*Homework 5 Yeşim Yalçın 20010202094

① * My previous divide and conquer algorithm for this problem was:
→ Divide the array into two parts
→ Find max profit of the left half
→ Find max profit of the right half
→ Find if the intersection can have a better profit.
* Here the main problem is to find if the intersection can have a better profit. This can be achieved in linear time like this:
→ Find the maximum sum starting from mid point and ending at some point on left of mid then find the maximum sum starting from mid+1 and ending with some point on right of mid+1.
↳ This gives us the recurrence: $T(n) = 2T(\frac{n}{2}) + O(n)$
↳ I found divide and conquer method's complexity as $O(n \log n)$ in HW3.

* Dynamic Programming Approach:
* In the dynamic programming approach, we need to start from the smallest piece. Starting from the first element of the array to the last an element:
→ Can be included into the existing max cluster.
→ Can be not included into the existing max cluster but start its own potential cluster.
* The element will be included if:
→ currentMax + Element > Element
* If it is the reverse, the element itself is better than the current max cluster so it can create a new better cluster potentially.
* We will have a currentMax variable to hold the value of the cluster we are analysing at the moment.
* We will have a totalMax variable to hold the max profit we got so far.

* The total Max will be updated if currentMax becomes greater than it.

* Requrrence for currentMax:
↳ CM is current max ↳ en is current element
→ $CM(n) = \max\{CM(n-1) + en, en\}$
* Requrrence for total Max:
↳ TM is total Max
→ $TM(n) = \max\{TM(n-1), CM(n)\}$

* Here Space Complexity = $O(1)$ because we hold only 2 variables with size 1.
* Time Complexity = $O(n)$ because there will be a loop that runs for all n elements and in each run will only do 2 comparisons and 1 addition.

$$\sum_{i=1}^{i=n-1} 1 = \frac{(n-1)-1+1}{2} = \frac{n-1}{2} \in O(n)$$

* Dynamic Programming method is better than divide and conquer method in terms of time complexity. Worse in terms of Space Complexity. However, the space complexity is pretty negligible.

(2) Stick candies with length n cm.

* We can divide all feasible solutions to 2 categories
→ Subsets that do not include the i'th item.
↳ Optimum solution is $F(i-1, j)$
→ Subsets that do include the i'th item
↳ Optimum solution is $p_i + F(i-1, j-l_i)$

* $F(i,j)$ ⇒ The optimum solution for i different possible lengths that when combined creates a candy with length j.
→ p_i ⇒ price of the element with length i
→ l_i ⇒ length of the element with length i

$$* F(i,j) = \begin{cases} \max(F(i-1,j), p_i + F(i-1, j-l_i)) & \text{if } j \geq l_i \\ F(i-1,j) & \text{if } j < l_i \end{cases}$$

* Assuming we have an array including different candy size options with prices. We need to determine if including that candy length will give us an optimum solution. We can do this by applying the recurrence relation formula. If we store all calculated values in a 2D array and run the formula at the top from 1'st index to n-1'th index, we can achieve a dynamic programming algorithm.

* Space Complexity: We will have a 2D array containing all $F(i,j)$'s. i indicates the candy number, j indicates the dividing candy size.

* Time Complexity: We will have a loop running for each possible candy amount → n. This loop will also have a loop inside to calculate the solution for all c initial candy sizes. Meaning we will have $O(n*c)$ time complexity. The basic operation is $O(1)$.

$$\sum_{i=1}^{i=n} \sum_{j=1}^{j=c} 1 \Rightarrow \sum_{i=1}^{i=n} \frac{c-1+1}{2} \Rightarrow \frac{c}{2} \sum_{i=1}^{i=n} 1 \Rightarrow \frac{c}{2} * \frac{n}{2} \in O(n*c)$$

(3) n different types of cheese. Has price p_i, weight w_i.
There are boxes. Capacity W
Need to maximize the price of box.

* Option for Optimal Solution:
→ Calculate the ratio of $\frac{price}{weight}$ for each cheese type.
→ Sort the cheese types according to the calculated ratios.
→ Place the cheese types as whole pieces in decreasing order to the box.
→ When it is not possible to place a whole piece to the box anymore, divide the cheese and put it to the box.
→ Box is completed.

* Sorting can be done in $O(n \log n)$ time. I will use a mergeSort.
* After sorting we need a loop that will iterate over the items in the sorted list. This loop will run at most n times for n different cheese types. Inside the loop there will be assignments and possibly division, substraction. Meaning basic operation of the loop is $O(1)$. So the loop has $O(n)$ time complexity.

$$\sum_{i=0}^{i=n-1} 1 = \frac{(n-1)-0+1}{2} = \frac{n}{2} \in O(n)$$

* That's why time complexity depends on the mergesort.
$T(n) = O(n \log n)$

(4) Each student can take courses among n courses.
Courses have start and finish times.

* Option for Optimal Solution:

→ Sort the courses according to their finish time.
→ Choose the course with the earliest finish time.
→ Eliminate the courses that conflicts the selected course.
→ Continue till there are no selections left.

* Sorting can be done in $O(n \log n)$ time. I will use a mergesort for this.
* For eliminating, we can simply remove the courses that has a start time in between start time and end time of selected course (start time inclusive, end time exclusive) But doing this will cause extra work. We can just simply skip them and not select them to avoid extra time complexity.
* In a loop from 0 to n-1, select the course if it does not have a conflict between the previously selected course. If not, skip it.
This loop will run for all courses and inside there will be assignments and comparisons only. So, the basic operation of it is $O(1)$.

$$\sum_{i=0}^{i=n-1} 1 \quad \frac{n-1+1}{2} = \frac{n}{2} \in O(n)$$

* The time complexity depends on the sorting which is $O(n \log n)$