Berkcan EKİCİ
171044015

# Homework 5 Answers:

1)

a) We can easily solve this problem in linear time using Kadane's Algorithm. The idea is to maintain a maximum subarray "ending" at each index of the given array. This subarray is either empty or consists of one more element than the maximum subarray ending of the previous index.

Number of for loop iterations needed in the worst case is:

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

$$T(n) = n = O(n)$$

b) The complexity of my algorithm in assigment 3 was $O(n \log n)$

Since our current solution is $O(n)$, if we take the limit:

$$\lim_{n \to \infty} \frac{n}{n \log n} = 0$$

Our current solution is more efficient.

Berkcan EKICI
171044015

2) Here, I compute the solutions for smaller candies first, knowing that they will later be used to compute the solutions for larger candies. The answer will once again be stored in arr[n]. My aproach is "bottom up approach".

• In the bottom up approach, we start by filling the array from start. So, we will first initialize an array "arr" and then we will iterate over it to fill it.

```
arr = [None] * (n+1)
arr[0] = 0
```

Now, the array arr contains the maximum value that can be generated by each length.

Analysis: The analysis of the "bottom up approach" is simple. We are using nested loops, the first loop is iterating 1 to n and the second loop is iterating from 1 to x. (x jonging from 1 to n)

$$T(n) = n + (n-1) + (n-2) + \dots + 2 + 1$$

$$= \frac{n.(n+1)}{2} = O(n^2)$$

Berkcan EKİCİ
171044015

3) We need to sort the values according to their price/weight ratio.

Start adding the item with the maximum price/weight ratio.

Add the whole cheese, if the current weight is less than the capacity, else, add a portion of the cheese to the box.

Stop, when all the cheeses have been considered and the total weight becomes equal to the weight of the given box.

## My Code Analysis:

- The Python list sort() function has been using the "Timsort Algorithm" since version 2.3. This algorithm has a runtime complexity of $O(n \log n)$

- my for loop:

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

So, time complexity of the sorting + time complexity of the loop $\Big\} O(n \log n) + O(n) = O(n \log n)$

Berkcan EKİCİ
171044015

4) My algorithm:

→ Sort the given courses in ascending order.

→ Select the first course from sorted array coursesArray[] and add it to maximumCoursesArray[]

→ If the start time of the currently selected course is greater than or equal to the finish time of previously selected course, then add it to the maximumCoursesArray[].

→ Print the maximumCoursesArray[] and return count.

Analyse: There are 2 cases.

Case 1: When a given set of courses are already sorted according to their finish time, then there is no sorting mechanism invoked, in such a case the complexity of the algorithm will be O(n)

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

Case 2: When a given set of courses is unsorted, then we will have to use the sort() function. The time complexity of this method will be O(nlogn).