CSE321-HW3

1. Derive recurrence relations of the following algorithms and solve them to decide the complexity of the algorithms. Which algorithm would you prefer for the same problem, elaborate your answer.

```
Algorithm alg1(L[0..n-1])
    if (n==1) return L[0]
    else
        tmp = alg1(L[0..n-2])
        if (tmp<=L[n-1]) return tmp
        else return L[n-1]
```

$T(n) = T(n-1) + 1$
$= T(n-2) + 2$
$\vdots$
$= T(1) + n = O(n)$

```
Algorithm alg2(X[l..r])
    if (l==r) return X[l]
    else
        flr = floor((l+r)/2)
        tmp1 = alg2(X[l..flr])
        tmp2 = alg2(X[flr+1..r])
        if (tmp1<=tmp2) return tmp1
        else return tmp2
```

$T(n) = 2T(n/2) + 1$

Master Theorem:

$f(n) = 1 = O(n^{\log_2 2 - \varepsilon})$

$\Rightarrow T(n) = O(n)$

They have the same asymptotic time complexity so they could both be used. But I would prefer the first one since it uses less memory and has less constant time operations.

You are given a polynomial $p(x)$ like

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0,$$

and you are supposed to write a brute-force algorithm for computing the value of the polynomial at a given point x_0. Analyze the complexity of your brute-force algorithm, and discuss whether it is possible to design an algorithm that has better complexity. Don't forget to implement your algorithm in Python.

```
function bruteForcePolynomial(coefficients,x)
    result = 0
    order = coefficients.length-1
    for coefficient in coefficients
        power=1
        for i in range 0 to order do: //pow(x,order)
            power = power * x
        result+=coefficient*power
        order=order-1
    return result
```

$O(n^2)$

```
function smartPolynomial(coefficients,x)
    res = coefficients[coefficients.length-1]
    power = 1
    for i in reverse range coefficients.length-1 to 0 do
        power = power * x
        res += coefficients[i] * power
    return res
```

$\Theta(n)$

3. You are asked to design a brute-force algorithm that counts the number of substrings that start with a specific letter and end with another letter in a given text. For example, let the first letter be 'X' and the last letter be 'Z', there are four such substrings in TXZXXJZWX. Write your brute-force algorithm, analyze its complexity, and implement in Python.

```
function countSubstr(string,start,end):
    count=0
    for i in range from 0 to string.length do
        substr = emptyString
        for j in range from i to string.length do
            append(substr,string[j])
            if(substr[0]==start and substr[substr.length-1]==end):
                count+=1
    return count
```

$T(n) = O(n^2)$

4. A metric space consists of a set and a distance function. We are given a metric space that is made up of a set of n points in k-dimensional space and Euclidean distance function. Design a brute-force algorithm that gives the distance between the closest pair of points, and find the complexity of the algorithm.

```
function closestPair(points[0..n],k):
    shortestDist = float.INFINITY
    for int i from 0 to n do:
        dist1= euclidianDist(points[i],k)
        for int j from i to n do:
            temp =abs(dist1 - euclidianDist(points[j],k))
            if temp < shortestDist:
                shortestDist = temp
    return temp
```

$T(n) = O(n^2k)$

5. Profit rates of many companies have changed due to the epidemic. XYZ is a retail company with many branches on the Marmaray line. The management of the company is trying to identify the regions where they make the most profit. For this purpose, they gather the profit-loss rates of their retail shops to the table. Entries on the table are sorted in Marmaray station order.

(a) Write a brute-force algorithm that can find **the most profitable cluster**, provided that the cluster must contain a consecutive region. Explain the time complexity of the algorithm. For example, the most profitable cluster is (C-D-E-F) on table below.

```
function bruteMostProfitableCluster(branches):

    mostProfitable = []
    maxProfit = -math.inf

    for i from 0 to branches.length do
        tempProfit = 0
        tempList = []

        for j from i to branches.length do
            tempList.append(branches[j][0])
            tempProfit += branches[j][1]

            if tempProfit > maxProfit:
                maxProfit = tempProfit
                mostProfitable = tempList.copy()

    return mostProfitable
```

(b) Write a divide and conquer algorithm that **finds the maximum profit** that belongs to the most profitable cluster (the cluster must contain a consecutive region), analyze its complexity, and write the Python code of the algorithm. For example, the maximum profit is 14 (C-D-E-F) on table below.

```
function findClusterProfit(branches, l, m, h):
    temp = 0
    left_profit = -math.inf
    #calculate left
    for i from m to l-1 (backwards) do
        temp = temp + branches[i].profit
        if temp > left_profit:
            left_profit = temp

    temp = 0
    right_profit = -math.inf
    #calculate right
    for i from m+1 to h+1 do
        temp = temp + branches[i].profit
        if temp > right_profit:
            right_profit = temp

    return max(left_profit + right_profit, left_profit, right_profit)


def smartMostProfitableCluster(arr, l, h):
    if l == h:
        return branches[l].profit

    m = (l + h) / 2

    return max(smartMostProfitableCluster(branches, l, m),
               smartMostProfitableCluster(branches, m + 1, h),
               findClusterProfit(branches, l, m, h))
```