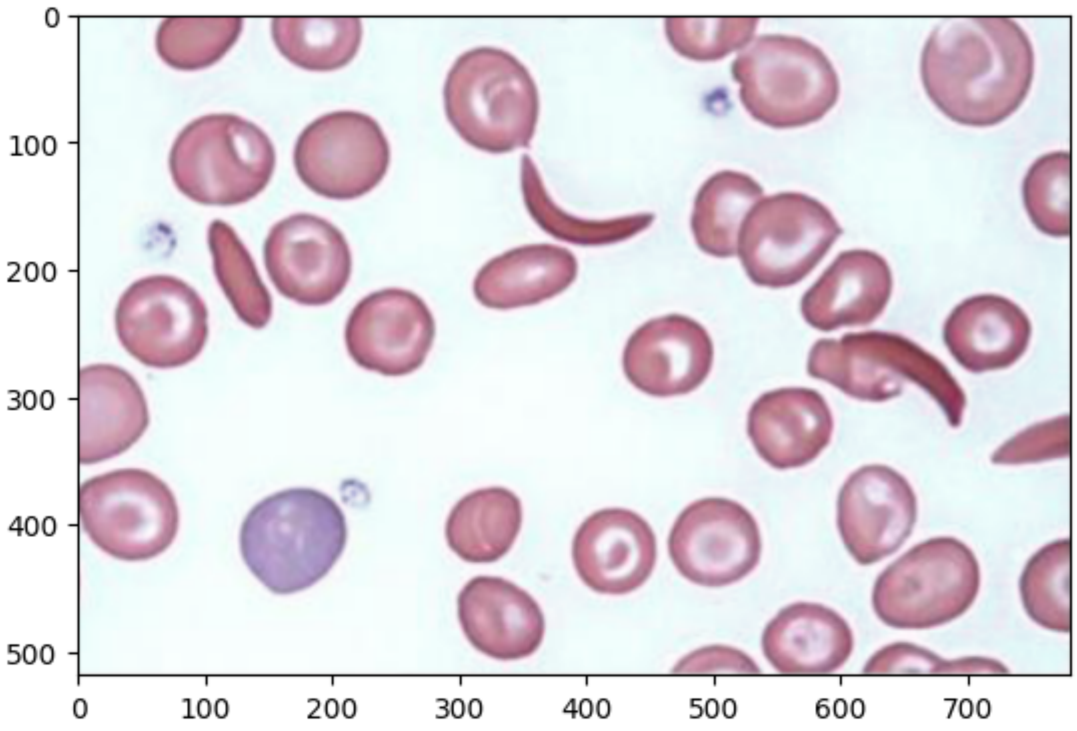
In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

Read Image

In [2]:
```
img = cv2.imread('images/DIP_1.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [3]:
```
plt.imshow(img)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x7bb441287bc0>
```

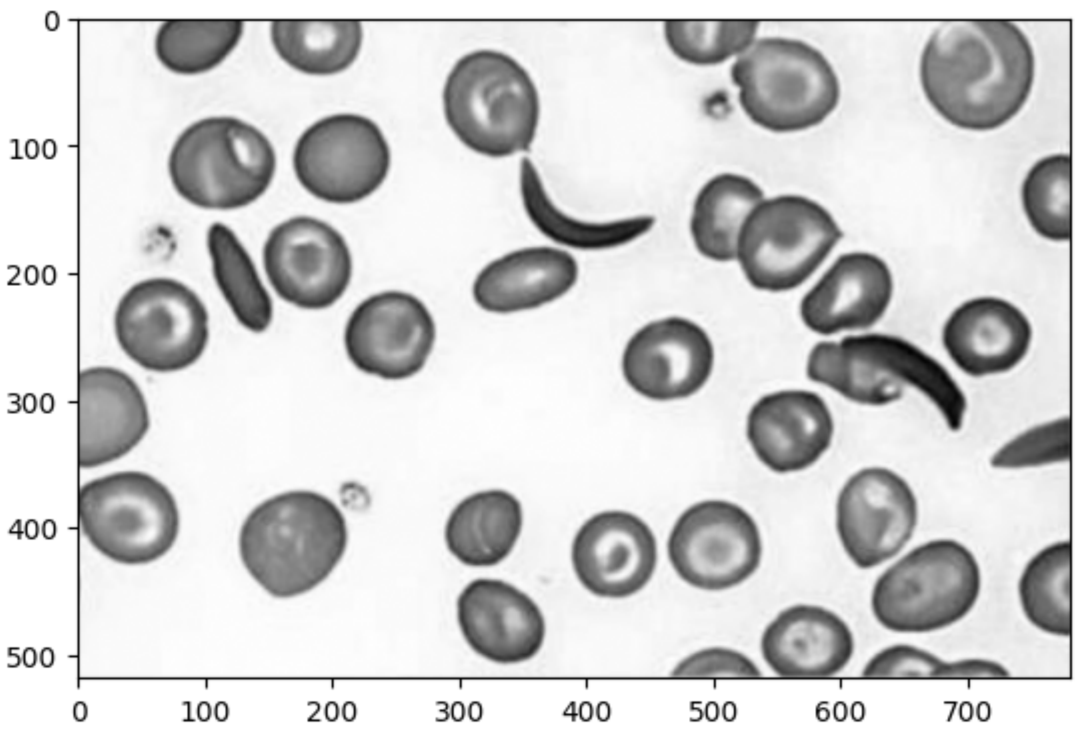


Convert to Grayscale

In [4]:
```
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```

In [5]:
```
plt.imshow(img_gray, cmap='gray')
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7bb44113d550>
```

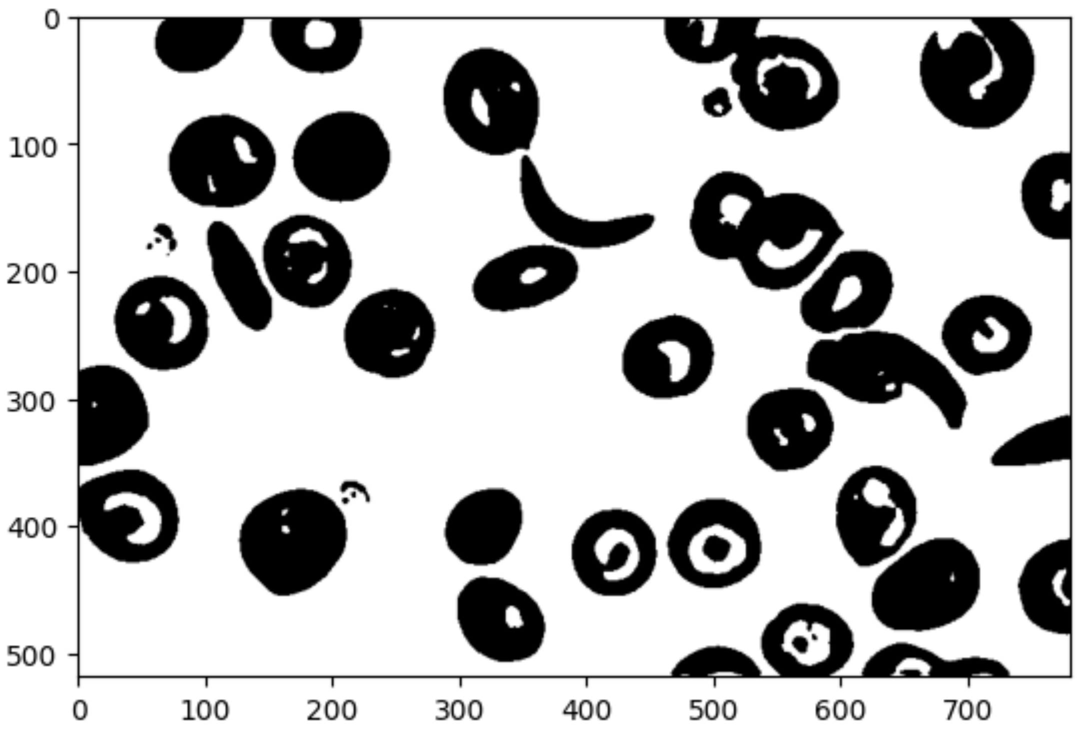


Convert to Binary

```
In [6]: # Using Otsu's method to find the optimal threshold
        img_binary = cv2.threshold(img_gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_
```

```
In [7]: plt.imshow(img_binary, cmap='gray')
```

```
Out[7]: <matplotlib.image.AxesImage at 0x7bb440fc9040>
```

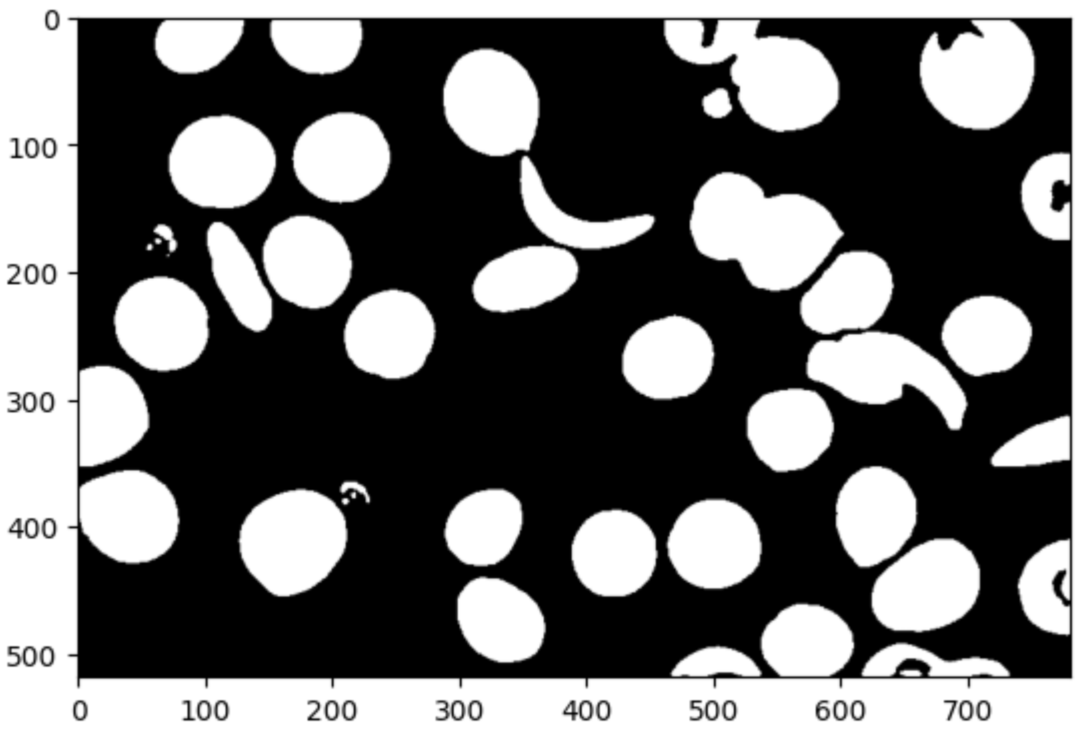


Fill Holes

In [8]:
```
# Inverting the binary image to find contours
img_inverted = cv2.bitwise_not(img_binary)

# Finding contours
contours, hierarchy = cv2.findContours(img_inverted, cv2.RETR_CCOMP, cv2.CHA

# Creating a mask for the contours
mask = np.zeros_like(img_binary)
for i, contour in enumerate(contours):
    if hierarchy[0][i][3] == -1: # Only fill the inner contours
        cv2.drawContours(mask, [contour], 0, 255, -1)

# Filling the holes in the original image
img_fill = cv2.bitwise_or(img_inverted, mask)
```

In [9]:
```
plt.imshow(img_fill, cmap='gray')
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x7bb44049b470>
```



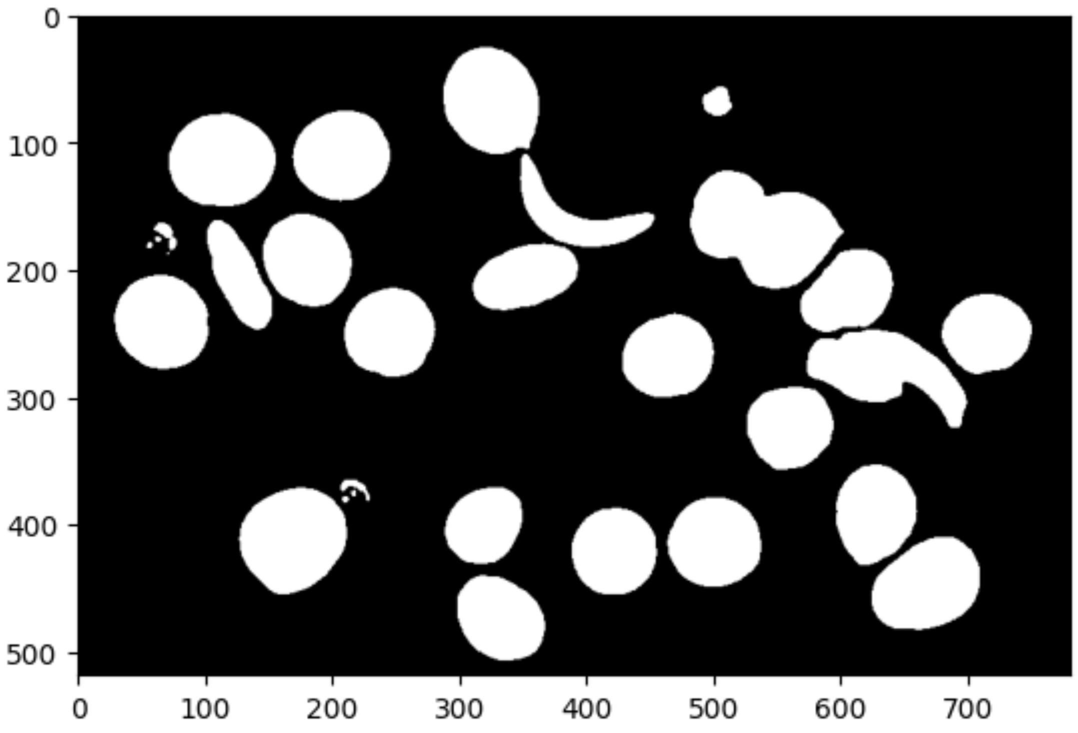
Clear Borders

In [10]:
```
# Make a border around the image to avoid filling the outermost contour
pad = cv2.copyMakeBorder(img_fill, 1, 1, 1, 1, cv2.BORDER_CONSTANT, value=25
h, w = pad.shape

# Flood fill from the top-left corner (0, 0) to fill the outermost contour
mask = np.zeros((h + 2, w + 2), dtype=np.uint8)
img_no_border = cv2.floodFill(pad, mask, (0, 0), 0, (5), (0), flags=8)[1]
```

```
# Convert back to the original size
img_no_border = img_no_border[1:h - 1, 1:w - 1]
```

In [11]:
```
plt.imshow(img_no_border, cmap='gray')
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7bb44051e840>
```

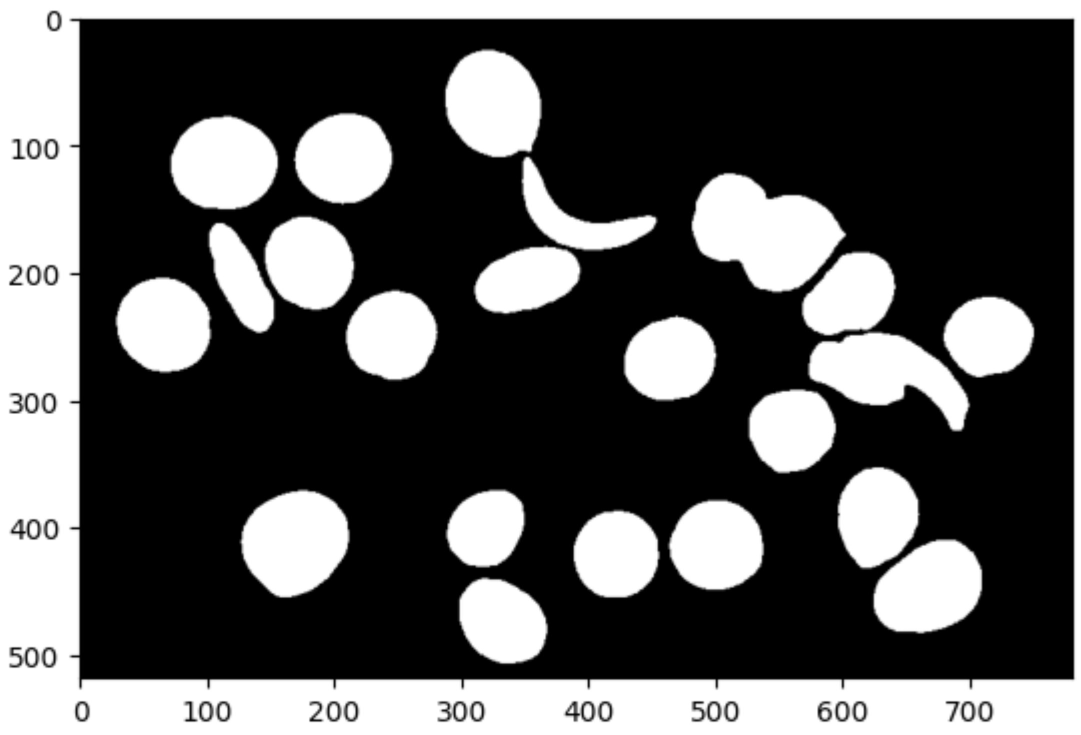


Remove Small Objects

In [12]:
```
# Finding contours again to remove small objects
contours, _ = cv2.findContours(img_no_border, cv2.RETR_EXTERNAL, cv2.CHAIN_A

# Removing small objects
img_no_small_objects = img_no_border.copy()
min_size = 1000
for contour in contours:
    if cv2.contourArea(contour) < min_size: # If the contour area is less th
        cv2.drawContours(img_no_small_objects, [contour], -1, 0, -1)
```

In [13]:
```
plt.imshow(img_no_small_objects, cmap='gray')
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x7bb4403a1c70>
```

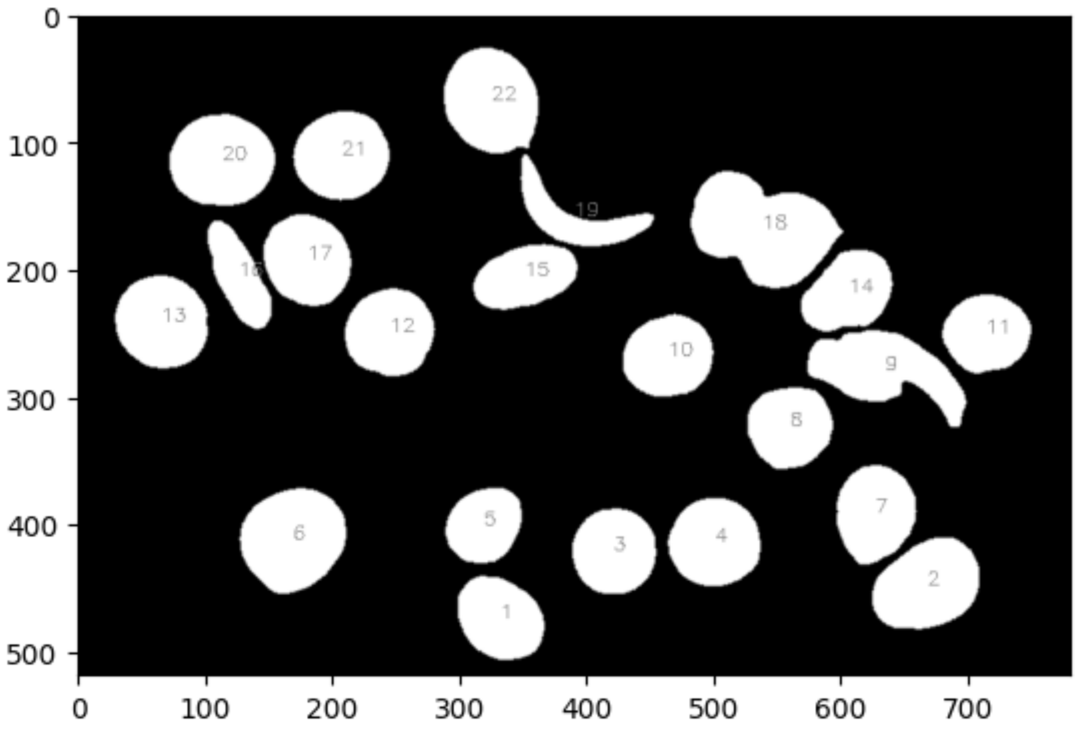


Count Objects

In [14]:
```
# Finding contours again to count the objects
contours, _ = cv2.findContours(img_no_small_objects, cv2.RETR_EXTERNAL, cv2.

# Drawing contours and counting them
img_counted = img_no_small_objects.copy()
for i, contour in enumerate(contours):
    cv2.drawContours(img_counted, [contour], -1, 128, 1)
    M = cv2.moments(contour)
    if M["m00"] != 0:
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        cv2.putText(img_counted, str(i + 1), (cX, cY), cv2.FONT_HERSHEY_SIMP
```

In [15]:
```
plt.imshow(img_counted, cmap='gray')
```

Out[15]:
```
<matplotlib.image.AxesImage at 0x7bb4403c5670>
```


0
100
200
300
400
500
0
100
200
300
400
500
600
700
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22