Burak Yildirim
19010426 09

Q1) In this algorithm, I initiate total_max and curr_max with the first element of the array. Then I traverse all elements of the array starting from index 1. In the loop, I assign the bigger value between current element and summation of current element and curr_max to curr_max. After that I assign the bigger value between total_max and curr_max to total_max. After the loop ended I return total_max.

Time complexity:

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

Time complexity of my previous algorithm for finding max profit was $\Theta(n.\log n)$. So I improved my algorithm from $\Theta(n.\log n)$ to $\Theta(n)$.

Q2) Vals array stores the maximum value obtained for each candy with length <= total length. In outer loop I first assign -99999 as a min int to max_val. Then in the inner loop which iterates i, current length of candy, times I compare max_val to the summation of price[j] and vals[i-j-1], and assign the bigger one to max_val. The reason I take i-j-1 as index is to make sure that index of vals array and j equals to i which is what we want to stop algorithm from going outside of the boundries of the candies. After inner loop, I store the obtained max value to vals[i] to use later on. Finally after the execution of outer loop ends, I return vals[n] which holds the last obtained max value.

Time complexity:

$$\sum_{i=1}^{n} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n} i = \frac{n.n+1}{2} \Rightarrow T(n) \in \Theta(n^2)$$

Q3) I first wrote a Cheese class that hold the weight, the price and the price per weight of a cheese. Then in algorithm I creat an array which contains cheeses with given prices and weights. After that I sort the array reversed according to cheeses' price per weight. In for loop, I take cheeses with full weight until a cheese's weight exceeds capacity and start to cut the cheeses. While doing so I increment total by the price of the cheese's used weight. After for loop I return total.

Time complexity:

$$\sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1} 1 = n+n = 2n$$

But python sort's time complexity is $n.\log n$, so overall time complexity of the algorithm is $T(n) \in \Theta(n.\log n)$

Q4) I assumed that courses are sorted according to finish times because the example in pdf was sorted. In algorithm I always select the first course first, then in the loop I check if the current course's start time is greater than or equal to the last course's finish time. If it is, then I assign the index of the current course to last course and increment the counter by 1. After loop is finished I return counter.

Time complexity:

$$\sum_{i=0}^{n-1} 1 = n \Rightarrow T(n) \in \Theta(n)$$