1. Profit rates of many companies have changed due to the pandemic. 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	B	C	D	E	F	G
3	-5	2	11	-8	9	-5

(a) Design a dynamic programming algorithm that **finds the maximum profit** belonging to the most profitable cluster (the cluster must contain a consecutive region). For example, the maximum profit is 14 (C-D-E-F) on the table above.

```
def maxProfitableCluster(arr):
    maximum = -999999
    temp_max = 0
    for i in arr:
        temp_max = max(i, i+temp_max)
        if(temp_max>maximum):
            maximum = temp_max
    return maximum


print(maxProfitableCluster([3,-5,2,11,-8,9,-5]))
```

const

In dynamic programming, best cluster found so far is always saved and added on top of the current element. So one iteration is enough to process all possible clusters.

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

(b) This question was asked in one of your homework before. **Compare** the algorithm with the algorithms you previously proposed in terms of complexity. Explain your arguments.

$T(n) = 2T(n/2) + \Theta(n)$

$f(n) = \Theta(n) = \Theta(n^{\log_2 2})$

Case 2 of Master T.

$T(n) = \Theta(n \log n)$

Analyze from previous homework (dividee and conquer)

In divide and conquer case, some of the arithmetic operations are done over and over again. But in dynamic programming case, old values are saved and used through the execution. So by using a bit extra memory (constant amount), it gives a asymptotically better performance.

There is a candy shop, and candies are produced as a stick of length n cm. They can cut and sell candies in different lengths, and there is a price list that shows prices of all pieces of size smaller than n. For example, if the length of the candy is 8 cm and the values of different pieces are given as in the following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	17	17	20

Design a dynamic programming algorithm that finds the maximum obtainable value.

```python
def cutCandy(prices,length):
    max_prices = [0 for x in range(len(prices)+1)] #empty list with size length

    for i in range(1, len(prices)+1):
        temp_val = -99999999999
        for j in range(i):
            temp_val = max(temp_val, prices[j] + max_prices[i-j-1]) #if there is a better option update temp_val
        max_prices[i] = temp_val

    return max_prices[length] #last index of max_prices is the highest profit
```

$$\sum_{i=1}^{n}\sum_{j=0}^{i} 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

3. There is a store where dairy products like milk and cheese are sold. There exist n different types of cheese in the store, and each cheese type has a price p_i and a weight w_i. The owner wants to put a combination of different cheeses in a box, and sell it. The box has a weight capacity W, and you are allowed to cut cheeses and put any portion of it in the box to maximize the total price without exceeding the box weight capacity. Design a greedy algorithm that finds the maximum price you can get.

I remember this problem is called the knapsack problem from the class but in the version we saw on class did not enable to cut the items. So I searched a bit and found out this is called the knapsack problem with a fraction.

```python
class Cheese:  # weight, price pairs
    def __init__(self, weight, price):
        self.weight = weight
        self.price = price
        self.cost = price // weight

    def __lt__(self, other):  # for sorting by cost
        return self.cost < other.cost


def getMaxPrice(cheeses, capacity):
    cheeses.sort(reverse=True)  # by cost (price/weight)
    totalprice = 0
    for i in cheeses:
        cur_weight = int(i.weight)
        cur_price = int(i.price)
        if capacity - cur_weight >= 0:
            capacity -= cur_weight
            totalprice += cur_price
        else:
            fraction = capacity / cur_weight
            totalprice += cur_price * fraction
            break
    return totalprice
```

const

$$T(n) = \sum_{i=1}^{n} 1 = \Theta(n)$$

If the current cheese can fit into the box, put it. Else find how much of it can fit and put it.

4. A group of gifted students will be prepared for an intelligence contest. In order to prepare them, a program has been designed where each student can take courses among n courses, and the start and end times of the courses are given. A table is given below as an example.

Course	Start	Finish
English	1	2
Mathematics	3	4
Physics	0	6
Chemistry	5	7
Biology	8	9
Geography	5	9

can't take chemistry and geography at the same time. Should be biology

A student can attend at most **4** of the above courses . The maximum set of courses that can be attended is {English, Mathematics, Chemistry, ~~Geography~~}. Design a greedy algorithm to find the maximum number of courses a student can attend among n courses.

```python
class Course:  # weight, price pairs
    def __init__(self, name, start, end):
        self.name = name
        self.start = start
        self.end = end

    def __str__(self): for printing
        return self.name

def findMaxCourses(courses):
    maxCourses = []
    i = 0
    maxCourses.append(courses[i]) #first lesson is always taken
    for j in range(len(courses)):
        if courses[j].start >= courses[i].end: #if the previous lesson starts after current lesson
            maxCourses.append(courses[j])  #add it to list
            i = j #and update the previous lesson
    return maxCourses
```

O(1)

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