**Problem link**: [Count Subsets with sum K](https://bit.ly/3B5JBkU)

**Approach**:

* DP-14: f(ind, target) signifies that from (0 to ind) can this target be achieved (true/ false). Now, this time we can count subsets whose sum add up to target.
* To count something in recursion we return 1 & 0 in base cases, will definitely use notTake- take technique so we can add (notTake + take) to return total count.

**1) recursion:** tc = O(2^n), sc = O(N) for recursion stack space

**int f(int ind, vector<int> &nums, int target){**

**if(target==0) return 1;**

**if(ind==0) return nums[ind] ==target;**

**int notTake = f(ind-1, target);**

**int take = 0;**

**if(nums[ind]<= target)**

**take= f(ind-1, target- nums[ind]);**

**return (notTake + take);**

**}**

**int findWays(vector<int> &nums, int target){**

**int n = nums.size();**

**return f(n-1, nums, target);**

**}**

2) **Memoization**: dp[n][target+1]

tc= O(n\* target)

sc = O(n\* target) + O(N)

**int f(int ind, vector<int> &nums, int target, vector<vector<int>> &dp){**

**if(target==0) return 1;**

**if(ind==0) return nums[ind] ==target;**

**if(dp[ind][target] != -1) return dp[ind][target];**

**int notTake = f(ind-1, target, dp);**

**int take = 0;**

**if(nums[ind]<= target)**

**take= f(ind-1, target- nums[ind], dp);**

**return dp[ind][target] = (notTake + take);**

**}**

**int findWays(vector<int> &nums, int target){**

**int n = nums.size();**

**vector<vector<int>> dp(n, vector<int>(target+1, -1));**

**return f(n-1, nums, target, dp);**

**}**

3) **Tabulation**: build a dp[n][target+1] table from bottom to up.

tc = O(n\* target)

sc = O(n\*target)

**int findWays(vector<int> &nums, int target){**

**int n = nums.size();**

**vector<vector<int>> dp(n, vector<int>(target+1, 0));**

**//use base cases to fill the dp[][] on a initial level**

**for(int i=0; i<n; i++){**

**dp[i][0] = 1; //target = 0**

**}**

**if(nums[0]<= target) dp[0][nums[0]] = 1;**

**//use nested for loops to build dp[][] table**

**for(int ind=1; ind<n; ind++){**

**for(int k= 1; k<= target; k++){**

**int notTake = dp[ind-1][k];**

**int take = 0;**

**if(nums[ind]<= k)**

**take= dp[ind-1][k- nums[ind]];**

**dp[ind][k] = (notTake + take);**

**}**

**}**

**return dp[n-1][target];**

**}**

4) **Space optimization**: use prev[target+1] to store [i-1]th row values & curr[target+1] to store current row values

tc = O(n\*target)

sc = O(2\*target)

**int findWays(vector<int> &nums, int target){**

**int n = nums.size();**

**vector<int> prev(target+1, 0), curr(target+1, 0);**

**//use base cases on a initial level**

**prev[0]= curr[0] = 1; //target = 0**

**//index = 0**

**if(nums[0]<= target) prev[nums[0]] = 1;**

**//use nested for loops to build dp[][] table**

**for(int ind=1; ind<n; ind++){**

**for(int k= 1; k<= target; k++){**

**int notTake = prev][k];**

**int take = 0;**

**if(nums[ind]<= k)**

**take= prev[k- nums[ind]];**

**curr[k] = (notTake + take);**

**}**

**prev = curr;**

**}**

**return prev[target];**

**}**