EXP NO: 7	IMPLEMENTATION OF VARIOUS IMAGE SEGMENTATION ALGORITHMS

AIM:

To implement different image segmentation algorithms (edge-based, region-based, clustering, and advanced methods like watershed and GrabCut).

REQUIREMENTS:

Software: Python 3.x
Libraries: OpenCV, NumPy, Matplotlib, scikit-image, scikit-learn

Install:
pip install opencv-python numpy matplotlib scikit-image scikit-learn

ALGORITHMS COVERED:

1. Edge-based Segmentation (Canny)
2. Region Growing (Seed-based)
3. Watershed Segmentation
4. Mean-Shift Segmentation (using skimage)
5. GrabCut (Graph-cut based)

EDGE-BASED SEGMENTATION (CANNY)

ALGORITHM STEPS:

Convert to grayscale and apply Gaussian blur to reduce noise.

1. Use cv2.Canny with thresholds (minVal, maxVal) to detect edges.
2. Optionally apply morphological closing to get continuous contours.
3. Find contours and mask regions if desired.

REGION GROWING (SEED-BASED)

ALGORITHM STEPS:

1. Choose one or more seed points inside the object region.

2. Initialize a region containing the seed(s).

3. Iteratively add neighboring pixels whose similarity to region (e.g., intensity difference) is below threshold.

4. Stop when no more pixels can be added.

WATERSHED SEGMENTATION

ALGORITHM STEPS:

1. Compute sure background via dilation and sure foreground via distance transform + threshold.

2. Compute unknown region and label markers.

3. Apply cv2.watershed to segment regions; boundaries marked on image.

4. Refine using morphological operations.

MEAN-SHIFT / FELZENSZWALB / SLIC (SUPERPIXEL-STYLE)

ALGORITHM STEPS:

1. Convert image to appropriate float range.

2. Use skimage.segment.felzenszwalb or slic or quickshift to obtain segments.

3. Map labels to colors and visualize.

GRABCUT (INTERACTIVE GRAPH-CUT)

ALGORITHM STEPS:

1. Provide an initial bounding rectangle around the object, or use mask with probable foreground/background.

2. Initialize bgdModel and fgdModel and call cv2.grabCut with iterCount.

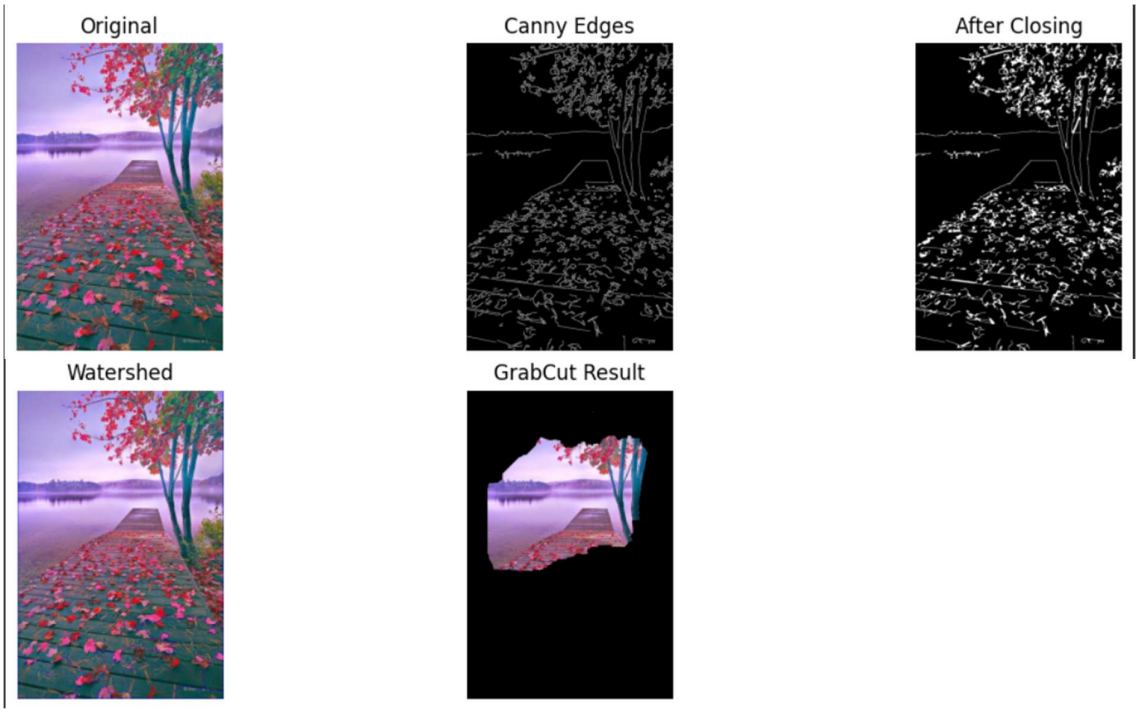
3. Extract mask where probable/definite foreground labels are kept and visualize result.

CODE:

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


# Read image
img = cv2.imread("/content/img.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# ---- Canny Edge Detection ----
edges = cv2.Canny(cv2.GaussianBlur(gray,(5,5),0),100,200)
closed = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, np.ones((3,3),np.uint8))


# ---- Watershed Segmentation ----
_,thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,np.ones((3,3),np.uint8),iterations=2)
```

```
sure_bg = cv2.dilate(opening,None,iterations=3)
dist = cv2.distanceTransform(opening,cv2.DIST_L2,5)
_,sure_fg = cv2.threshold(dist,0.7*dist.max(),255,0)
unknown = cv2.subtract(sure_bg,np.uint8(sure_fg))
_,markers = cv2.connectedComponents(np.uint8(sure_fg))
markers = markers+1; markers[unknown==255]=0
ws_img = img.copy(); markers = cv2.watershed(ws_img,markers)
ws_img[markers==-1]=[255,0,0]
ws_disp = ws_img.astype(np.uint8)

# ---- GrabCut Segmentation ----
mask = np.zeros(img.shape[:2],np.uint8)
bgdModel,fgdModel = np.zeros((1,65),np.float64),np.zeros((1,65),np.float64)
rect = (50,50,img.shape[1]-100,img.shape[0]-100)
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
result = img * np.where((mask==2)|(mask==0),0,1)[:,:,None]

# ---- Display ----
titles = ["Original","Canny Edges","After Closing","Watershed","GrabCut Result"]
images = [img[...,::-1], edges, closed, ws_disp[...,::-1], result[...,::-1]]
plt.figure(figsize=(12,6))
for i in range(5):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i], cmap='gray' if i in [1,2] else None)
    plt.title(titles[i]); plt.axis('off')
plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

The image was successfully segmented using different algorithms, effectively separating regions of interest from the background. Methods like thresholding, region-based, and edge-based segmentation highlighted distinct objects and boundaries, producing clear segmented outputs for analysis.