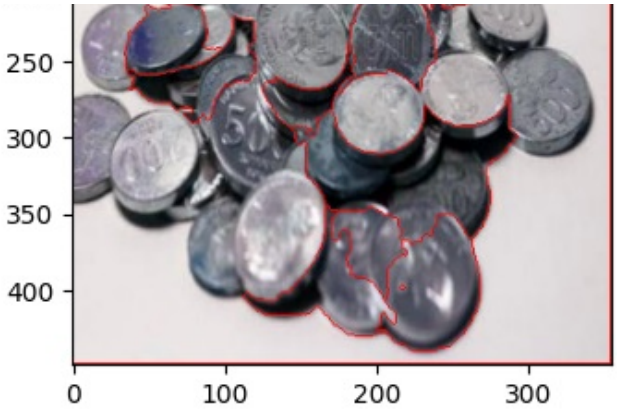
```
<ipython-input-13-e6dcb800c248>:34: RuntimeWarning: overflow encountered in ubyte_scalars
  if abs(image[x, y] - seed_intensity) <= threshold:
```



TASK 3: Watershed Segmentation

In [ ]:

```
image = cv2.imread("/content/sample_data/3.JPG")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

kernel = np.ones((7, 7), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

sure_bg = cv2.dilate(opening, kernel, iterations=3)
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.6 * dist_transform.max(), 255, cv2.THRESH_BI
NARY)

sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

_, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0

cv2.watershed(image, markers)

image[markers == -1] = [255, 0, 0]

plt.imshow(image)
plt.axis("on")
plt.show()
```



TASK 4: Cluster-Based Segmentation

In [ ]:

```
image = cv2.imread("/content/sample_data/4.png")

pixel_values = image.reshape((-1, 3))

pixel_values = np.float32(pixel_values)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 180, 0.4)
K = 3
_, labels, centers = cv2.kmeans(pixel_values, K, None, criteria, 50, cv2.KMEANS_RANDOM_C
ENTERS)

centers = np.uint8(centers)
segmented_image = centers[labels.flatten()]
segmented_image = segmented_image.reshape(image.shape)
cv2_imshow(segmented_image)
```