171044032

Türker Tercan

HW 5

Q1:

```
pseudocode sum_with_zero (index, summation, arr, n, subset_list)
    if index == n
        if summation == 0
            print (subset_list)
            return 1
        end if
        else
            return 0
        end if
    end if
    if visit[index][summation + array_size]
        if dp[index][summation + array_size] != 0
            print (subset_list)
        end if
        return dp[index][summation + array_size]
    end if
    visit[index][summation + array_size] = 1

    subset_list.append (arr[index])
    temp1 = sum_with_0 (index + 1, summation + arr[index], arr, n, subset_list)
    subset_list.remove (arr[index])
    temp2 = sum_with_0 (index + 1, summation, arr, n, subset_list)
    dp[index][summation + array_size] = temp1 + temp2

    return dp[index][summation]
end
```

without dynamic approach this would be $O(2^n)$ to determine all the subsets
we have table with size [length (input)][total_absolute_value(input)]
we have table with size because we may have negative inputs and we don't want to index it with negative numbers
so algorithm does simillar to brute-force approach instead of calalate every possible subset, store the calculated subsets for further use and when find a subset with sum zero print it

Time Complexity: $O(n \cdot S)$ S is the sum of all elements

Q3: Knapsack Problem With Repeated Elements :

Capacity + 1

Values					- - -	Result

} knapsack array

Knapsack [capacity +1] can be used such that knapsack[i] stores maximum value can be stored using all items and i is the capacity of the knapsack

For example: weight = [5,4,2] value = [10,4,3] capacity = 9

0	1	2	3	4	5	6	7	8	9

our array will be sized as Capacity +1

↑

1- Check if you can put a item at current index weight[5,4,2] so can't put any item to 2 index

0	0	3							

↑

2- Check you can put a item here and its value needs to be greater than knapsack [i - weight[index]]. So 3 can be put here

0	1	2	3	4					
0	0	3	3						

↑

3- We can put here second and the third item lets check them

2nd weight: 4 value: 4 index - weight = 0 knapsack [0] = 0

3rd weight: 2 value: 3 index - weight = 2 knapsack [2] = 3

Knapsack [0] + value = 4

knapsack[2] + value = 6 → chose this and continue like that

0	0	3	3	6	10	10	13	13	16

→ result

And i used different array like knapsack and store all the subsets to current capacity. You can review it in the python code