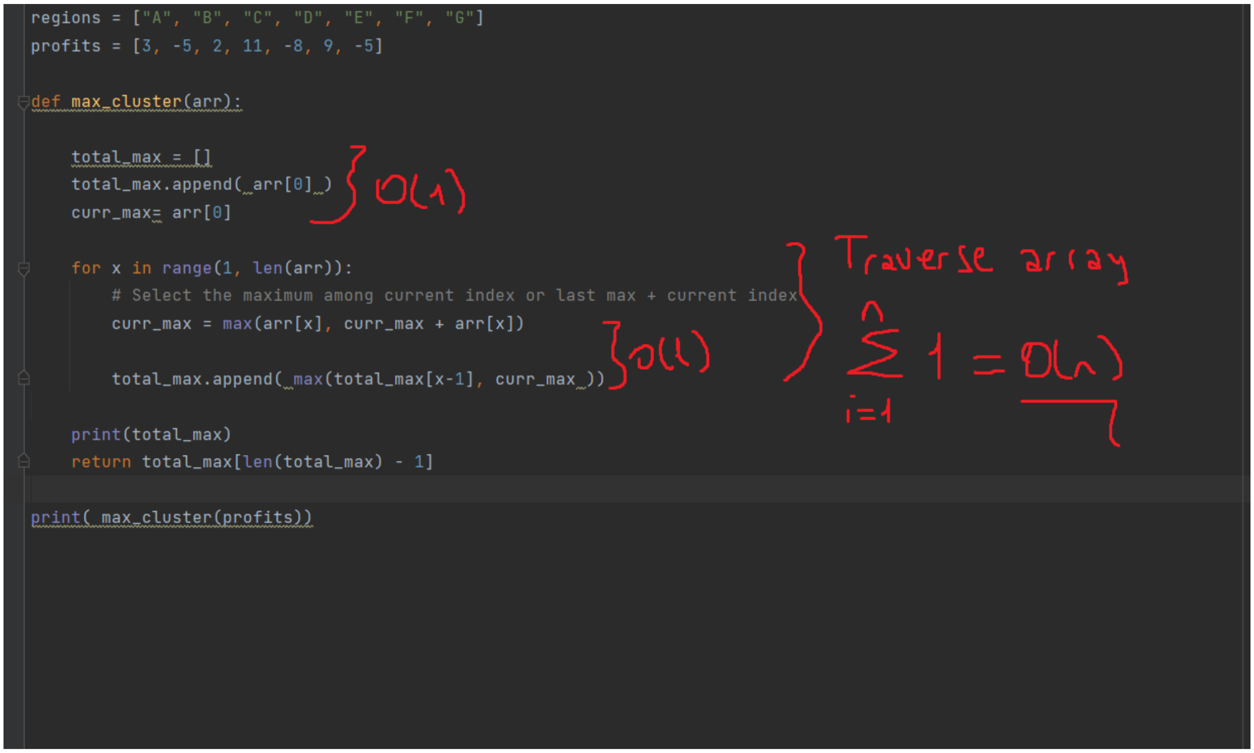
1)



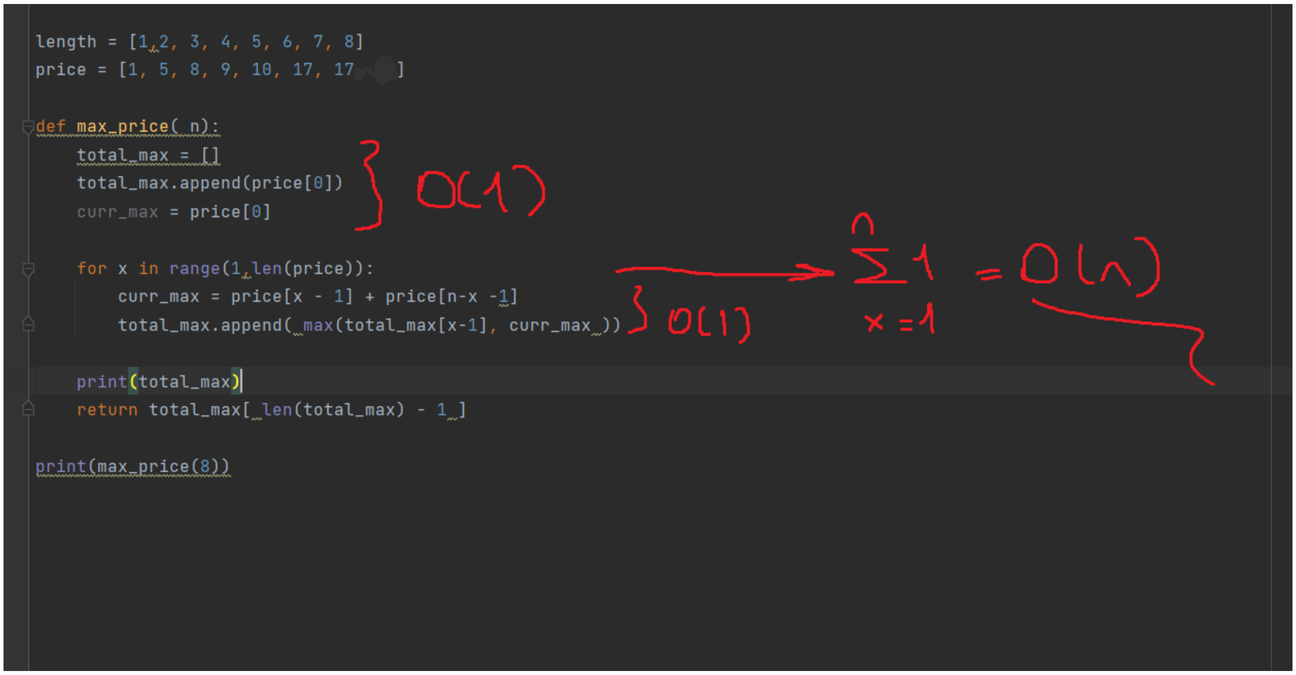
```python
regions = ["A", "B", "C", "D", "E", "F", "G"]
profits = [3, -5, 2, 11, -8, 9, -5]

def max_cluster(arr):

    total_max = []
    total_max.append( arr[0] )
    curr_max= arr[0]

    for x in range(1, len(arr)):
        # Select the maximum among current index or last max + current index
        curr_max = max(arr[x], curr_max + arr[x])

        total_max.append( max(total_max[x-1], curr_max ))

    print(total_max)
    return total_max[len(total_max) - 1]

print( max_cluster(profits))
```

Curr_max's value depends on "Is the next value on array greater than the last elements' max?" and picks the greater.

Total_max's value depends on "Is the current max's value greater than the last max? If yes , pick the greater" . Total_max's last index has the value of the maximum possible value.

Because it is just a traversing array algorithm , time complexity is O(n)

2)



```python
length = [1,2, 3, 4, 5, 6, 7, 8]
price = [1, 5, 8, 9, 10, 17, 17[redacted]]

def max_price( n):
    total_max = []
    total_max.append(price[0])
    curr_max = price[0]

    for x in range(1,len(price)):
        curr_max = price[x - 1] + price[n-x -1]
        total_max.append( max(total_max[x-1], curr_max ))

    print(total_max)
    return total_max[ len(total_max) - 1 ]

print(max_price(8))
```

$\}\ O(1)$

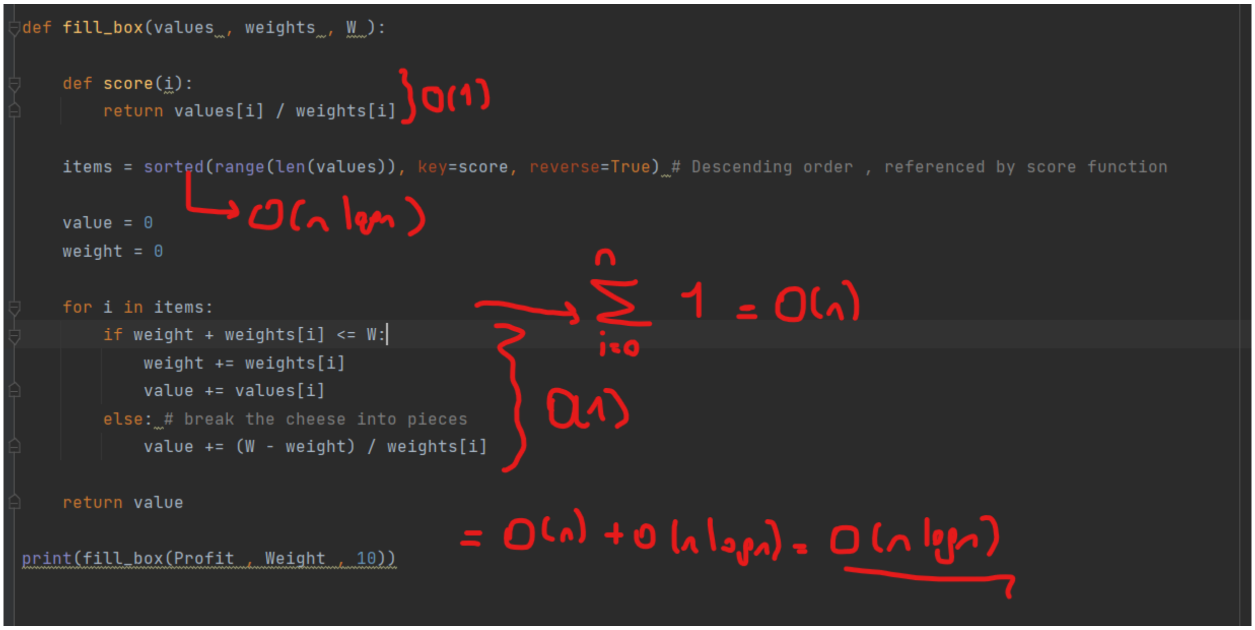
$\}\ O(1)$

$$\sum_{x=1}^{n} 1 = O(n)$$

Curr_max's value is depends on the remainder and the split portion's price of the candy stick.

Total_max decides if the next index will has the last index's value or the curr_max's value. That way last index of total_max will has the maximum value possible.

3)



```
def fill_box(values , weights , W ):

    def score(i):
        return values[i] / weights[i]

    items = sorted(range(len(values)), key=score, reverse=True) # Descending order , referenced by score function

    value = 0
    weight = 0

    for i in items:
        if weight + weights[i] <= W:
            weight += weights[i]
            value += values[i]
        else: # break the cheese into pieces
            value += (W - weight) / weights[i]

    return value

print(fill_box(Profit , Weight , 10))
```

$\}O(1)$

$\rightarrow O(n \log n)$

$\rightarrow \sum_{i=0}^{n} 1 = O(n)$

$\}O(1)$

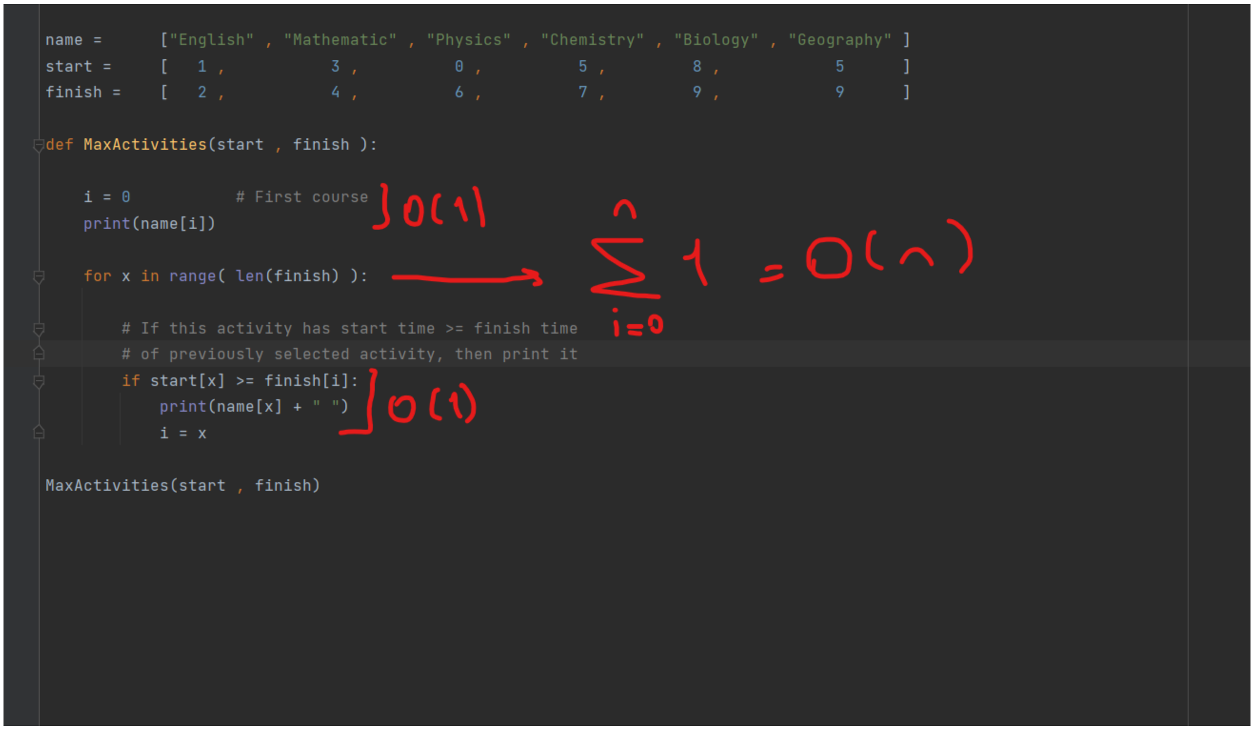
$= O(n) + O(n \log n) = \underline{O(n \log n)}$

Explanation:

First I defined a nested function inside the main function which calculates profit/weight ratio. Then, I sorted the values descending order with reference to nested function score() to reach most reasonable ones by incrementing the index. Then I started to fill the box starting with the most reasonable ones which are cheeses that have the biggest profit/weight ratio. I used every space in the box by breaking the next corresponding cheese into pieces.

Sorted function is from python standard library and its time complexity is O(n logn).

4)



```
name =      ["English" , "Mathematic" , "Physics" , "Chemistry" , "Biology" , "Geography" ]
start =     [   1 ,           3 ,          0 ,          5 ,          8 ,             5      ]
finish =    [   2 ,           4 ,          6 ,          7 ,          9 ,             9      ]

def MaxActivities(start , finish ):

    i = 0           # First course
    print(name[i])

    for x in range( len(finish) ):

        # If this activity has start time >= finish time
        # of previously selected activity, then print it
        if start[x] >= finish[i]:
            print(name[x] + " ")
            i = x

MaxActivities(start , finish)
```

Selecting the maximum possible number of courses means don't wasting any time between courses. So comparing the next course's start time with last course's finish time will result us to decide how not to waste anytime.