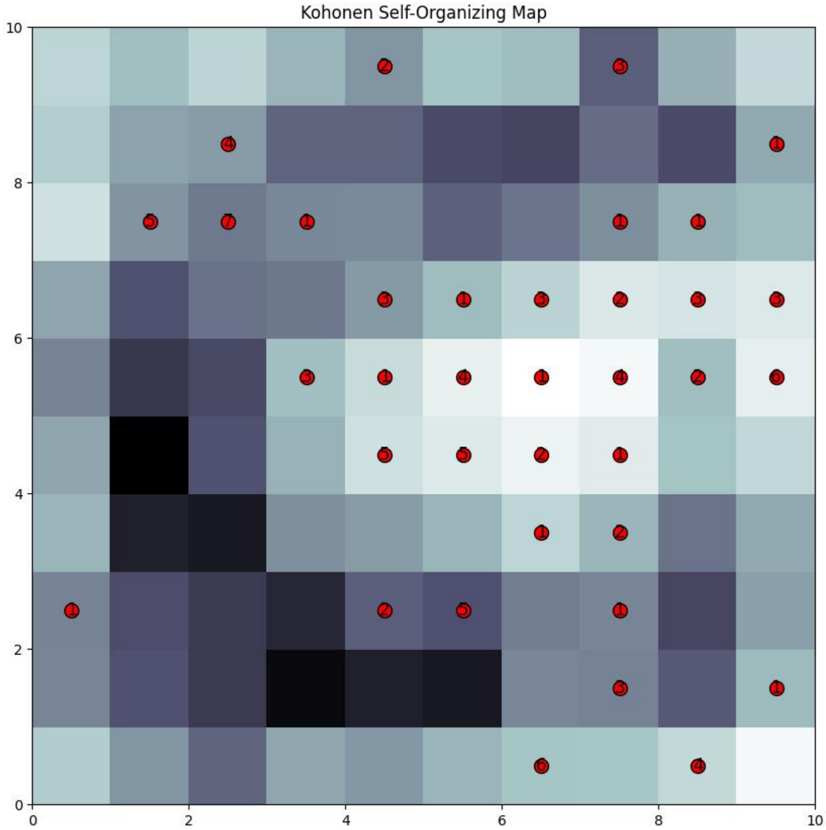
Practical 6A

```python
# A. Aim: Kohonen Self organizing map using python
from minisom import MiniSom
import numpy as np
import matplotlib.pyplot as plt
# Generate sample data (you can replace this with your own dataset)
data = np.random.rand(100, 2)
# Define the SOM dimensions (grid size) and the input dimensionality
grid_size = (10, 10)
input_dim = 2
# Initialize the SOM
som = MiniSom(grid_size[0], grid_size[1], input_dim, sigma=1.0, learning_rate=0.5)
# Train the SOM with the data
som.train(data, 100) # 100 iterations, you can adjust this number
# Create a map of the winning neurons
win_map = som.win_map(data)
# Visualize the SOM
plt.figure(figsize=(10, 10))
plt.pcolor(som.distance_map().T, cmap='bone_r') # Plot the distance map
# Mark the winning neurons on the map
for position, values in win_map.items():
 x, y = position
 plt.scatter(x + 0.5, y + 0.5, s=100, color='red', edgecolor='black')
 plt.text(x + 0.5, y + 0.5, str(len(values)), fontsize=12, ha='center', va='center')
plt.title('Kohonen Self-Organizing Map')
plt.show()
```



Practical 6B

#B. Aim: Adaptive resonance theory.

```
import numpy as np
class ART1:
    def __init__(self, num_input, vigilance_parameter):
        self.num_input = num_input
        self.vigilance_parameter = vigilance_parameter
        self.weights = np.random.rand(num_input)
        self.reset()
    def reset(self):
        self.activation = 0
        self.category = -1
    def normalize(self, input_pattern):
        return input_pattern / input_pattern.sum()
    def match(self, input_pattern):
        return np.dot(self.weights, input_pattern)
    def train(self, input_pattern):
        normalized_pattern = self.normalize(input_pattern)
        self.activation = self.match(normalized_pattern)
        if self.activation >= self.vigilance_parameter:
            self.category = np.argmax(normalized_pattern)
        else:
            self.category = -1
# Example usage
if __name__ == "__main__":
    num_input = 4
    vigilance_parameter = 0.5
    art_network = ART1(num_input, vigilance_parameter)
    # Define training patterns and categories
    training_patterns = np.array([[0.1, 0.4, 0.2, 0.3],
                                  [0.6, 0.3, 0.4, 0.1],
                                  [0.2, 0.2, 0.3, 0.3]])
    for i, pattern in enumerate(training_patterns):
        art_network.train(pattern)
        print(f"Pattern {i + 1}: Category {art_network.category}")
        art_network.reset()
    test_pattern = np.array([0.5, 0.2, 0.3, 0.1])
    art_network.train(test_pattern)
    if art_network.category != -1:
        print(f"Test pattern belongs to Category {art_network.category}")
    else:
        print("Test pattern does not belong to any category")
```

Output:

```
Pattern 1: Category 1
Pattern 2: Category 0
Pattern 3: Category 2
Test pattern belongs to Category 0
```