**Count of Subset Sum**

### Problem Statement [**#**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG#problem-statement)

Given a set of positive numbers, find the total number of subsets whose sum is equal to a given number ‘S’.

##### Example 1: [**#**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG#example-1)

Input: {1, 1, 2, 3}, S=4  
Output: 3  
The given set has '3' subsets whose sum is '4': {1, 1, 2}, {1, 3}, {1, 3}  
Note that we have two similar sets {1, 3}, because we have two '1' in our input.

##### Example 2: [**#**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG#example-2)

Input: {1, 2, 7, 1, 5}, S=9  
Output: 3  
The given set has '3' subsets whose sum is '9': {2, 7}, {1, 7, 1}, {1, 2, 1, 5}

private int countSubsetsRecursive(Integer[][] dp, int[] num, int sum, int currentIndex) {

    // base checks

    if (sum == 0)

      return 1;

    if(num.length == 0 || currentIndex >= num.length)

      return 0;

    // check if we have not already processed a similar problem

    if(dp[currentIndex][sum] == null) {

      // recursive call after choosing the number at the currentIndex

      // if the number at currentIndex exceeds the sum, we shouldn't process this

      int sum1 = 0;

      if( num[currentIndex] <= sum )

        sum1 = countSubsetsRecursive(dp, num, sum - num[currentIndex], currentIndex + 1);

      // recursive call after excluding the number at the currentIndex

      int sum2 = countSubsetsRecursive(dp, num, sum, currentIndex + 1);

      dp[currentIndex][sum] = sum1 + sum2;

    }

    return dp[currentIndex][sum];

  }

### Bottom-up Dynamic Programming [**#**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG#bottom-up-dynamic-programming)

We will try to find if we can make all possible sums with every subset to populate the array db[TotalNumbers][S+1].

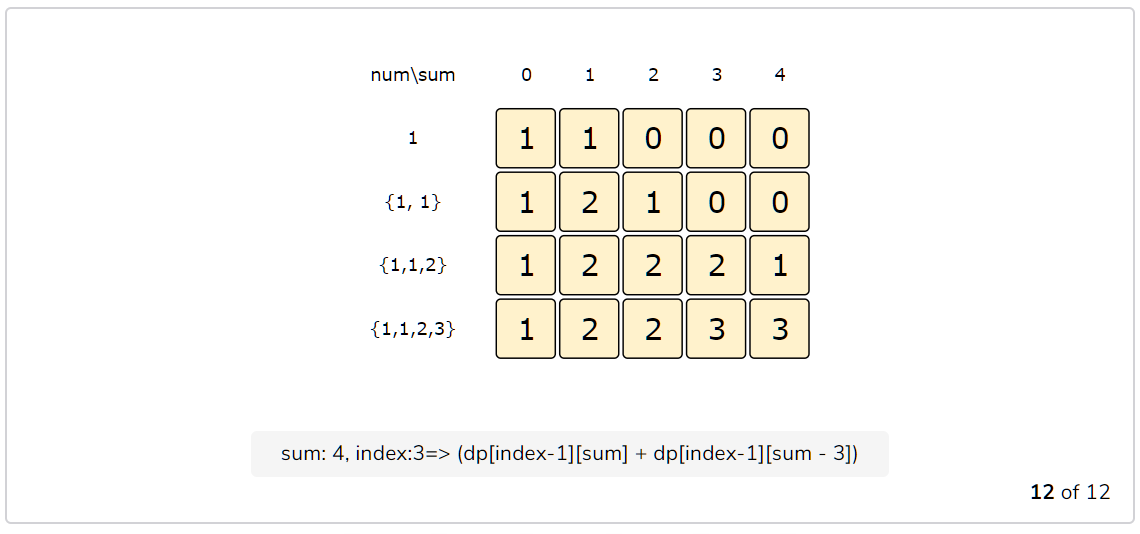
So, at every step we have two options:

1. Exclude the number. Count all the subsets without the given number up to the given sum => dp[index-1][sum]
2. Include the number if its value is not more than the ‘sum’. In this case, we will count all the subsets to get the remaining sum => dp[index-1][sum-num[index]]

To find the total sets, we will add both of the above two values:

    dp[index][sum] = dp[index-1][sum] + dp[index-1][sum-num[index]])

Let’s start with our base case of size zero:



class SubsetSum {

  public int countSubsets(int[] num, int sum) {

    int n = num.length;

    int[][] dp = new int[n][sum + 1];

    // populate the sum=0 columns, as we will always have an empty set for zero sum

    for(int 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(int s=1; s <= sum ; s++) {

      dp[0][s] = (num[0] == s ? 1 : 0);

    }

    // process all subsets for all sums

    for(int i=1; i < num.length; i++) {

      for(int 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];

  }

  public static void main(String[] args) {

    SubsetSum ss = new SubsetSum();

    int[] num = {1, 1, 2, 3};

    System.out.println(ss.countSubsets(num, 4));

    num = new int[]{1, 2, 7, 1, 5};

    System.out.println(ss.countSubsets(num, 9));

  }

}

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

### Challenge [**#**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m27pwPZ3yvG#challenge)

Can we further improve our bottom-up DP solution? Can you find an algorithm that has O(S)*O*(*S*) space complexity?

class SubsetSum {

  static int countSubsets(int[] num, int sum) {

    int n = num.length;

    int[] dp = new int[sum + 1];

    dp[0] = 1;

    // with only one number, we can form a subset only when the required sum is equal to its value

    for(int s=1; s <= sum ; s++) {

      dp[s] = (num[0] == s ? 1 : 0);

    }

    // process all subsets for all sums

    for(int i=1; i < num.length; i++) {

      for(int s=sum; s >= 0; s--) {

        if(s >= num[i])

          dp[s] += dp[s-num[i]];

      }

    }

    return dp[sum];

  }

}