Problem Statement

Given a set of positive numbers (non-zero) and a target sum 'S'. Each number should be assigned either a '+' or '-' sign. We need to find out total ways to assign symbols to make the sum of numbers equal to target 'S'.

Example 1:

```
Input: {1, 1, 2, 3}, S=1
Output: 3
Explanation: The given set has '3' ways to make a sum of '1': {+1-1-2+3} &
{-1+1-2+3} & {+1+1+2-3}
```

Example 2:

```
Input: {1, 2, 7, 1}, S=9
Output: 2
Explanation: The given set has '2' ways to make a sum of '9': {+1+2+7-1} &
{-1+2+7+1}
```

Solution

The problem follows the **0/1 Knapsack pattern** and can be converted into "Count of Subset Sum". Let's dig into this

We are asked to find two subsets of the given numbers whose difference is equal to the given target 'S'. Take the first example above. As we saw, {+1-1-2+3}. So, the two subsets we are asked to find are {1,3} & {1,2} because: `{1 + 3} - {1 + 2} = 1`

Now, let's say `Sum(s1)` denotes the total sum of set `s1`, and `Sum(s2)` denotes the total sum of set `s2`. So, the required equation is: `Sum(s1) - Sum(s2) = S`.

The equation can be reduced to the subset sum problem Let's assumed that `Sum(num)` denotes the total sum of all numbers, therefore: `Sum(s1) + Sum(s2) = Sum(num)`

Let's add the above two equations:

```
Sum(s1) - Sum(s2) + Sum(s1) + Sum(s2) = S + Sum(num)`
2 * Sum(s1) = S + Sum(num)
Sum(s1) = (S + Sum(num)) / 2
```

This essentially converts our problem to: "Find count of subsets of the given numbers whose sum is equal to" `(S + Sum(num)) / 2`

Code: We have taken the DP code of Count of Subset Sum and extended it to solve this:

```
const findTargetSubsets = (num, sum) => {
    let totalSum = 0;
    for (let i = 0; i < num.length; i++) totalSum += num[i];

    // if 's + totalSum' is odd, we can't find a subset with sum equal to
'(s + totalSum) / 2`
    if (totalSum < sum || (sum + totalSum) % 2 === 1) return 0;

    return countSubsets(num, (sum + totalSum) / 2);
};

let countSubsets = function (num, sum) {
    const n = num.length;
    const dp = Array(n)
        .fill(0)
        .map(() => Array(sum + 1).fill(0));

    // populate the sum=0 columns, as we will always have an empty set for
zero sum
    for (let i = 0; i < n; i++) {
        dp[i][0] = 1;
    }

    // with only one number, we can form a subset only when the required
sum is equal to its value
    for (let s = 1; s <= sum; s++) {
        dp[0][s] = num[0] == s ? 1 : 0;
    }

    // process all subsets for all sums
    for (let i = 1; i < num.length; i++) {
        for (let s = 1; s <= sum; s++) {
            // exclude the number
            dp[i][s] = dp[i - 1][s];
            // include the number, if it does not exceed the sum
            if (s >= num[i]) {
                dp[i][s] += dp[i - 1][s - num[i]];
            }
        }
    }

    // the bottom-right corner will have our answer.
    return dp[num.length - 1][sum];
};
console.log(`Count of Target sum is: ---> ${findTargetSubsets([1, 1, 2,
3], 1)}`);
console.log(`Count of Target sum is: ---> ${findTargetSubsets([1, 2, 7,
1], 9)}`);
```

The above solution has time and space complexity of `O(N*S)`, where 'N' represents total numbers and 'S' is the desired sum.

We can further improve the solution to use only O(S) space.

Space Optimized Solution

```
const findTargetSubsets = function (num, s) {
    let totalSum = 0;
    for (let i = 0; i < num.length; i++) totalSum += num[i];

    // if 's + totalSum' is odd, we can't find a subset with sum equal to
'(s + totalSum) / 2'
    if (totalSum < s || (s + totalSum) % 2 == 1) return 0;

    return countSubsets(num, (s + totalSum) / 2);
};

let countSubsets = (num, sum) => {
    const n = num.length;
    const dp = Array(sum + 1).fill(0);
    dp[0] = 1;

    for (let s = 1; s <= sum; s++) {
        dp[s] = num[0] === s ? 1 : 0;
    }

    for (let i = 1; i < n; i++) {
        for (let s = sum; s >= 0; s--) {
            if (s >= num[i]) dp[s] += dp[s - num[i]];
        }
    }

    return dp[sum];
};

console.log(`Count of Target sum is: ---> ${findTargetSubsets([1, 1, 2,
3], 1)}`);
console.log(`Count of Target sum is: ---> ${findTargetSubsets([1, 2, 7,
1], 9)}`);
```