Return Subsets Sum to K

Problem Level: Hard

Problem Description:

Given an array A of size n and an integer K, return 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.

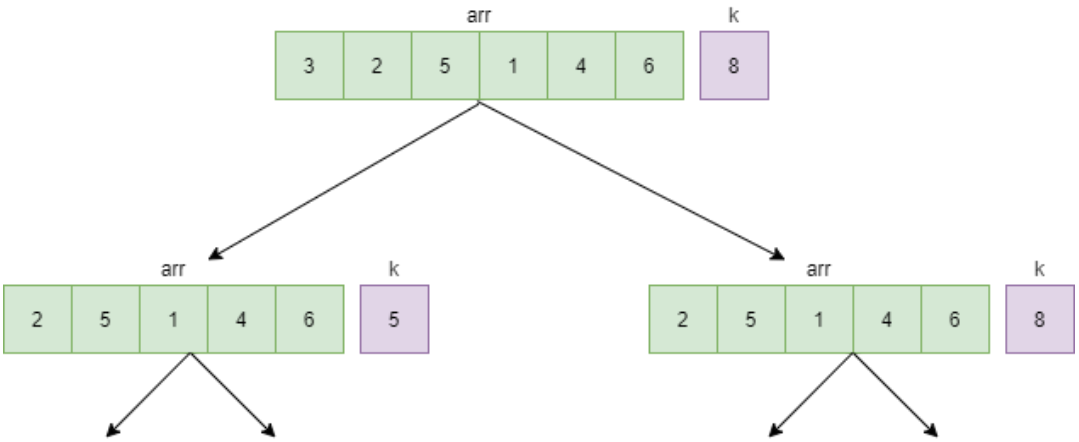
For example, consider the array [3, 2, 5, 1, 4, 6] with K = 8. In this, let us select 3. Now, the recursive calls made will be

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.

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



In Java, Recursion calls will return a 2D array containing the subsets, and using them, we will create a new 2D array, output, which will store the return subsets from both recursions. Make sure you remember to append 3 in the subsets returned from the first recursive call. If we reach a point, where the **startIndex** becomes equal to the size, and value of **K** becomes 0, this means we have found a valid subset, so in this case we return a 2D array with 1 row and 0 columns. Otherwise, if K is not 0, we return a 2D array with 0 rows and 0 columns.

In C++, since we cannot use jagged arrays, we need to make some changes. Before calling recursion, we will create two 2D arrays, and pass it inside recursion. These will hold the subsets for us. Using recursion, we will return the size of these subsets, so we may create a final 2D output array, where the first column will hold the size of subsets, and from the second column onwards we have the subsets themselves. If we reach a point, where the **startIndex** becomes equal to the size, and value of **K** becomes 0, this means we have found a valid subset, so in this case we return 1 as the size of the subset, otherwise we return 0.

Steps:

1. Initialise two 2D arrays to store the output for the above mentioned two recursive calls, say, **smallOutput1** and **smallOutput2.**
2. Using two recursion calls, find out the subsets which include **K**, where the output array passed is **smallOutput2,** and do not include **K,** where the output array passed is **smallOutput1 .**
3. Let the sizes returned by these recursions be stored in **size1** and **size2**.
4. Copy the output stored in **smallOutput1** in a final output array.
5. Also, copy the output stored in **smallOutput2,** with **K** appended in the end, in the final output array.
6. Keep a count of the number of rows while copying. This will give you the size of your output matrix.

7. Finally, return this size.
8. Base Case: Repeat the above steps until startIndex becomes equal to n. If K is also equal to 0, return 1, else return 0.

Pseudo Code:

```
function subsetSumToK(input[], n, startIndex, output[][50], int k)

     if startIndex equals n

         if k equals 0

             output[0][0] = 0

               return 1

     else

         return 0


     declare smallOutput1[1000][50] and  smallOutput2[1000][50]

     next = startIndex+1

     size1 = subsetSumToK(input, n, startIndex + 1, smallOutput1, k)

     size2 = subsetSumToK(input, n, startIndex + 1, smallOutput2, k -
input[startIndex])



     row = 0

     loop from i = 0 to i < size1:

         loop from j = 0 to j <= smallOutput1[i][0]:

             output[row][j] = smallOutput1[i][j]

     row = row + 1


     loop from i = 0 to i < size1:

         output[row][0] = smallOutput2[i][0] + 1

         output[row][1] = input[startIndex]

         loop from j = 0 to j <= smallOutput2[i][0]:

             output[row][j+1] = smallOutput2[i][j]
```

```
            row++
        return row


function subsetSumToK(input[], n, output[][50], k)
        return subsetSumToK(input, n, 0, output, 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)**.