1: The First Problem

(a) Algorithm:

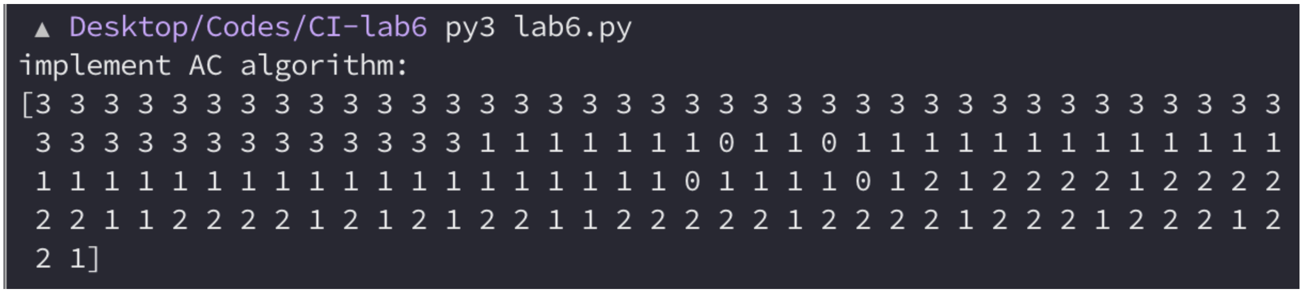
```
import math
import numpy as np

def cal_dist(point1: np.ndarray, point2: list) -> float:
    distance = 0.0
    for a, b in zip(point1, point2):
        distance += math.pow(a - b, 2)
    return math.sqrt(distance)


class Node(object):
    def __init__(self,
                 cent,
                 left=None,
                 right=None,
                 distance=-1,
                 tag=None,
                 count=1):
        self.cent = cent
        self.left = left
        self.right = right
        self.distance = distance
        # tag calculated node
        self.tag = tag
        self.count = count


class AC(object):
    def __init__(self, k=1):
        assert k > 0
        self.k = k
        self.labels = None

    def fit(self, x):
        nodes = [Node(cent=v, tag=i) for i, v in enumerate(x)]
        distances = {}
        point_num, future_num = np.shape(x)
        self.labels = [-1] * point_num
        current_tag = -1
        while len(nodes) > self.k:
            min_dist = math.inf
            nodes_len = len(nodes)
            closest_part = None
            for i in range(nodes_len - 1):
                for j in range(i + 1, nodes_len):
                    d_key = (nodes[i].tag, nodes[j].tag)
                    if d_key not in distances:
                        distances[d_key] = cal_dist(nodes[i].cent,
                                                    nodes[j].cent)
                    d = distances[d_key]
                    if d < min_dist:
                        min_dist = d
                        closest_part = (i, j)
            # merge
            part1, part2 = closest_part
            node1, node2 = nodes[part1], nodes[part2]
            new_cent = [
                (node1.cent[i] * node1.count + node2.cent[i] * node2.count) /
                (node1.count + node2.count) for i in range(future_num)
            ]
            new_node = Node(cent=new_cent,
                            left=node1,
                            right=node2,
                            distance=min_dist,
                            tag=current_tag,
                            count=node1.count + node2.count)
            current_tag -= 1
            del nodes[part2], nodes[part1]
            nodes.append(new_node)
        self.nodes = nodes
        self.cal_label()

    def cal_label(self):
        for i, node in enumerate(self.nodes):
            self.order(node, i)

    def order(self, node: Node, label):
        if node.left is None and node.right is None:
            self.labels[node.tag] = label
        if node.left:
            self.order(node.left, label)
        if node.right:
            self.order(node.right, label)
```

2: The Second Problem

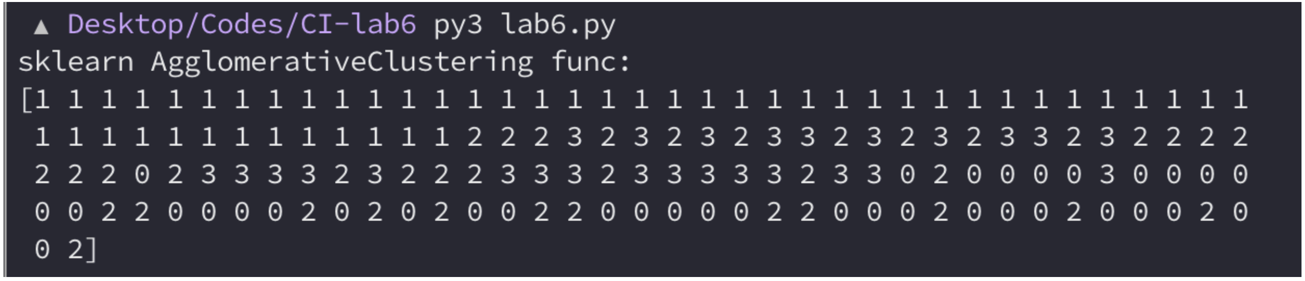
(a) Algorithm:

```
from sklearn import datasets
from sklearn import cluster

iris = datasets.load_iris()
ac = AC(4)
ac.fit(iris.data)
print(np.array(ac.labels))

sk = cluster.AgglomerativeClustering(4)
sk.fit(iris.data)
print(sk.labels_)
```

(b) Output: Self-implemented AC algorithm VS AgglomerativeClustering from sklearn



```
 ▲ Desktop/Codes/CI-lab6 py3 lab6.py
implement AC algorithm:
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
```



```
 ▲ Desktop/Codes/CI-lab6 py3 lab6.py
sklearn AgglomerativeClustering func:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 2 3 2 3 2 3 3 2 3 2 2 2 2
 2 2 2 0 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 3 2 3 3 0 2 0 0 0 0 3 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 2 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```