CSE321 - Introduction to Algorithms - HW5

Fatih Kaan Salgır - 171044009

1

a)

At any position, the maximum profit is either the element at the position or it is combined with previous cluster with maximum profit. By applying this to the algorithm, iteratively, we can calculate the local maximum of the cluster, and then combine them to obtain maximum profit.

$C(n) = \sum_{i=0}^{n} 1 \in \Theta(n)$, where n is the size of the array

Application of the dynaminc programming doesn't affect the space complexity.

b)

In brute-force approach the complexity is $\Theta(n^2)$, in divide & conquer approach the complexity is $\Theta(n \log(n))$. The dynamic programming approach can be considered as improved version of the brute-force version, since it eliminates the checking of all possible sub-clusters but still using the previously calculated values. And we have $\Theta(n)$ complexity, which is the most efficient one among three algorithms.

2

The algorithm calculates the maximum profit for each length starting from 0. For the subsequent values, the prior value is memoized. This decreases the runtime of the inner loop drastically. Since it is iterated over the first n element of the array, where n is the initial length, the complexity algorithm doesn't depend on the size of the price array but only the length of the candy. If we wouldn't implement dynamic programming;

$C(n) = \sum_{i=0}^{n} \sum_{j=0}^{i} 1 \in \Theta(n^2)$, where n is the initial length of the candy

With memoization the time complexity decreases to linear time (it is demonstrated in Figure 2). Space complexity is also only dependant on given length which is $\Theta(n)$, since it is memoized the maximum price corresponding to given length.

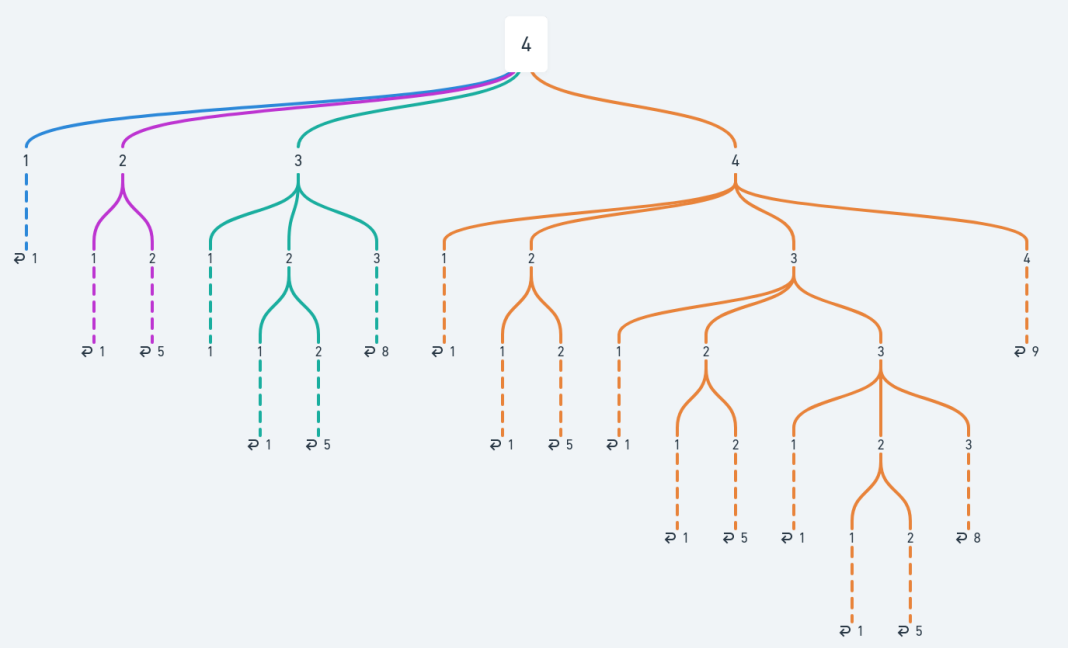

Figure 1: Example: length = 4, without using memoization

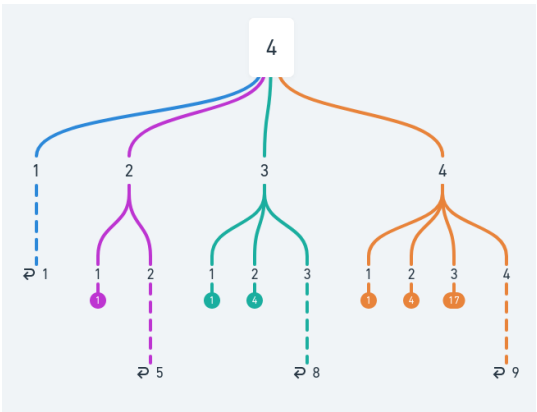

Figure 2: Example: length = 4, dynamic programming with memoization

3

First the cost of the cheese, ratio of price to weight, is calculated. At each step of the algorithm, the cheese with maximum cost is chosen, which is locally optimal choice at each stage. In order to choose the cheese with maximum cost, the array is sorted by cost. Both calculating the costs, and choosing among them takes linear time. Sorting ideally takes $\Theta(n \log(n))$ time. In total we have $\Theta(n \log(n))$ time complexity.

$C(n) \in \Theta(n \log(n) + n + n) = \Theta(n \log(n))$

However, this sorting can lead unnecessary complexity if we have low capacity but higher number of cheese types. Since, it sorts all array, but we need only first n number of maximum cost values. So, one optimization might be getting the maximum value at each step.

4

The optimal choice of the greedy algorithm is finding the earliest finished course, and removing the overlapped courses with the earliest finished course. Finding the earliest is basically finding the minimum element in an array which takes linear time. Removing the overlapped courses also takes linear time. At worst case where we don't have overlapping course, the number of courses is getting decreased by 1 at each step. That leads to $\Theta(n^2)$ complexity. However, in an average case, the removal of overlapping courses propagates the next step, which decreases the running time drastically.

$C_{worst}(n) = \sum_{i=0}^{n} \sum_{j=0}^{i} 1 \in \Theta(n^2),$ where n number of courses