Return Subset of an Array

Problem Level: Hard

Problem Description:

Given an integer array (of length n), find and return all the subsets of the input array.

Note : The order of subsets is not important.

Sample Input 1:

```
3
15 20 12
```

Sample Output 1:

```
[] (this just represents an empty array, don't worry about the square brackets)
12
20
20 12
15
15 12
15 20
15 20 12
```

Approach to be followed:

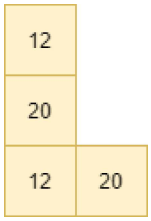
All the subsets of an array can be found using recursion.

Suppose the input array is **"15 20 12".** Let us fix **"15"**, and using recursion, we shall find all the subsets of **"20 12".** The subsets of **"20 12"** are **- "[] , [12], [20], [20 12].** Hence, the subsets can be merged with the fixed number to get the desired output.

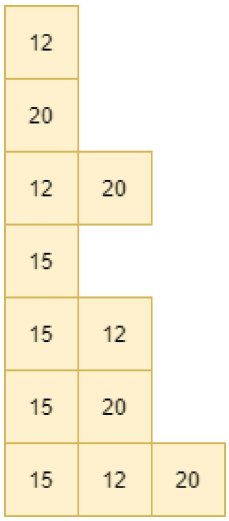
Thus, the subsets found by recursion and the subsets after merging "15" are**- [], [12] ,[20] [20,12], [15] , [15,12], [15,20], [15,20,12] .**

We shall maintain a 2D array structure to store all these outputs and return them.

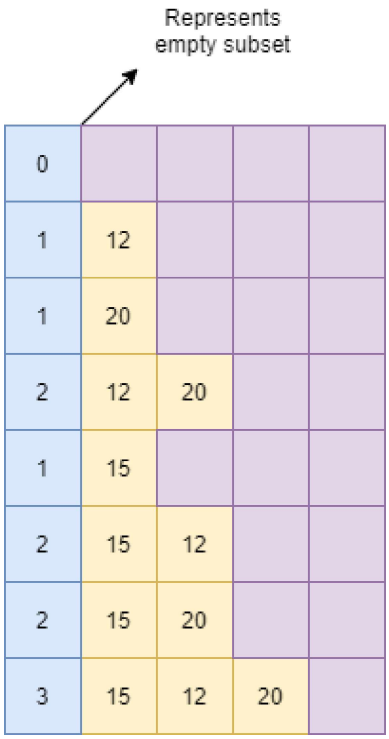
In Java, we may use jagged lists to store this information. So, when we are fixing 15, we call recursion to find subsets of 20 and 12. The jagged array in this case, or the smallerOutput in this case looks like:



Next, including these returned 4, we have 4 more subsets that can be generated by merging these with 15. Thus, jagged array now would look like:



In C++ however, since we do not have jagged lists, we use a column specifically to store the size of subsets at that level. This would look like:



Steps:

1. Initialise a variable **startIndex**, and pass it in the recursive helper function, which will require **input**, **output**, and the **startIndex** (initially 0).
2. Recursively find the subsets which do not contain the element at **startIndex**.
3. Recursion returns the size for the subsets, say, **smallSize.**
4. Using a loop, we append the subsets returned to our 2D output array, along with their size as the first column.
5. Now, we append the element at **startIndex** to the 2D output array as well.
6. Return 2*smallSize, as the total number of subsets generated will be twice the number obtained from the recursion at that level. (for example, if we receive 3 subsets from recursion, we need to create 3 more subsets after appending the element at **startIndex**)

Pseudo Code:

```
function subset(input[], n, startIndex, output[][20])
     if startIndex equals n
         output[0][0] = 0
         return 1

     smallSize = subset(input, n, startIndex + 1, output)

     loop from i = 0 till i < smallSize:
         output[i+smallSize][0] = output[i][0] + 1
         output[i+smallSize][1] = input[startIndex]
         loop from j = 1 till j < output[i][0]:
             output[i+smallSize][j+1] = output[i][j]

     return 2 * smallSize

function subset(input[], n, output[][20]):
     return subset(input, n, 0, output)  //initially startIndex = 0
```

Time Complexity: O(2^N), where **N** is the size of the input array.