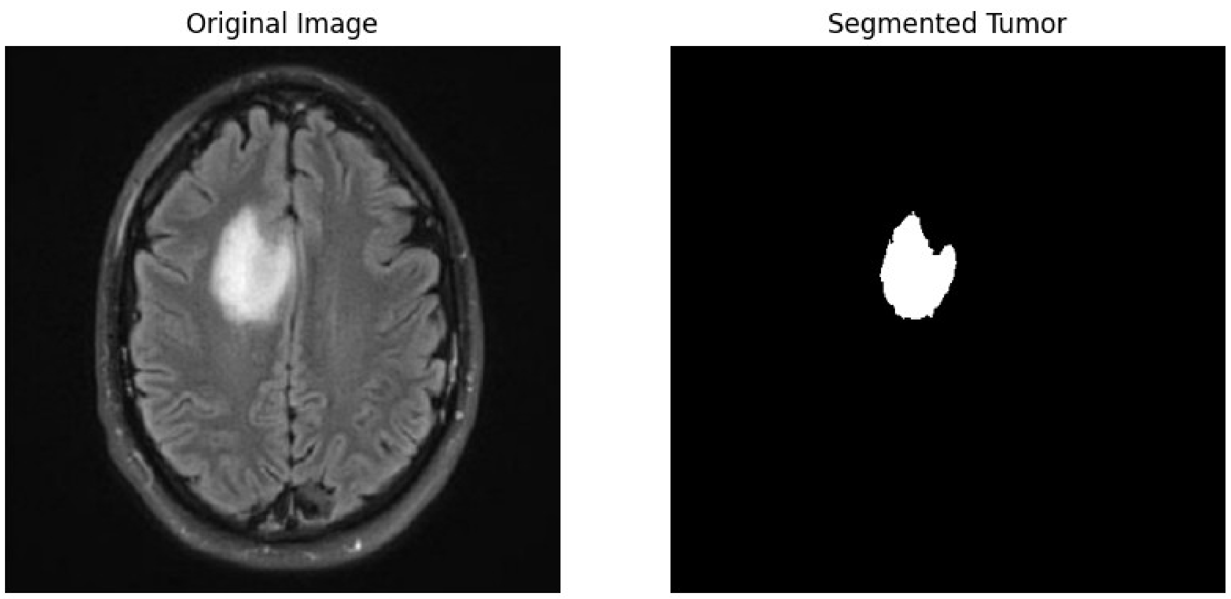
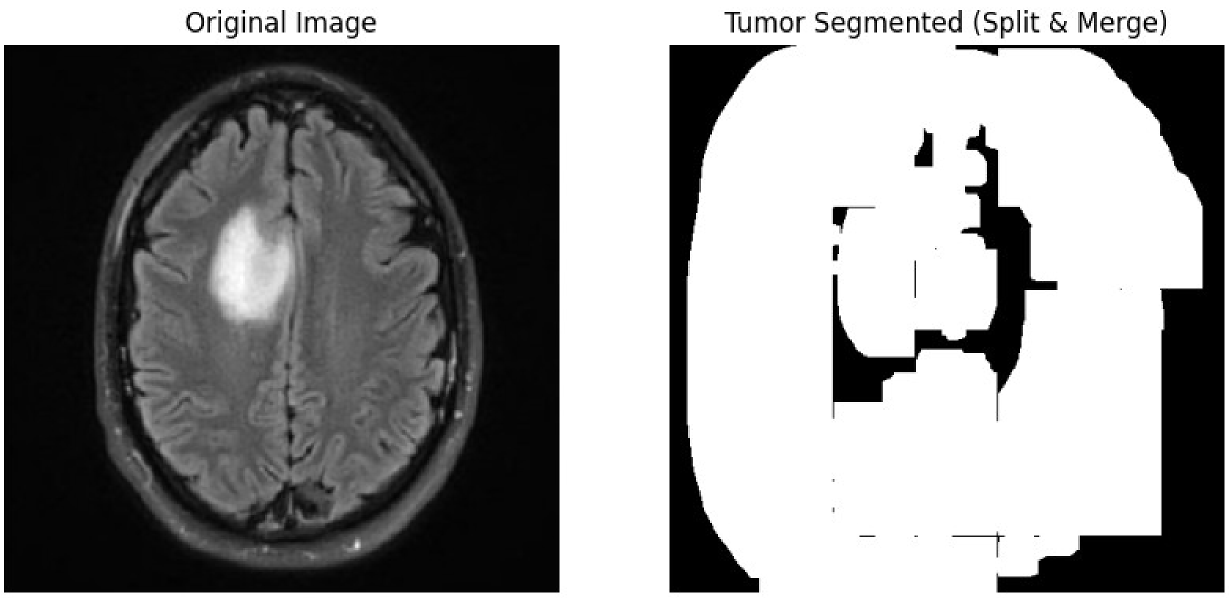
ID: 2023-1-60-075

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from collections import deque

def region_growing(image, seed_row, seed_col, threshold,
connectivity=8):
    if len(image.shape) == 3:
        image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        image_gray = image.copy()

    segmented_image = np.zeros_like(image_gray, dtype=np.uint8)
    queue = deque()
    queue.append((seed_row, seed_col))
    seed_intensity = int(image_gray[seed_row, seed_col])

    while queue:
        r, c = queue.pop()
        if r < 0 or r >= image_gray.shape[0] or c < 0 or c >=
image_gray.shape[1]:
            continue
        if segmented_image[r, c] == 1:
            continue
        if abs(int(image_gray[r, c]) - seed_intensity) <= threshold:
            segmented_image[r, c] = 1
            for i in range(-1, 2):
                for j in range(-1, 2):
                    if i == 0 and j == 0:
                        continue
                    if connectivity == 4 and abs(i) + abs(j) == 2:
                        continue
                    queue.append((r + i, c + j))
    return segmented_image

image = cv2.imread("/content/figure1.png")
seed_row, seed_col = 174, 177
threshold = 52
segmented_image = region_growing(image, seed_row, seed_col, threshold)
segmented_image = segmented_image * 255
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(segmented_image, cmap="gray")
plt.title("Segmented Tumor")
plt.axis("off")
plt.show()
```

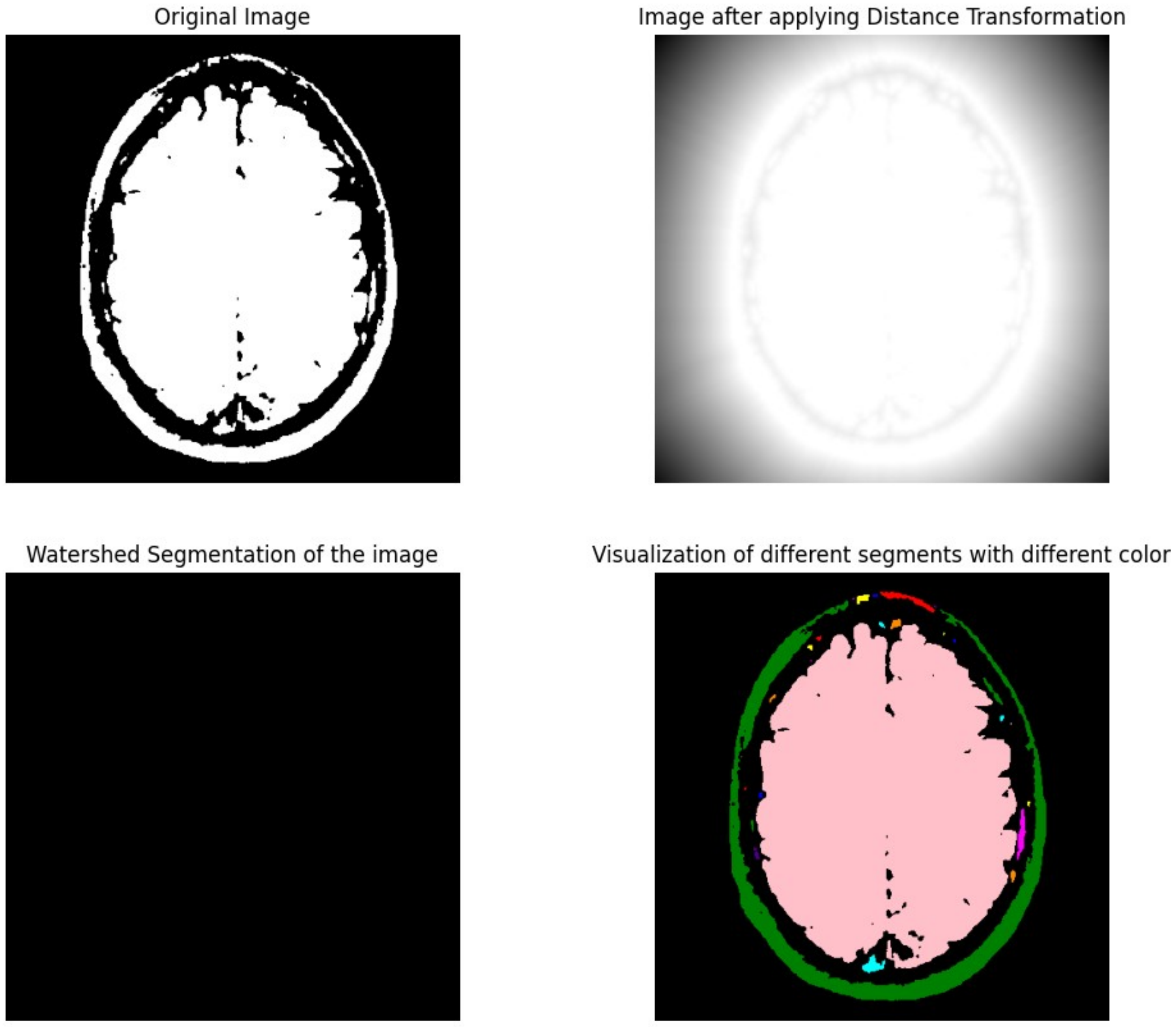


```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import label

def predicate(region, threshold=500):
    return np.var(region) > threshold

def split_merge(image, mindim, fun):
    h, w = image.shape
    size = 2 ** int(np.ceil(np.log2(max(h, w))))
    padded = np.zeros((size, size), dtype=image.dtype)
    padded[:h, :w] = image
    S = np.zeros((size, size), dtype=np.int32)

    def split(x, y, l):
        block = padded[x:x+l, y:y+l]
        if l <= mindim:
            S[x:x+l, y:y+l] = l
            return
        if fun(block):
            half = l // 2
            split(x, y, half)
            split(x, y+half, half)
            split(x+half, y, half)
            split(x+half, y+half, half)
        else:
            S[x:x+l, y:y+l] = l

    split(0, 0, size)

    g = np.zeros_like(S, dtype=np.uint8)
    unique_sizes = np.unique(S)
    for K in unique_sizes[unique_sizes > 0]:
        idxs = np.argwhere(S == K)
        for (x, y) in idxs:
            if S[x, y] == K:
                region = padded[x:x+K, y:y+K]
                if fun(region):
                    g[x:x+K, y:y+K] = 1

    labeled, _ = label(g)
    labeled = labeled[:h, :w]
    return labeled

image = cv2.imread("/content/figure1.png", cv2.IMREAD_GRAYSCALE)
result = split_merge(image, mindim=8, fun=lambda r: predicate(r,
threshold=300))

tumor_mask = (result > 0).astype(np.uint8) * 255

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.imshow(image, cmap="gray")
plt.title("Original Image")
plt.axis("off")

plt.subplot(1,2,2)
plt.imshow(tumor_mask, cmap="gray")
plt.title("Tumor Segmented (Split & Merge)")
plt.axis("off")
plt.show()
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.segmentation import watershed
from skimage.color import label2rgb
from scipy import ndimage as ndi

image = cv2.imread("/content/figure1.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

_, x = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

plt.figure(figsize=(12, 10))
plt.subplot(2, 2, 1)
plt.imshow(x, cmap="gray")
plt.title("Original Image")
plt.axis("off")

x_inv = cv2.bitwise_not(x)
dist = cv2.distanceTransform(x_inv, cv2.DIST_L2, 5)
ms = 255 - cv2.normalize(dist, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)

plt.subplot(2, 2, 2)
plt.imshow(ms, cmap="gray")
plt.title("Image after applying Distance Transformation")
plt.axis("off")

markers, _ = ndi.label(x)
labels = watershed(ms, markers, mask=x)
```

```
ws = labels == 0
plt.subplot(2, 2, 3)
plt.imshow((x > 0) | ws, cmap="gray")
plt.title("Watershed Segmentation of the image")
plt.axis("off")

colored_segments = label2rgb(labels, bg_label=0)
plt.subplot(2, 2, 4)
plt.imshow(colored_segments)
plt.title("Visualization of different segments with different color")
plt.axis("off")

plt.show()
```


Original Image
Image after applying Distance Transformation
Watershed Segmentation of the image
Visualization of different segments with different color

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import median_filter

def qtdecomp_var(img, threshold=10, min_size=4):
    h, w = img.shape
    S = np.zeros((h, w), dtype=int)

    def split(r0, c0, size):
        block = img[r0:r0+size, c0:c0+size]
        if size <= min_size:
            S[r0:r0+size, c0:c0+size] = size
            return
        if np.var(block) > threshold:
            half = size // 2
            split(r0, c0, half)
            split(r0, c0+half, half)
            split(r0+half, c0, half)
            split(r0+half, c0+half, half)
        else:
            S[r0:r0+size, c0:c0+size] = size

    size = 2**int(np.floor(np.log2(min(img.shape))))
    split(0, 0, size)
    return S

def qtsetblk(blocks, S, dim, values):
    idx = np.argwhere(S == dim)
    for r, c in idx:
        blocks[r:r+dim, c:c+dim] = values[:, :, 0]
    return blocks

image = cv2.imread("/content/figure1.png")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
Ifilt = median_filter(gray, size=(8,8))

S = qtdecomp_var(gray, 10)
Sfilt = qtdecomp_var(Ifilt, 10)

blocks = np.zeros_like(S, dtype=np.uint8)
for dim in [256,128,64,32,16,8,4,2,1]:
    numblocks = np.sum(S == dim)
    if numblocks > 0:
        values = np.ones((dim, dim, numblocks), dtype=np.uint8)
        values[1:dim,1:dim,:] = 0
        blocks = qtsetblk(blocks, S, dim, values)

blocks[-1,:] = 1
blocks[:,-1] = 1
```

```
blocks_filt = np.zeros_like(Sfilt, dtype=np.uint8)
for dim in [128,64,32,16,8,4,2,1]:
    numblocks = np.sum(Sfilt == dim)
    if numblocks > 0:
        values = np.ones((dim, dim, numblocks), dtype=np.uint8)
        values[1:dim,1:dim,:] = 0
        blocks_filt = qtsetblk(blocks_filt, Sfilt, dim, values)

blocks_filt[-1,:] = 1
blocks_filt[:,-1] = 1

plt.figure(figsize=(8,8))
plt.imshow(Ifilt, cmap="gray")
plt.title("Filtered Image (Median + Quadtree Blocks)")
plt.axis("off")
plt.show()
```

Filtered Image (Median + Quadtree Blocks)