题目：

Given a **non-empty** array 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.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]


Output: true


Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]


Output: false


Explanation: The array cannot be partitioned into equal sum subsets.
```

1.时间：O（N*Sum）; 空间：O（N*Sum）

```
class Solution {
    /* dp[i][j]：利用 nums 在区间[0~i]上的数（限一次）, 在背包容量为 j 时，所能获得的最大重量；
        dp[i-1][j]：不使用 nums[i]
        dp[i-1][j-nums[i]] + nums[i]：使用 nums[i]
        dp[i][j] = max(dp[i-1][j], dp[i-1][j-nums[i]] + nums[i]) */
public:
    bool canPartition(vector<int>& nums) {
        if (nums.empty()) return false;
        int sum = std::accumulate(nums.begin(), nums.end(), 0);
        if (sum & 0x01) return false;
        sum = sum / 2;
        std::vector<std::vector<int>>                    dp(nums.size(), std::vector<int>(sum + 1, 0));
        for (int i = nums.front(); i < dp.front().size(); ++i) dp[0][i] = nums.front();
        for (int i = 1; i < nums.size(); ++i){
            for (int k = nums[i]; k <= sum; ++k){
                dp[i][k] = std::max(dp[i - 1][k], dp[i - 1][k - nums[i]] + k);
            }
        }
        return dp[nums.size() - 1][sum] == sum;
    }
};
```

2.时间：O（ ）; 空间：O（Sum）　 -->出错

```
class Solution {
    /* dp[i][j]：利用 nums 在区间[0~i]上的数（限一次）, 在背包容量为 j 时，所能获得的最大重量；
    dp[i-1][j]：不使用 nums[i]
    dp[i-1][j-nums[i]] + nums[i]：使用 nums[i]
    dp[i][j] = max(dp[i-1][j], dp[i-1][j-nums[i]] + nums[i]) */
public:
    bool canPartition(vector<int>& nums) {
        if (nums.empty()) return false;
        int sum = std::accumulate(nums.begin(), nums.end(), 0);
        if (sum & 0x01) return false;
        sum = sum / 2;
        std::vector<int> dp(sum + 1, 0);
        for (int i = nums.front(); i <= sum; ++i) dp[i] = nums.front();
        for (int i = 1; i < nums.size(); ++i){
            for (int k = nums[i]; k <= sum; ++k){
                dp[k] = std::max(dp[k], dp[k - nums[i]] + nums[i]);
    /* 数据例子：[1,2,5]，出错 */
            }
        }
        return dp[sum] == sum;
    }
};
```

3.时间：( ); 空间：O（Sum）

```
class Solution {
    /* dp[i][j]：利用 nums 在区间[0~i]上的数（限一次），在背包容量为 j 时，所能获得的最大重量；
    dp[i-1][j]：不使用 nums[i]
    dp[i-1][j-nums[i]] + nums[i]：使用 nums[i]
    dp[i][j] = max(dp[i-1][j], dp[i-1][j-nums[i]] + nums[i]) */
public:
    bool canPartition(vector<int>& nums) {
        if (nums.empty()) return false;
        int sum = std::accumulate(nums.begin(), nums.end(), 0);
        if (sum & 0x01) return false;
        sum = sum / 2;
        std::vector<int> dp(sum + 1, 0);
        for (int i = nums.front(); i <= sum; ++i) dp[i] =
nums.front();
        for (int i = 1; i < nums.size(); ++i){
            for (int k = sum; k >= nums[i]; --k){
                dp[k] = std::max(dp[k], dp[k - nums[i]] + nums[i]);
            }
        }
        return dp[sum] == sum;
    }
};
```

4.时间：O（ ）; 空间：O（N）　　-->dfs,超时

```
class Solution {
public:
    bool canPartition(vector<int>& nums) {
        if (nums.empty()) return false;
        int sum = std::accumulate(nums.begin(), nums.end(), 0);
        if (sum & 0x01) return false;
        std::sort(nums.begin(), nums.end());
        return dfs(nums, 0, sum / 2);
    }
private:
    bool dfs(const std::vector<int>& nums, int startIndex, int
target){
        if (target < 0) return false;
        if (target == 0) return true;
        for (int i = startIndex + 1; i < nums.size(); ++i){
            if (dfs(nums, i, target - nums[startIndex]) || dfs(nums, i, target)) return true;
            else break;
        }
        return false;
    }
};
```