Q) Write a program to obtain one-dimensional discrete Fourier transform of a given one-dimensional vector consists of numbers generated randomly.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
import random
import math

size = 15

random_ls = []
for i in range(size):
    random_ls.append(random.randint(0,256))
print("Input: ", end=" ")
print(random_ls)

print("\nTransformed list: ")
transformed_ls = []
for u in range(size):
    real, imag = 0, 0
    for x in range(size):
        A = random_ls[x]
        angle = 2 * math.pi*u*x/size
        real += A * math.cos(angle)
        imag += A * math.sin(angle)
    transformed_ls.append("({0:8.2f}) + ({0:8.2f})i".format(real,imag))
    print(transformed_ls[u])
```

Q) Write a program to obtain two-dimensional discrete fourier transform and its inverse of a gray level image. Also compute the magnitude, phase angle and the power spectrum of the Fourier Transform.

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

img = cv2.imread('gray2.jpg', 0)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

magnitude = cv2.magnitude(dft_shift[:,:,0], dft_shift[:,:,1])
phase_angle = cv2.phase(dft_shift[:,:,0], dft_shift[:,:,1])
power = magnitude**2

power_spectrum = 20*np.log(power)
```

```
inverse = np.fft.ifftshift(dft_shift)
img_back = cv2.idft(inverse)
img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])

fig, axs = plt.subplots(1, 3, figsize=(15, 15))

axs[0].imshow(img, cmap = 'gray')
axs[0].set_title('Input Image')
axs[1].imshow(power_spectrum, cmap = 'gray')
axs[1].set_title('Power Spectrum')
axs[2].imshow(img_back, cmap = 'gray')
axs[2].set_title('Inverse Fourier')

plt.show()
```

Q) Write a program to prove that if f(x,y) is real, then its Fourier transform is conjugate symmetric.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
import random
import math

size = 15

random_ls = []
for i in range(size):
  random_ls.append(random.randint(0,256))
print("Input: ", end=" ")
print(random_ls)


transformed_ls = []
conjugate_ls = []
for u in range(size):
  real, imag, con_real, con_imag= 0, 0, 0, 0
  for x in range(size):
    A = random_ls[x]
    angle = 2 * math.pi*(u)*x/size
    con_angle = -angle
    real += A * math.cos(angle)
    imag += A * math.sin(angle)
    con_real += A * math.cos(con_angle)
    con_imag += A * math.sin(con_angle)
  conjugate_ls.append((con_real,-con_imag))
  transformed_ls.append((real,imag))
```

```
print("Fourier: ", end=" ")
print(transformed_ls)
print("Conjugate: ", end=" ")
print(conjugate_ls)
```

Q)Write a program to program to demonstrate the basics of filtering in the frequency domain (Low-pass filter, High-pass filter)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('peppers_gray.tif', 0)
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

rows, cols = img.shape
crow, ccol = rows//2, cols//2

# High pass filter
fshift = np.fft.fftshift(dft)
fshift[crow-30:crow+30, ccol-30:ccol+30] = [0, 0]
f_ishift = np.fft.ifftshift(fshift)
img_hpf = cv2.idft(f_ishift)
img_hpf = cv2.magnitude(img_hpf[:,:,0], img_hpf[:,:,1])

# Low pass Filter
mask = np.zeros((rows, cols, 2), np.uint8)
mask[crow-30:crow+30, ccol-30:ccol+30] = [1, 1]
fshift = np.fft.fftshift(dft) * mask
f_ishift = np.fft.ifftshift(fshift)
img_lpf = cv2.idft(f_ishift)
img_lpf = cv2.magnitude(img_lpf[:,:,0], img_lpf[:,:,1])

fig, axs = plt.subplots(1, 3, figsize=(15, 15))

axs[0].imshow(img, cmap = 'gray')
axs[0].set_title('Input Image')
axs[1].imshow(img_hpf, cmap = 'gray')
axs[1].set_title('High Pass Filter')
axs[2].imshow(img_lpf, cmap = 'gray')
axs[2].set_title('Low Pass Filter')

plt.show()
```

Q)Write a program to read a gray level image, a color image and a binary image.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

color = cv2.imread('img.jpg')
grayscale = cv2.imread('img.jpg',0)

bw = cv2.imread('img.jpg',0)
row,col = bw.shape

for x in range(row):
	for y in range(col):
		if bw[x][y]<126:
			bw[x][y]=0
		else:
			bw[x][y]=255

plt.subplot(131),plt.imshow(color)
plt.title('Color Image'),plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(grayscale,cmap='gray')
plt.title('Grayscale Image'),plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(bw,cmap='gray')
plt.title('Black and White'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q) Write a program to add two gray levels of same size and display the output image.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray1.jpg',0)
img2 = cv2.imread('gray2.jpg',0)
shape = img.shape
new = np.array(np.zeros(shape))

for x in range(shape[0]):
	for y in range(shape[1]):
		new[x][y]=img[x][y]//2+img2[x][y]//2

plt.imshow(new,cmap='gray')
plt.title('Addition of two images'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q)Write a program to transform 256 gray levels of a gray level image into 8 different gray levels and then multiply 1,2,3,...,8 with 8 different gray levels in decreasing order. Display the output image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray1.jpg',0)

row,col = img.shape
new = np.array(np.zeros((row,col)))

for x in range(row):
	for y in range(col):
		val=(img[x][y])//32
		new[x][y]=((val)*8+4)*(8-val)

plt.subplot(121),plt.imshow(img,cmap='gray')
plt.title('Image'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(new,cmap='gray')
plt.title('Transformed Image'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q) Write a program to reduce the gray level from 256 to 128, 64, 32, 16, 8, 4 and 2 of a monochrome image according to user input.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray1.jpg',0)

row,col = img.shape
new = np.array(np.zeros((row,col)))

n = int(input('Enter the required number of bits:'))
div = 2**(8-n)

for x in range(row):
	for y in range(col):
		val=(img[x][y])//div
		new[x][y]=(val

plt.subplot(121),plt.imshow(img,cmap='gray')
plt.title('Image'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(new,cmap='gray')
plt.title('Transformed Image'),plt.xticks([]),plt.yticks([])
```

```
plt.show()
```

Q) Write a program to zoom and shrink a gray level image at a desired level. To achieve zooming and shrinking, one can apply oversampling and undersampling to the gray level image.

```
import matplotlib.pyplot as mp
import cv2
import numpy as np

img = cv2.imread('gray2.jpg',0)

scale = int(input("Enter the scale:"))
height, width = img.shape
new_h = int(scale*height)
new_w = int(scale*width)

new_img = np.zeros(shape=(new_h,new_w))
new_img2 = np.zeros(shape=(height//2,width//2))


for i in range(new_h):
        for j in range(new_w):
                k = int(i/scale)
                l = int(j/scale)
                new_img[i][j] = img[k][l]

for i in range(height//2):
        for j in range(width//2):
                k = int(i*scale)
                l = int(j*scale)
                new_img2[i][j] = img[k][l]



mp.subplot(131),mp.imshow(img,cmap='gray')
mp.title('Original')
mp.subplot(132),mp.imshow(new_img,cmap='gray')
mp.title('Zoomed')
mp.subplot(133),mp.imshow(new_img2,cmap='gray')
mp.title('Shrinked')
mp.show()
```

Q) Write a program to decompose an image into eight 1-bit planes ranging from 0^{th} bit to 7^{th} bit plane and set 0 to most significant bits (first 4 bits). Then subtract the resultant image from the input image.

```
import cv2
```

```python
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray2.jpg',0)

row,col = img.shape

new = np.array(np.zeros((row,col)))
eight = np.array(np.zeros((row,col)))
seven = np.array(np.zeros((row,col)))
six = np.array(np.zeros((row,col)))
five = np.array(np.zeros((row,col)))

for x in range(row):
	for y in range(col):
		val=img[x][y]
		if val>=128:
			eight[x][y]=128
			val=val-128
		if val>=64:
			seven[x][y]=64
			val=val-64
		if val>=32:
			six[x][y]=32
			val=val-32
		if val>=16:
			five[x][y]=16
			val=val-16

new = img - eight - seven - six - five
print(new)

plt.subplot(121),plt.imshow(img,cmap='gray')
plt.title('Image'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(new,cmap='gray')
plt.title('Transformed Image'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q) Write a program to apply a sequence of following filtering operation on a niosy finger print image with a structuring element of dimension 3*3 whose elements are all 1.
a)erosion
b)opening
c)Closing
d)dilation

Q)Write a program to extract the boundary of an object given in an image with the help of an appropriate structuring elements.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import copy

def threshold(img):

	row,col = img.shape

	for x in range(row):
		for y in range(col):
			if img[x][y]<126:
				img[x][y]=0
			else:
				img[x][y]=255

	return img

def erosion(img):

	eroded = copy.deepcopy(img)
	row,col = eroded.shape

	movements = [[0,0],[0,1],[1,0],[1,1],[-1,-1],[-1,1],[1,-1],[0,-1],[-1,0]]
	for x in range(1,row-1):
		for y in range(1,col-1):
			flag=1

			for i in movements:
				if img[x+i[0]][y+i[1]]==0:
					flag=0
					break

			eroded[x][y]=255*flag
```

```python
        return eroded

def dilation(img):

        dilated = copy.deepcopy(img)
        row,col = dilated.shape

        movements = [[0,0],[0,1],[1,0],[1,1],[-1,-1],[-1,1],[1,-1],[0,-1],[-1,0]]
        for x in range(1,row-1):
                for y in range(1,col-1):
                        flag=0

                        for i in movements:
                                if img[x+i[0]][y+i[1]]==255:
                                        flag=1
                                        break

                        dilated[x][y]=255*flag

        print(dilated)
        return dilated


img = cv2.imread('img3.jpg',0)
img = threshold(img)

eroded = erosion(img)
dilated = dilation(img)
opening = dilation(erosion(img))
closing = dilation(erosion(img))
bound = dilated-eroded

plt.subplot(231),plt.imshow(img,cmap='gray')
plt.title('Original'),plt.xticks([]),plt.yticks([])
plt.subplot(232),plt.imshow(eroded,cmap='gray')
plt.title('Erosion'),plt.xticks([]),plt.yticks([])
plt.subplot(233),plt.imshow(dilated,cmap='gray')
plt.title('Dilation'),plt.xticks([]),plt.yticks([])
plt.subplot(234),plt.imshow(bound,cmap='gray')
plt.title('Boundary'),plt.xticks([]),plt.yticks([])
plt.subplot(235),plt.imshow(opening,cmap='gray')
plt.title('Opening'),plt.xticks([]),plt.yticks([])
plt.subplot(236),plt.imshow(closing,cmap='gray')
plt.title('Closing'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q) Write a program to estimate a noise function in a noisy image by displaying the histogram of a given input image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('gray1.jpg',0)
row, col = img.shape

gauss = np.random.normal(20,10,(row,col))
rayleigh = np.random.rayleigh(20,(row,col))
gamma = np.random.gamma(1, 20, (row,col))
exponential = np.random.exponential(10, (row,col))
uniform = np.random.uniform(2, 12, (row,col))


noisy = img + gauss
smooth_part = noisy[:100, :100]


plt.subplot(321),plt.hist(rayleigh.ravel(),256,[0,256])
plt.title('Rayleigh Distribution Image'), plt.xticks([]), plt.yticks([])
plt.subplot(322),plt.hist(gauss.ravel(),256,[0,256])
plt.title('Gauss Distribution'), plt.xticks([]), plt.yticks([])
plt.subplot(323),plt.imshow(img,cmap = 'gray')
plt.title('Input image'), plt.xticks([]), plt.yticks([])
plt.subplot(325),plt.hist(img.ravel(),256,[0,256])
plt.title('Input Image Histogram'), plt.xticks([]), plt.yticks([])
plt.subplot(324),plt.hist(noisy.ravel(),256,[0,256])
plt.title('Noisy Image Histogram'), plt.xticks([]), plt.yticks([])
plt.subplot(326),plt.hist(smooth_part.ravel(),256,[0,256])#; plt.show()
plt.title('Estimated Noise Distribution'), plt.xticks([]), plt.yticks([])
plt.show()
```

Q) Write a program to detect the points in the given gray level image with a suitable mask

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('points.jpeg',0)
row,col = img.shape

laplacian=np.array([[-1,-1,-1],[-1, 8, -1],[-1, -1, -1]],np.int32)

new = np.zeros((row,col))
new = cv2.filter2D(img,-1,laplacian)

for x in range(row):
    for y in range(col):
        if new[x][y] < 5:
            new[x][y]=0
        else:
            new[x][y]=255

print(new)
plt.imshow(new, cmap='gray')
plt.title('Extracted Points')
plt.show()
```

Q) Write a program to detect the lines in horizontal and vertical directions in the given image with suitable masks.

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

img = cv2.imread("house.jpeg",0)

vertical_kernel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]],np.int32)
horizontal_kernel = np.array([[-1,-2,-1],[0,0,0],[1,2,1]],np.int32)

v_img = cv2.filter2D(img,-1,vertical_kernel)
h_img = cv2.filter2D(img,-1,horizontal_kernel)

plt.subplot(121),plt.imshow(v_img,cmap='gray')
plt.title('Vertical Lines'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(h_img,cmap='gray')
plt.title('Horizontal Lines'),plt.xticks([]),plt.yticks([])
plt.show()
```

Q) Write a program to detect the edges in the given image with a suitable edge detection operator and then further detect the edge

```
import cv2
import numpy as np

from scipy import ndimage
from math import sqrt

def magnitude(img1,img2,t):
    newimage=img1[:]
    for i in range(img1.shape[0]):
        for j in range(img1.shape[1]):
            newimage[i][j]=sqrt((img1[i][j]**2+img2[i][j]**2))
            if newimage[i][j]<t:
                newimage[i][j]=0
            else:
                newimage[i][j]=255

    return newimage


image =cv2.imread("house.jpeg",0)
# print(image)
avg=np.array([[1,1,1],
    [1, 1, 1],
    [1, 1, 1]])
# avg=np.array(avg)

img1=ndimage.convolve(image,avg , mode="constant", cval=0)
img1=img1/9
cv2.imshow("avg",img1)
vertical_kernel=np.array([[-1,0,1],
                          [-2,0,2],
                          [-1,0,1]])
horizontal_kernel=np.array([[-1,-2,-1],
                          [0,0,0],
                          [1,2,1]])


img1=ndimage.convolve(img1, vertical_kernel, mode="constant", cval=0)
img2=ndimage.convolve(img1, horizontal_kernel, mode="constant", cval=0)
```

```
img4=magnitude(img1,img2,50)
cv2.imshow("edges",img4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q)Write a program to find the alignment of a set of edge points using hough transform.

```
import cv2
import numpy as np

img = cv2.imread('house.jpeg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, bw_img = cv2. threshold(gray,127,255,cv2. THRESH_BINARY)
edges = cv2.Canny(bw_img,50,150,apertureSize = 3)

lines = cv2.HoughLines(edges,1,np.pi/180,200)
for rho,theta in lines[0]:
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))

    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

plt.imshow(img)
plt.title('HoughLines')
plt.show()
```

Q)Convolute using the following mask

a) **1/9 [[1,1,1],[1,1,1],[1,1,1]]**
b) **Gx = [[-1,0,1],[-2,0,2],[-1,0,1]] , Gy = [[-1,-2,-1],[0,0,0],[1,2,1]]**
c) **Gx = [[-1,0,1],[-1,0,1],[-1,0,1]] , Gy = [[-1,-1,-1],[0,0,0],[1,1,1]]**
d) **Gx = [[1,0],[0,-1]], Gy = [[0,1],[-1,0]]**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def convolve3(image, mask):
    row, col = image.shape
    res = np.zeros((row, col), dtype=np.uint8)
    for x in range(1, row-1):
        for y in range(1, col-1):
            res[x][y] = mask[0][0]*img[x-1][y-1] + mask[0][1]*img[x-1][y] +
mask[0][2]*img[x-1][y+1] + mask[1][0]*img[x][y-1] +                mask[1][1]*img[x][y] +
mask[1][2]*img[x][y+1] +                mask[2][0]*img[x+1][y-1] + mask[2][1]*img[x+1][y] +
mask[2][2]*img[x+1][y+1]

    return res

def convolve2(image, mask):
    row, col = image.shape
    res = np.zeros((row, col), dtype=np.uint8)
    for x in range(0, row-1):
        for y in range(0, col-1):
            res[x][y] = mask[0][0]*img[x][y] + mask[0][1]*img[x][y+1] +
mask[1][0]*img[x+1][y] + mask[1][1]*img[x+1][y+1]

    return res

img = cv2.imread('wirebond_mask.tif', 0)

mask1 = np.array([[1,1,1], [1,1,1], [1,1,1]], np.int32)
res1 = convolve3(img, mask1)
res1 = res1//9

mask2_hor = np.array([[-1,0,1], [-2,0,2], [-1,0,1]], np.int32)
mask2_ver = np.array([[1,2,1], [0,0,0], [-1,-2,-1]], np.int32)
res2_x = convolve3(img, mask2_hor)
res2_y = convolve3(img, mask2_ver)

mask3_hor = np.array([[-1,0,1], [-1,0,1], [-1,0,1]], np.int32)
mask3_ver = np.array([[1,1,1], [0,0,0], [-1,-1,-1]], np.int32)
res3_x = convolve3(img, mask3_hor)
res3_y = convolve3(img, mask3_ver)
```

```
mask4_hor = np.array([[1,0], [0,-1]], np.int32)
mask4_ver = np.array([[0,1], [-1,0]], np.int32)
res4_x = convolve2(img, mask4_hor)
res4_y = convolve2(img, mask4_ver)

fig0, axs0 = plt.subplots(1, 3, figsize=(15, 10))
axs0[0].imshow(res1, cmap='gray')
axs0[0].set_title("Mean filter")
axs0[1].imshow(res2_x, cmap='gray')
axs0[1].set_title("Sobel filter X")
axs0[2].imshow(res2_y, cmap='gray')
axs0[2].set_title("Sobel filter Y")

fig1, axs1 = plt.subplots(1, 4, figsize=(15, 15))
axs1[0].imshow(res3_x, cmap='gray')
axs1[0].set_title("Prewitt filter X")
axs1[1].imshow(res3_y, cmap='gray')
axs1[1].set_title("Prewitt filter Y")
axs1[2].imshow(res4_x, cmap='gray')
axs1[2].set_title("2x2 Gradient filter X")
axs1[3].imshow(res4_y, cmap='gray')
axs1[3].set_title("2x2 Gradient filter Y")

plt.show()
```

Q)Write a program to enhance a low contrast image using different image enhancement techniques.

```
import cv2
import numpy as np
import math
import copy

img = cv2.imread("gray1.jpg",0)
neg = copy.deepcopy(img)
log = copy.deepcopy(img)
power = copy.deepcopy(img)
equalize = copy.deepcopy(img)

equalize = cv2.equalizeHist(equalize)


(height, width) = img.shape
cv2.imshow('image', img)

for i in range(width):
        for j in range(height):
                neg[i][j]= 255-neg[i][j]
                log[i][j]=105*math.log(1 + (img[i][j]),10)
                power[i][j]=1*img[i][j]**0.7

cv2.imshow('negative', neg)
cv2.imshow('log', log)
cv2.imshow('power', power)
cv2.imshow("equalize",equalize)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q) Write a program to enhance a low contrast gray level image using histogram equalization and histogram matching (specification) and then analyse the resultant images.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt


def find_nearest_above(my_array, target):
    diff = my_array - 7
    mask = np.ma.less_equal(diff, -1)

    if np.all(mask):
        c = np.abs(diff).argmin()
        return c
```

```python
    masked_diff = np.ma.masked_array(diff, mask)
    return masked_diff.argmin()


def hist_match(original, specified):

    oldshape = original.shape
    original = original.ravel()
    specified = specified.ravel()

    s_values, bin_idx, s_counts = np.unique(
        original, return_inverse=True, return_counts=True)
    t_values, t_counts = np.unique(specified, return_counts=True)

    s_q = np.cumsum(s_counts).astype(np.float64)
    s_q /= s_q[-1]

    t_q = np.cumsum(t_counts).astype(np.float64)
    t_q /= t_q[-1]

    sour = np.around(s_q * 255)
    temp = np.around(t_q * 255)

    b = []
    for data in sour:
        b.append(find_nearest_above(temp, data))
    b = np.array(b, dtype='uint8')

    return b[bin_idx].reshape(oldshape)


def main():
    source = cv2.imread('ship1.jpg', 0)
    template = cv2.imread('boat1.jpg', 0)

    equalized = cv2.equalizeHist(source)
    matched = hist_match(source, template)

    plt.subplot(131),plt.imshow(source,cmap='gray')
    plt.title('Source')
    plt.subplot(132),plt.imshow(equalized,cmap='gray')
    plt.title('Equalized')
    plt.subplot(133),plt.imshow(matched,cmap='gray')
    plt.title('Matched')
    plt.show()


if __name__ == '__main__':
```

```
    main()
```

Q) Write a program to find 4, 8 and m adjacent among the pixels for V = {1} in the following binary image. Here, V is a criterion based on which the adjacency can be measured.

```
import numpy as np

def N4(arr, i, j, V):
    m, n = arr.shape
    s = set()

    if arr[i][j] in V:
        if i-1 >= 0 and arr[i-1][j] in V:
            s.add((i-1, j))
        if i+1 < m and arr[i+1][j] in V:
            s.add((i+1, j))
        if j-1 >= 0 and arr[i][j-1] in V:
            s.add((i, j-1))
        if j+1 < n and arr[i][j+1] in V:
            s.add((i, j+1))

    if len(s) > 0:
        return s

def ND(arr, i, j, V):
    m, n = arr.shape
    s = set()

    if i-1 >= 0 and j-1 >= 0 and arr[i-1][j-1] in V:
        s.add((i-1, j-1))
    if i+1 < m and j+1 < n and arr[i+1][j+1] in V:
        s.add((i+1, j+1))
    if i-1 >= 0 and j+1 < n and arr[i-1][j+1] in V:
        s.add((i-1, j+1))
    if i+1 < m and j-1 >= 0 and arr[i+1][j-1] in V:
        s.add((i+1, j-1))

    if len(s) > 0:
        return s

def N8(arr, i, j, V):
    m, n = arr.shape
    s = set()

    if arr[i][j] in V:
        n4 = N4(arr, i, j, V) or []
        nd = ND(arr, i, j, V) or []
        for ng in n4:
            s.add(ng)
        for ng in nd:
            s.add(ng)

    if len(s) > 0:
```

```python
        return s

def Nm(a, i, j, v):
    m, n = arr.shape
    s = set()

    if arr[i][j] in V:
        n4 = N4(arr, i, j, V) or []
        for ng in n4:
            s.add(ng)
        nd = ND(arr, i, j, V) or []
        for ng in nd:
            nd_q = ND(arr, ng[0], ng[1], V)
            if len(nd.intersection(nd_q)) == 0:
                s.add(ng)

    if len(s) > 0:
        return s

arr = np.array([[0, 1, 1 , 0, 1],
       [1, 1, 0 , 1, 1],
       [1, 0, 1 , 1, 1],
       [0, 1, 0 , 1, 1],
       [0, 1, 1 , 1, 0]])

m, n = arr.shape

V = [1]

print("4 Neighbours:")
for i in range(m):
    for j in range(n):
        print(N4(arr, i, j, V), end=", ")
    print()

print("\n8 Neighbours:")
for i in range(m):
    for j in range(n):
        print(N8(arr, i, j, V), end=", ")
    print()

print("\nM-adjacent Neighbours:")
for i in range(m):
    for j in range(n):
        print(Nm(arr, i, j, V), end=", ")
    print()
```