Problem 1: Consider partial digest ***L* = {1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 6, 7, 7, 7, 8, 9, 10, 11, 12 }**
Implement an algorithm to solve Partial Digest problem for L (i.e. find X such that ΔX = L).

Solution:

```
X = []

L = [1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 6, 7, 7, 7, 8, 9, 10, 11, 12]

max_value = 0

def partialDigest(L):
    global X, max_value
    max_value = max(L)
    L.remove(max_value)
    X = [0, max_value]
    place(L, X)

def place(L, X):

    if not L:
        print("Result is:  ", X)
        return

    y = max(L)

    if Subset(y, X, L):
        X.append(y)
        removeElmnt(y, X, L)
        place(L, X)
        if y in X:
            X.remove(y)
        L.extend(Difference(y, X))

    if Subset(abs(max_value-y), X, L):
        X.append(abs(max_value-y))
        removeElmnt(abs(max_value-y), X, L)
        place(L, X)
        if abs(max_value-y) in X:
            X.remove(abs(max_value-y))
        L.extend(Difference(abs(max_value-y), X))

    return

def Difference(y, X):
    diff = []
    for i in X:
        diff.append(abs(y-i))
    return diff

def removeElmnt(y, X, L):
    for i in X:
        if abs(y - i) in L:
            L.remove(abs(y - i))

def Subset(y, X, L):
        for i in X:
            if abs(y-i) not in L:
                return False
        return True

def main():
    partialDigest(L)

main()
```

Problem 2: Implement any dynamic programming algorithm for pair-wise global alignment of sequences S1= {CTCGCAGC} and S2={CATTCAG}. Give the final alignment table given that E(a,-)=-2, E(-,b)=-2 and E(a,b)=5 for match E(a,b)=-2 for mismatch.

solution:

```
import numpy as np

S1 = 'CTCGCAGC'
S2 = 'CATTCAG'


length_of_S1 = len(S1)
length_of_S2 = len(S2)

#Create Matrices
#matrix = [[0 for i in range(length_of_v+1)] for j in range(length_of_w+1)]
direction = [[0 for i in range(length_of_S2+1)] for j in range(length_of_S1+1)]
matrix = np.zeros((len(S1)+1,len(S2)+1))


match = 5
mismatch = -2
indel = -2

# Initialisation
for i in range(length_of_S2+1):
    matrix[0][i] = i* indel
```

```
for i in range(length_of_S1+1):
    matrix[i][0] = i* indel

print(matrix)
#Matrix Filling
for i in range(1,length_of_S1+1):
    for j in range(1,length_of_S2+1):

        if (S1[i-1] == S2[j-1]):
            #print(i,j)
            diagonal_value = matrix[i-1][j-1] + match
            upper_value = matrix[i-1][j] + indel
            left_value = matrix[i][j-1] + indel
            maxx = max(diagonal_value, upper_value, left_value)
            if(maxx == diagonal_value):
                direction[i][j] = 'd'
            if(maxx == upper_value):
                direction[i][j] = 'u'
            if(maxx ==left_value):
                direction[i][j] = 'l'
            matrix[i][j] = max(diagonal_value,upper_value,left_value)
            #print(diagonal_value,upper_value,left_value)
        elif (S1[i-1] != S2[j-1]):
            diagonal_value = matrix[i - 1][j - 1] + mismatch
            upper_value = matrix[i - 1][j] + indel
            left_value = matrix[i][j - 1] + indel
            maxx = max(diagonal_value, upper_value, left_value)
            if (maxx == diagonal_value):
                direction[i][j] = 'd'
            if (maxx == upper_value):
                direction[i][j] = 'u'
            if (maxx == left_value):
                direction[i][j] = 'l'
            matrix[i][j] = max(diagonal_value, upper_value, left_value)
            #print(i,j,diagonal_value, upper_value, left_value)

#printing Matrix
print ("          c     A     T     T     C     A     G")
row_labels = [' ','C','T','C','G','C','A','G','C']

for row_label, row in zip(row_labels, matrix):
    print ('%s [%s]' % (row_label, ' '.join('%05s' % i for i in row)))

#for i in range(length_of_w+1):
 #   print(matrix[i])
```

```
#traceback
seq1 = ''
seq2 = ''

i= 8
j=7
while(i>0 and j>0):
    if(direction[i][j] == 'd'):
        seq1 = S2[j-1]+seq1
        seq2 = S1[i-1] + seq2
        i = i-1
        j = j-1
    elif(direction[i][j] == 'u'):
        seq1 = '-'+seq1
        seq2 = S1[i-1] + seq2
        i = i-1
    else:
        seq2 = '-' + seq2
        seq1 = S2[j - 1] + seq1
        j = j - 1

#printing final alignment and final score,
total_match = 0
total_mismatch = 0
for i in range(9):
    if(seq2[i] == seq1[i]):
        total_match = total_match + 1
    else:
        total_mismatch = total_mismatch + 1
#print(total_match)
#print(total_mismatch)

print(S1 + ', after alignment: '+seq1)
print(S2 + ', after alignment: '+seq2)

score = mismatch*total_mismatch + match*total_match

print('Final Score: ' + str(score))
```

Problem 04: Implement an algorithm to find a de Bruijn graph for the sequence **GACTTACGTACT** with k= 3 and generate the corresponding Eulerian walk.

solution:

```
def de_Bruijn(sequence, k):
    edges = []
    nodes = set()
    eulerian_walk = ''
    for i in range(len(sequence) - k + 1):
        eulerian_walk = eulerian_walk + sequence[i:i+k-1] + '->'
        edges.append((sequence[i:i+k-1], sequence[i+1:i+k]))
        nodes.add(sequence[i:i+k-1])
        nodes.add(sequence[i+1:i+k])
    eulerian_walk = eulerian_walk[:-2]
    #print(eulerian_walk)
    return nodes, edges, eulerian_walk

def main():
    L = "GACTTACGTACT"
    k = 3
    nodes, edges, eulerian_walk = de_Bruijn("GACTTACGTACT", 3)
    print("nodes: ",nodes)  #total nodes in de Bruijn Graph.
    #print("edges: ",edges) #edges from one nodes to another node.
    print("Eulerian walk: ",eulerian_walk)

main()
```

Problem 5: Implement agglomerative algorithm with single link distance measure and produce a dendogram tree for the following single continuous feature.

solution:

```
 import numpy as np

def updateMatrix(distanceMatrix, i, minValTrack):
    newDisMatrix = np.zeros((i,i))
    position = findMinValueposition(distanceMatrix)
    minValTrack[i][0] = position[0]
    minValTrack[i][1] = position[1]

    x = 0
    y = 1
    for j in range(len(distanceMatrix)):
        if j == position[1]:
            continue
        y = x + 1
        for k in  range(j+1,len(distanceMatrix)):
            if k == position[1]:
                continue

            if j == position[0]:
                temp = min(distanceMatrix[position[0]][k], distanceMatrix[position[1]][k])
                newDisMatrix[x][y] = newDisMatrix[y][x] = temp
            else:
                newDisMatrix[x][y] = newDisMatrix[y][x] = distanceMatrix[j][k]
            y = y+1
        newDisMatrix[x][x] = 0
        x = x+1

    return newDisMatrix

def printMatrix(mat):
    for i in range(len(mat)):
        for j in range(len(mat)):
            print(mat[i][j], end =" ")
        print("\n")
    print("\n")
    return

def build_DistanceMatrix(feature):
    length = len(feature)
    distanceMatrix = np.zeros((length,length))

    for i in range(length):
        for j in range(length):
            if i == j:
                distanceMatrix[i][j] = 0
            else:
                distanceMatrix[i][j] = distanceMatrix[j][i] = abs(feature[i] - feature[j])

    return distanceMatrix

def findMinValueposition(matrix):
    miniVal = 100
    position = np.zeros(2)

    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if miniVal > matrix[i][j]:
                miniVal = matrix[i][j]
                position[0] = i
                position[1] = j

    return position

def main():
    feature = [1,2,5,6,8]
    Gene = ['a', 'b', 'c', 'd', 'e']
    minValTrack = [[0 for i in range(2)] for j in range(len(Gene)+1)]

    distanceMatrix = build_DistanceMatrix(feature)
    cnt = 1
    for i in range(len(feature), 1, -1):
        print("Number of step "+str(cnt))
        printMatrix(distanceMatrix)
        distanceMatrix = updateMatrix(distanceMatrix, i,  minValTrack)
        cnt = cnt + 1
    print(print("Number of step "+str(cnt)))
    printMatrix(distanceMatrix)

main()
```