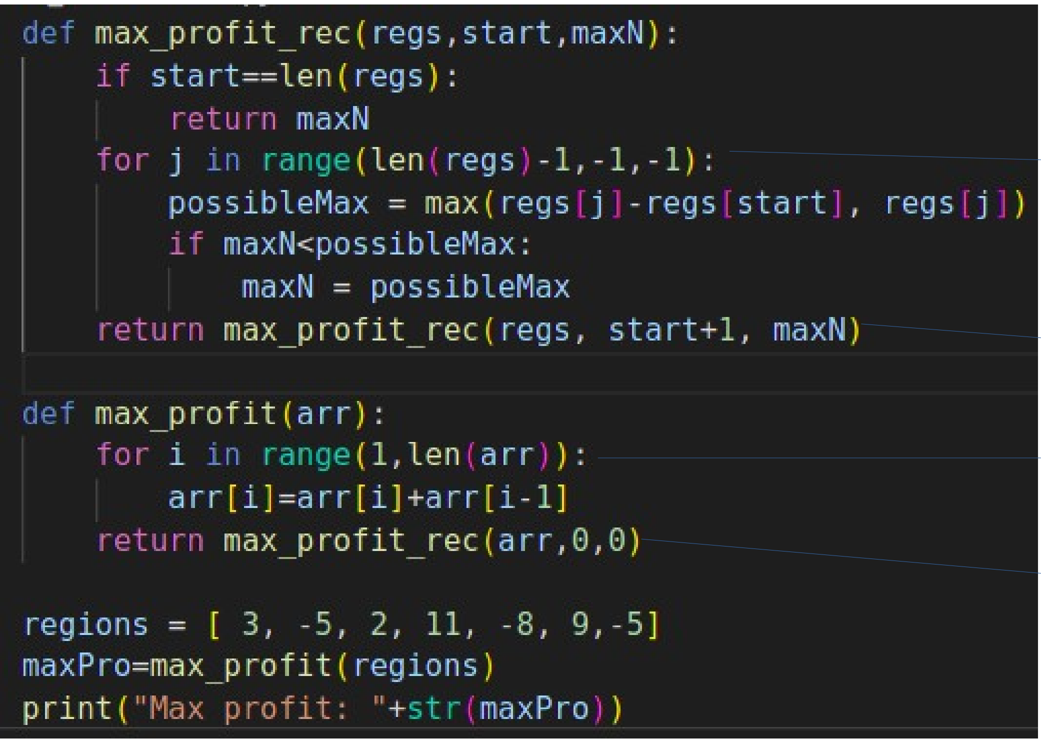
1)Cluster

a)



```
def max_profit_rec(regs,start,maxN):
    if start==len(regs):
        return maxN
    for j in range(len(regs)-1,-1,-1):
        possibleMax = max(regs[j]-regs[start], regs[j])
        if maxN<possibleMax:
            maxN = possibleMax
    return max_profit_rec(regs, start+1, maxN)

def max_profit(arr):
    for i in range(1,len(arr)):
        arr[i]=arr[i]+arr[i-1]
    return max_profit_rec(arr,0,0)

regions = [ 3, -5, 2, 11, -8, 9,-5]
maxPro=max_profit(regions)
print("Max profit: "+str(maxPro))
```

$\sum_{0}^{n-1} 1=\Theta(n)$

T(n-1)

$\sum_{0}^{n-1} 1=\Theta(n)$

T(n)

Complexity:

$T(n)=T(n-1)+\Theta(n)$
$T(n-1)=T(n-2)+\Theta(n-1)$
:
$T(1)=\Theta(1)$

n times ==> $n*\Theta(n)=\Theta(n^2)$

Obtaining:

arr[]=[A,B,C,D,E,F,G]

Firstly,I wrote to that location by starting from 1 and adding up with the previous number.

Ex: arr[1]=arr[1]+arr[0]; arr[2]=arr[2]+arr[1]

End of the summation:

arr[0]=A arr[2]=A+B+C arr[4]=A+B+C+D+E
arr[1]=A+B arr[3]=A+B+C+D arr[5]=A+B+C+D+E+F
arr[6]=A+B+C+D+E+F+G

Then substract arr[start-index] from all element one by one and check if it is greater then current max;

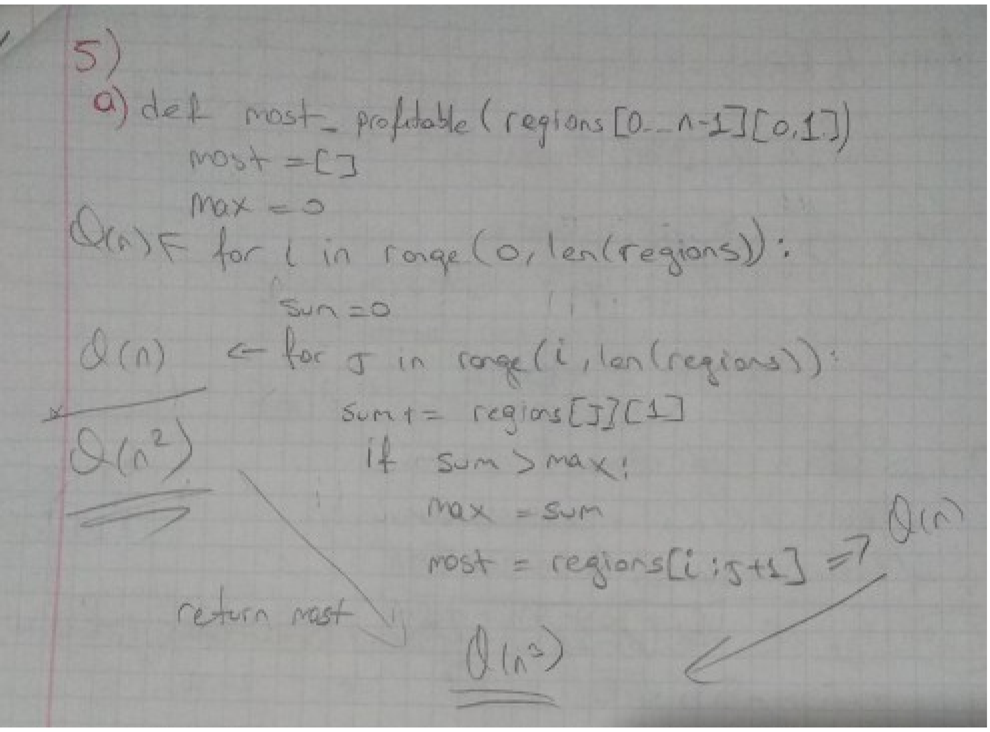
Ex: arr[5]-arr[2]= A+B+C+D+E+F - (A+B+C)= D+E+F
arr[4]-arr[1]= A+B+C+D+E - (A+B)= C+D+E

if substaction result is greater then current max, new max= substraction result

In the end:

if start index is equals to size of array , return max.

b)



5)

a)
```
def most_profitable(regions[0..n-1][0,1])
    most =[]
    max = 0
    for i in range(0,len(regions)):
        sum=0
        for j in range(i,len(regions)):
            sum += regions[j][1]
            if sum > max:
                max = sum
                most = regions[i:j+1]
return most
```

$\Theta(n) \leftarrow$

$\Theta(n) \leftarrow$

$\Theta(n^2)$

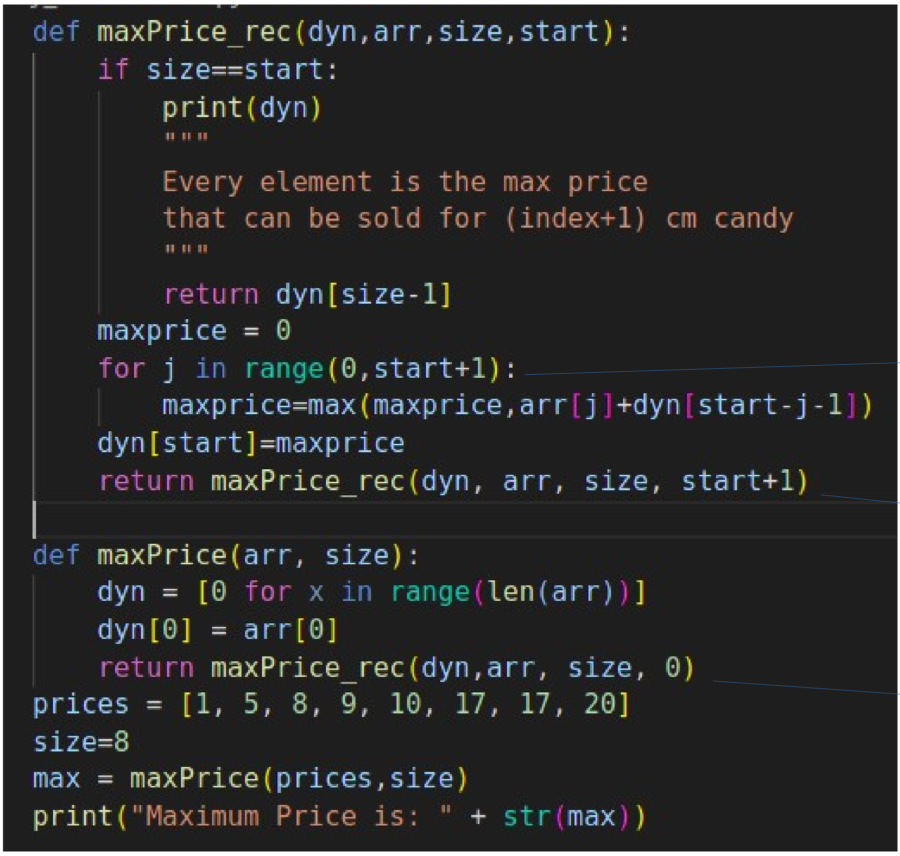
$\Rightarrow \Theta(n)$

$\Theta(n^3)$

In my previous homework, my algorithm finds in $\Theta(n^3)$ due to copying subarray in a loop .

So,current algorithm is better.

2)



```
def maxPrice_rec(dyn,arr,size,start):
    if size==start:
        print(dyn)
        """
        Every element is the max price
        that can be sold for (index+1) cm candy
        """
        return dyn[size-1]
    maxprice = 0
    for j in range(0,start+1):
        maxprice=max(maxprice,arr[j]+dyn[start-j-1])
    dyn[start]=maxprice
    return maxPrice_rec(dyn, arr, size, start+1)

def maxPrice(arr, size):
    dyn = [0 for x in range(len(arr))]
    dyn[0] = arr[0]
    return maxPrice_rec(dyn,arr, size, 0)
prices = [1, 5, 8, 9, 10, 17, 17, 20]
size=8
max = maxPrice(prices,size)
print("Maximum Price is: " + str(max))
```

$\rightarrow \sum_{0}^{start} 1=\Theta(n)$

$\rightarrow$ T(n-1)

$\rightarrow$ T(n)

Complexity:

$T(n)=T(n-1)+\Theta(n)$
$T(n-1)=T(n-2)+\Theta(n-1)$
:
$T(1)=\Theta(1)$

n times ==> $n*\Theta(n)=\Theta(n^2)$

Obtaining:

Firstly,starting form index=1, I compared given price with cut prices

Ex:

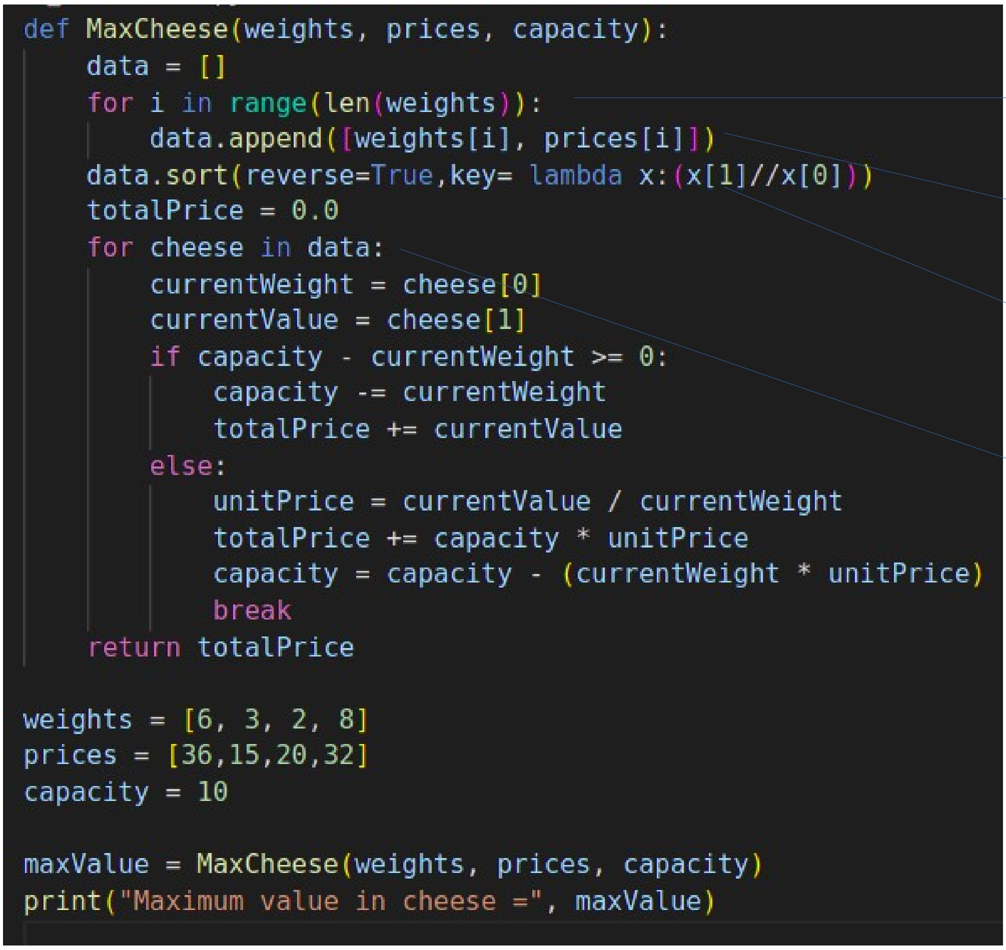
if the sum of the x-cm candy and y-cm candy prices is greater than (x+y) cm candy ,I changed the price of (x+y) cm candy with new price that can be sold

Ex:Sum of prices of 1 cm + 6 cm is greater than the price of 7 cm so new price of 7 cm is 18

In the end:

We have higher prices that can be sold x-cm candy

3)



```
def MaxCheese(weights, prices, capacity):
    data = []
    for i in range(len(weights)):
        data.append([weights[i], prices[i]])
    data.sort(reverse=True,key= lambda x:(x[1]//x[0]))
    totalPrice = 0.0
    for cheese in data:
        currentWeight = cheese[0]
        currentValue = cheese[1]
        if capacity - currentWeight >= 0:
            capacity -= currentWeight
            totalPrice += currentValue
        else:
            unitPrice = currentValue / currentWeight
            totalPrice += capacity * unitPrice
            capacity = capacity - (currentWeight * unitPrice)
            break
    return totalPrice

weights = [6, 3, 2, 8]
prices = [36,15,20,32]
capacity = 10

maxValue = MaxCheese(weights, prices, capacity)
print("Maximum value in cheese =", maxValue)
```

$$\sum_{0}^{n-1} 1=\Theta(n)$$

Amortized Ɵ(1)

Ɵ(n*log(n))

$$\sum_{0}^{n-1} 1=\Theta(n)$$

Complexity:

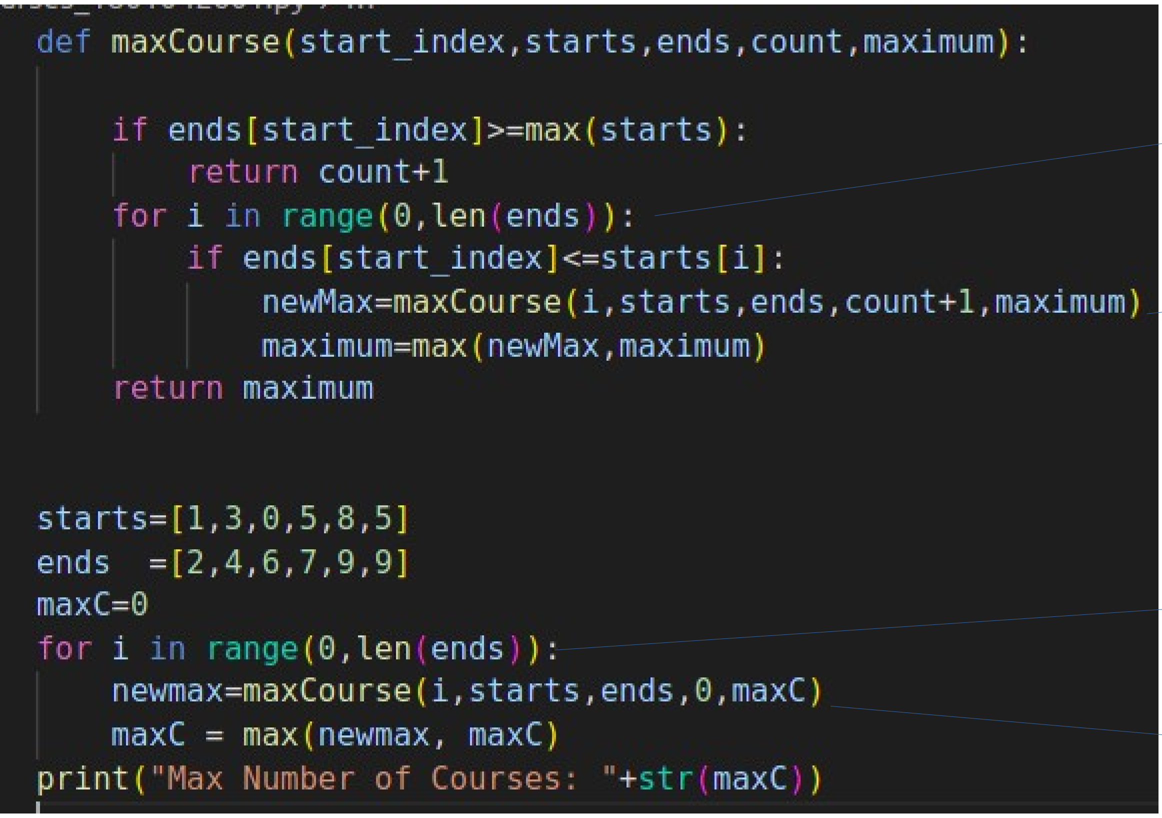
T(n)=Ɵ(n*log(n))

Obtaining:

Firstly,I sorted the arrays by unit prices (price /weight) in decreasing order. In the end,lowest indexs have more expensive cheese.

Then starting from zeroth index,I checked the capacity if there is enough space for whole cheese.If there is, I added whole cheese and increase total price by price of that cheese. If not ,I added as much cheese as the empty space and increase total price correspondingly, then returned total price because there is not any empty space

4)



```python
def maxCourse(start_index,starts,ends,count,maximum):

    if ends[start_index]>=max(starts):
        return count+1
    for i in range(0,len(ends)):
        if ends[start_index]<=starts[i]:
            newMax=maxCourse(i,starts,ends,count+1,maximum)
            maximum=max(newMax,maximum)
    return maximum


starts=[1,3,0,5,8,5]
ends  =[2,4,6,7,9,9]
maxC=0
for i in range(0,len(ends)):
    newmax=maxCourse(i,starts,ends,0,maxC)
    maxC = max(newmax, maxC)
print("Max Number of Courses: "+str(maxC))
```

$\sum_{0}^{n-1} T(n-1)$

$T(n-1)$

$\sum_{0}^{n-1} T(n)$

$T(n)$

Complexity:

$$\left.\begin{array}{l} T(n)= n*T(n-1)+1 \\ T(n-1)= n*T(n-2)+1 \\ \quad\vdots \qquad\qquad \vdots \\ T(1)= 1 \end{array}\right\} \quad \text{n times}==>O(n^n)$$

Total Complexity: $O(n^n)$

Obtaining:

Starting from zeroth course , I checked if there is any course that starts after current course’s end time,if there is I chose it and do the process again,if not return current number of taken courses. If the returned number is greater than current max number of courses,new max is returned value . This algorithm works for all arrays(sorted / unsorted).

In the begining, if we sort the courses by the time of lesson(end- begin time) and sort the courses that has same time difference by starting time in increasing order, new complexity would be definitely better than this algorithm.

Since the purpose of this question is to design a greedy algorithm,so this algorithm should be sufficient.