**Problem link:** [Partition a set into two subsets such that the difference of subset sums is minimum.](https://www.codingninjas.com/codestudio/problems/partition-a-set-into-two-subsets-such-that-the-difference-of-subset-sums-is-minimum_842494?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTab=0)

**Approach**: In the DP14 we have seen that - last row of dp[][] contains 0 to k (target/ totalSum in this case) true/ false values which can be achieved by sum up of a subset made from the given array.

In this problem we need to find the minimum difference of subset sum of s1 & s2 such that abs(s1- s2) will be minimum. If we can get which sum values s1 can take (0 to totalSum) then we can find s2 easily by s2 = totalSum- s1. And we can use a variable mini to keep track of abs(s1- s2) min value.

so all possible subset 1 sum can be = last row true values from (0 - totalSum).

s2= totalSum- s1; min = min(min, abs(s2- s1));

1. **Tabulation Solution:** //TC = O(n) + O(n\*K) + O(k/2)

//SC = O(n\*k)

**int minSubsetSumDifference(vector<int>& arr, int n)**

**{**

**int totalSum = 0;**

**for(auto it: arr)**

**totalSum += it;**

**int k = totalSum;**

**//make a dp table [n][k+1] using tabulation**

**vector<vector<bool>> dp(n, vector<bool>(k+1, 0));**

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

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

**if(arr[0]<= k) dp[0][arr[0]] = 1; //index = 0;**

**//build dp from (i+1-- n-1) & (1-- totalSum)**

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

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

**bool notTake = dp[ind-1][target];**

**bool take = 0;**

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

**take = dp[ind-1][target- arr[ind]];**

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

**}**

**}**

**//last row of dp contains 0 to totalSum which target can be achieved by subset 1**

**//so subset 1 sum can be = last row true values for target (0 - totalSum)**

**//s2= totalSum- s1; min = min(min, abs(s2- s1));**

**int mini = 1e9;**

**for(int i=0; i<=totalSum/2; i++){**

**if(dp[n-1][i] == 1){**

**int s1 = i;**

**int s2 = totalSum - s1;**

**mini = min(mini, abs(s2- s1));**

**}**

**}**

**return mini;**

**}**

1. **Space Optimization**: //TC = O(N) + O(N\*K) + O(K/2)

//SC = O(2K)

We can use a prev[k+1] to keep track of previous row values and curr[k+1] that will store current row values.

**//space optimized**

**int minSubsetSumDifference(vector<int>& arr, int n)**

**{**

**int totalSum = 0;**

**for(auto it: arr)**

**totalSum += it;**

**int k = totalSum;**

**//make a dp table [n][k+1] using tabulation**

**vector<bool> prev(k+1, 0), curr(k+1, 0) ;**

**//target = 0;**

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

**if(arr[0]<= k) prev[arr[0]] = 1; //index = 0;**

**//build dp from (i+1-- n-1) & (1-- totalSum)**

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

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

**bool notTake = prev[target];**

**bool take = 0;**

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

**take = prev[target- arr[ind]];**

**curr[target] = (notTake || take);**

**}**

**prev = curr;**

**}**

**//last row of dp contains 0 to totalSum which target can be achieved by subset 1**

**//so subset 1 sum can be = last row true values for target (0 - totalSum)**

**//s2= totalSum- s1; min = min(min, abs(s2- s1));**

**int mini = 1e9;**

**for(int i=0; i<=totalSum/2; i++){**

**if(prev[i] == 1){**

**int s1 = i;**

**int s2 = totalSum - s1;**

**mini = min(mini, abs(s2- s1));**

**}**

**}**

**return mini;**

**}**