Partition problem

An example of Dynamic programming

Question-

Given a set of integers find if it can be divided into subsets having equal sum.

Example:

S= {3, 1, 1, 2, 2, 1}

We can partition S into 2 partitions each having sum 5

S_1= {1, 1, 1, 2}

S_2= {2, 3}

This is not a unique solution. Below is another one

S_1= {3, 1, 1}

S_2= {2, 2, 1}

It is a special case of Subset Sum problem.

The idea is to calculate sum of all elements in the set. If sum id odd we can't divide the array into two sets. If sum is even we check if the subset with sum/2 exists or not.

We consider each item in the given array one by one and for each item, there are 2 possibilities-

1. We include current item in the subset and recur for remaining items with remaining sum.

2. We exclude current item in the subset and recur for remaining items.

Finally we return true if we get subset by including or excluding current item else we return false.

The base case of recursion would be when no items left or the subset sum is negative.

Recursion Algorithm:

```
bool subsetSum(int arr[], int n, int sum)
{
	// return true if sum becomes 0 (subset found)
	if (sum == 0)
		return true;

	// base case: no items left or sum becomes negative
	if (n < 0 || sum < 0)
		return false;

	// Case 1. include current item in the subset (arr[n]) and recur
	// for remaining items (n - 1) with remaining sum (sum - arr[n])
	bool include = subsetSum(arr, n - 1, sum - arr[n]);

	// return true if we get subset by including the current item
	if (include)
		return true;

	// Case 2. exclude current item n from subset and recur for
	// remaining items (n - 1)
	bool exclude = subsetSum(arr, n - 1, sum);

	// return true if we get subset by excluding the current item;
	// false otherwise
	return exclude;
}

bool partition(int arr[], int n)
{
	int sum = 0;
	for (int i = 0; i < n; i++)
		sum += arr[i];

	// return true if sum is even and array can can be divided into
	// two subsets with equal sum
	return !(sum & 1) && subsetSum(arr, n - 1, sum/2);
}
```

The time complexity of the above solution is exponential. The problem has an optimal substructure and it also exhibits overlapping sub problems, i.e. the problem can be split into smaller sub problems and same sub problems will get computed again and again. This can be proved by drawing recursion tree.

Memoization algorithm:

```
bool subsetSum(int arr[], int n, int sum)
{
	// T[i][j] stores true if subset with sum j can be attained with
	// using items up to first i items
	bool T[n + 1][sum + 1];
```

```
        // if 0 items in the list and sum is non-zero
        for (int j = 1; j <= sum; j++)
                T[0][j] = false;

        // if sum is zero
        for (int i = 0; i <= n; i++)
                T[i][0] = true;

        // do for ith item
        for (int i = 1; i <= n; i++)
        {
                // consider all sum from 1 to sum
                for (int j = 1; j <= sum; j++)
                {
                        // don't include ith element if j-arr[i-1] is negative
                        if (arr[i - 1] > j)
                                T[i][j] = T[i - 1][j];
                        else
                        // find subset with sum j by excluding or including the ith item
                                T[i][j] = T[i - 1][j] || T[i - 1][j - arr[i - 1]];
                }
        }

        // return maximum value
        return T[n][sum];
}

bool partition(int arr[], int n)
{
        int sum = 0;
        for (int i = 0; i < n; i++)
        sum += arr[i];

        // return true if sum is even and array can can be divided into
        // two subsets with equal sum
        return !(sum & 1) && subsetSum(arr, n, sum/2);
}
```

The time complexity of above solution is O(n x sum) where sum is the sum of all elements in the array.