[C - Subset sum](https://vjudge.net/problem/UVA-12911" \t "_blank)

Given a set S of integers, your task is to determine how many different non-empty subsets sum up to a target value T.

**INPUT**

The first line of each test case is a line containing two integers N and T, the number of items of the original set of integers and the target value. After that comes one line with the N integers si that belong to the original set s.

* 1 ≤ N ≤ 40
* −109 ≤ T, si ≤ 109

**OUTPUT**

For each test case print on a single line an integer indicating the number of different non-empty subsets that sum up to the target value T.

**SAMPLE INPUT**

6 0

-1 2 -3 4 -5 6

5 0

-7 -3 -2 5 8

**SAMPLE OUTPUT**

4

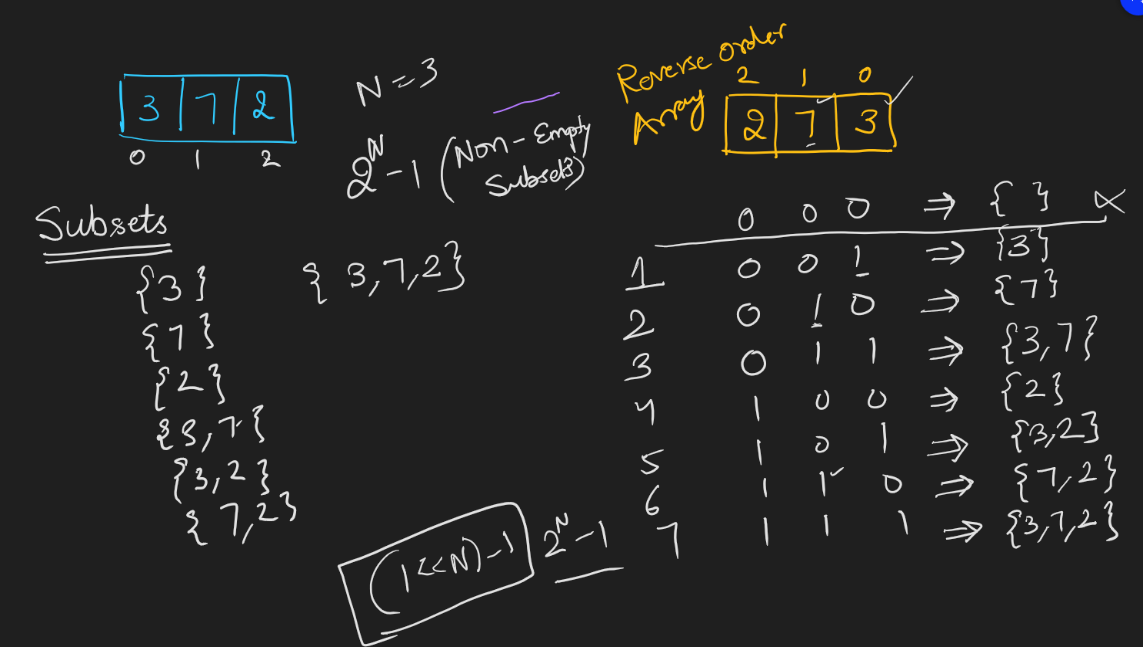
1

**EXPLANATION**

On the first test case the target is 0 and the following are the valid subsets: (2, 4, -1, -5), (2, 6, -5, -3), (4, -1, -3), (6, -5, -1).

Brute Force (with Bit Manipulation)

Maintain count = 0, Generate all subsets of given array, find their sum, if sum matches target value increase count.



Skeleton Code

long long countSubsets(list v, int targetSum){  
    long long count = 0;  
    long long totalSubsets = (1 << v.size());  
    for(int i = 1; i < totalSubsets; i++){  
        **long long curSum = 0;**  
        for(int bitPos = 0; bitPos < v.size(); bitPos++){  
            if(checkBit(i, bitPos))   
                curSubset.add(v.get(bitPos));  
        }  
        **if(curSum == targetSum)  
            count++;**  
     }  
     **return count;**  
}

Time Complexity:  T \* N \* 2N, Space Complexity:  1,  Verdict:  TLE

Because for N = 40, 40 \* 240 = 40 \* (210)4 = 40\*1012

IDEA

240 gives TLE, what about 220?

220 => 106, which is acceptable.  So what if we do this?

Divide array of size 40 into 2 separate arrays of size 20 each.

Generate all subsets sum of first half and second half.  firstHalfSums[] and secondHalfSums[]

Now we need to just find how many pairs of firstHalfSums[i] +  secondHalfSums[j] = targetSum, this can be solved efficiently using MAP. as we need to find number of subsets in first half whose sum is equal to targetSum - secondHalfSums[j].

So steps are essentially

* Generate all subsets sum of [0, n/2-1] elements and store them in a map with KEY as subset sum and value as FREQUENCY OF SUBSET SUM.
* count = 0
* Generate subsets sum of second half of array [n/2, n-1]
  + for each subsetSumOfSecondHalf
    - count += map[targetSum - subsetSumOfSecondHalf]
* return count;

Time Complexity:

Let X = N/2 and Y = 2N/2

Subsets Generation of First Half + Inserting Each Sum in Map + Subset Generation of Second Half + Searching Each Sum in Map

X\*Y + Y logY + X\*Y + Y logY

2X\*Y + 2Y\*logY = 2 \* N/2 \* 2N/2 + 2 \* 2N/2 log2N/2= N \* 2N/2 + N\*2N/2= 2\*N\*2N/2

For N = 40,  2\*N\*2N/2  = 80 \* 220  = 80 \* 106

This approach is called \*MEET IN THE MIDDLE\* where we divide array into two parts to convert 2N complexity problem solution to two subproblems solutions of 2N/2 complexities and then combine their results to solve actual problem.