Partition Equal Subset Sum

Problem link - https://leetcode.com/problems/partition-equal-subset-sum/

Problem statement

Given a non-empty array `nums` containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Example 1:

```
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into equal sum subsets.
```

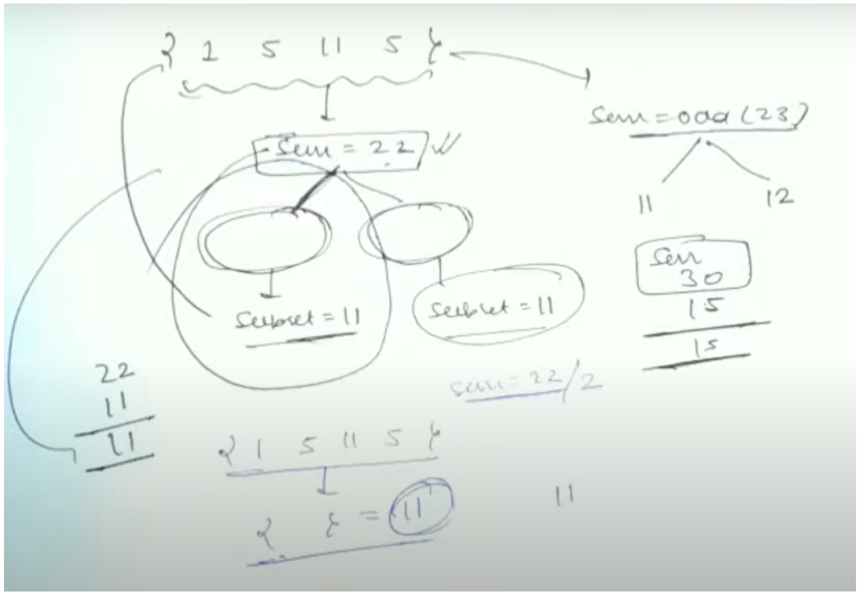
Method.

So the first insight we can see is that if the sum of all the numbers is even then only we can divide or partition in two different subsets. Let's suppose we have a sum of all the numbers that are odd then how can we divide into different subsets with equal sum one subset might be greater or lesser than other subset .
Let's understand with example
Arr = [1 , 5 , 11 , 5]
Sum = 22 , then chance that we can make two subset of sum 11 and 11.
But let's suppose total sum = 23 , then no chance of making two subset of equal sum, one might be 12 and other 11 or 11 and 12.

if(sum % 2 != 0) return false;

How this problem is related to subset sum.

Now our question breaks into that if Given a set of non-negative integers array , and a value sum / 2, determine if there is a subset of the given set with sum equal to given sum.
Why sum/2 ?
For sure we know that if you find one subset with half sum then other subset must be equal to half sum.



Now our question looks familiar to subset sum with given input , we can simply apply the subset concept on this problem. Because we have same input and same output

```
bool canPartition(vector<int>& nums) {
        long long sum = 0 ;
        for(int i =0; i<nums.size(); i++){
            sum += nums[i];
        }
        if(sum%2 != 0) return false;
        else {
            return subset(nums , nums.size() , sum/2);
        }
    }
```

Here we use long long because the sum might be larger.
And after that, calculating the total sum . And after that checking if the sum is even or odd , if odd then return else call the isSubsetSum function with these parameters and it returns true or false.

Time Complexity: O(sum*n)

// complete code

```cpp
class Solution {
public:
    bool subset(vector<int> nums , int n , int sum ){
         vector<vector<bool>> dp(n+1 , vector<bool>(sum+1 , false));
        for(int i = 0 , j= 0; i<n+1; i++){
            dp[i][j] = true;
        }
        for(int i = 1; i<n+1; i++){
             for(int j = 1; j<sum+1; j++){
                 if(nums[i-1] <= j){
                     dp[i][j] = ( dp[i-1][j-nums[i-1]] || dp[i-1][j] );
                 }else{
                     dp[i][j] = dp[i-1][j];
                 }
             }
        }
        return dp[n][sum];
    }
    bool canPartition(vector<int>& nums) {
        long long sum = 0 ;
        for(int i =0; i<nums.size(); i++){
            sum += nums[i];
        }
        if(sum%2 != 0) return false;
        else {
            return subset(nums , nums.size() , sum/2);
        }
    }
};
```