Print Subsets Sum to K

Problem Level: Hard

Problem Description:

Given an array A of size n and an integer K, print all subsets of A which sum to K.

Note : The order of subsets is not important.

Sample Input 1:

```
9
5 12 3 17 1 18 15 3 17
6
```

Sample Output 1:

```
3 3
5 1
```

Approach to be followed:

To find the subsets that sum to K, we will utilise a recursive approach. Just as always, we will try to break our problem into a bunch of subproblems. Let us select one element from our array, then we break our problem in two ways. They are – recursive call with that element included in the sum, and recursive call with that element not included in the sum.

For example, consider the array [3, 2, 5, 1, 4, 6] with K = 8. In this, let us select 3. Now, the recursive calls made will be

1. On array [2, 5, 1, 4, 6], with K = 8 - 3 = 5, as 3 has been included in the sum already.
2. On array [2, 5, 1, 4, 6] with K = 8, as in this, we do not want to include 3 in sum.

Using a **startIndex,** initially 0, we can easily utilise the array part other than 3.

The answer that we get from recursion in the first step will be smallOutput, which contains all the subsets which sum to 5. To make it the final output, we append 3 in the returned subsets, hence making the total sum equal to 8.

Since we want to print our answer and not return it, we shall maintain a 1D array, which stores the output so far for a particular case. When the base case is achieved for this particular case, we shall simply print the array. Base case occurs when the length of the input array becomes 0 (startIndex becomes equal to size of array), and K also becomes 0.

The recursion tree for the first two levels is show below:

Steps:

1. Initialise a 1D array to store the output so far and send it as a parameter in the helper function.
2. Inside the helper function, declare a 1D array (**newOutput**) to maintain the output so far for one of the two recursive calls to be made.
3. Using a loop, copy the elements of the previous output into **newOutput**. Also, append the number at **startIndex** to the **newOutput**, as this will be storing the output where the fixed number is considered to be a part of the output.
4. Call recursion first with the previous output and **K** as it is, and increasing **startIndex** by one.
5. Second, call recursion with the **newOutput**, **K - arr[startIndex]** (as we have considered this in our output itself) and increase startIndex by one.
6. Base Case: Repeat the above steps until startIndex becomes equal to n. If K is also equal to 0, print the output so for, else return from the function.

Pseudo Code:

```
void helper(input[], startIndex, size, output[], oSize, k)

     if startIndex equals size:

         if k equals 0:

             loop from i = 0 till i = oSize:

                 print output[i]


         return

     declare smallOutout1[1000], smallOutput2[1000]

     smallSize1 = 0

     smallSize2 = 0

     helper(input, startIndex + 1, size, output, oSize, k)
```

```
    if k greater than 0

        i = 0

        loop from i = 0 till i < oSize:

            smallOutput2[i] = output[i]

        smallOutput2[i] = input[startIndex]

        helper(input, startIndex+1, size, smallOutput2, oSize+1,
k-input[startIndex])


void printSubsetSumToK(input[], size, k)

    declare output[1000]

    helper (input, 0, size, output, 0, k)
```

Time Complexity: $O(2^N)$, where **N** is the size of the input array. Since, for every index **i,** two recursive cases originate, hence the time complexity is **$O(2^N)$**.