**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/3j64vRY6JnR#problem-statement)

Given a set of positive numbers, determine if there exists a subset whose sum is equal to a given number ‘S’.

**Example 1:**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/3j64vRY6JnR#example-1)

Input: {1, 2, 3, 7}, S=6  
Output: True  
The given set has a subset whose sum is '6': {1, 2, 3}

**Example 2:**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/3j64vRY6JnR#example-2)

Input: {1, 2, 7, 1, 5}, S=10  
Output: True  
The given set has a subset whose sum is '10': {1, 2, 7}

**Example 3:**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/3j64vRY6JnR#example-3)

Input: {1, 3, 4, 8}, S=6  
Output: False  
The given set does not have any subset whose sum is equal to '6'.

//Bottom up approach same as: com.interview.leetcode.dp.grokkingdp.boundedknapsacktype.PartitonEqualSubsetSum

static boolean canPartition(int[] num, int sum) {

    int n = num.length;

    boolean[] dp = new boolean[sum + 1];

    // handle sum=0, as we can always have '0' sum with an empty set

    dp[0] = true;

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

    for (int s = 1; s <= sum; s++) {

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

    }

    // process all subsets for all sums

    for (int i = 1; i < n; i++) {

      for (int s = sum; s >= 0; s--) {

        // if dp[s]==true, this means we can get the sum 's' without num[i], hence we can move on to

        // the next number else we can include num[i] and see if we can find a subset to get the

        // remaining sum

        if (!dp[s] && s >= num[i]) {

          dp[s] = dp[s - num[i]];

        }

      }

    }

    return dp[sum];

  }