CSE 321 - Homework #5

Harun ALBAYRAK - 171044014

Q1) I have used dynamic programming to solve this problem. (q1.py)
So we will create a 2D array of size (arr.size() +1 * (target +1) of type boolean.

The state Array [i][J] will be true if there exists a subset of elements from A [0..... i] with sum value zero.

My approach : if (A[i] > 0)
Array [i][0] = Array [i - 1][0]
else
Array [i][0] = Array [i-1] [0 - A[i]]

1. This means that if current element has value greater than current sum value we will copy the answer for previous cases

2. And if the current sum value (0) is the greater than ith elementh we will see if any of previous states have already experienced the sum = 0 or any previous states experienced a value 0 - A[i] which will solve our purpose.

set = [2 , 3, -5, -8 , 6 , -1]
sum = 0

	0
0	True
2	True
3	True
-5	True
8	True
6	True
-1	True

* Every set has a subset that is 0.
Hence, this problem must be true for every Array.

CamScanner ile tarandı

Q2) I solved this problem using Memoization, dynamic programming. (q2.py)

→ Starting from the top node, traverse recursively with each node, till the pathsum of that node is calculated.

→ And then store the result in 'memo' array. But this will take $O(n^2)$ space to maintain the array.

Q3) I have used dynamic programming to solve this knapsack problem (q3.py)

→ First of all, all the probable weights from 1 to W serve as the columns and weights are kept as the rows.

→ The state Array[i][J] denotes the maximum value of "J-weight" considering all values from 1 to ith. So if we consider "w_i" it is put in all columns which have "weight values > w_i". Two possibilities occur - to fill or not to fill "w_i" in the given column.

→ If we don't fill "ith" weight in "jth" column then Array[i][J] state will be same as Array[i-1][J]. But if we fill the weight, Array[i][j] will be equal to the value of "w_i" + value of the column weighing "J-w_i" in the previous row. Hence, we take the maximum of these two possibilities to fill the current state.