Subset Sum problem

An example of Dynamic programming

Question-

Given a set of integers and an integer s, find if there exists a non-empty subset of the set of integers, the sum of whose elements is equal to s.

Example:

set={7,3,2,5,8}

s=14

output: Yes

subset {7,2,5} has sum 14

Naïve algorithm would be to cycle through all subsets of N numbers and for every one of them check if the subset sums to s. The running time is of the order $2^n * N$, since there are 2^n subsets and to check each subset we need to add N elements at the most.

A better exponential time algorithm uses Recursion. Subset sum can also be though of 0/1 Knapsack problem. For each item there are 2 possibilities-

1. We include the 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]);

        // 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 can get subset by including or excluding the
        // current item
        return include || exclude;
}
```

Memoization 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;

        // construct a unique map key from dynamic elements of the input
        string key = to_string(n) + "|" + to_string(sum);


        // if sub-problem is seen for the first time, solve it and
        // store its result in a map
        if (lookup.find(key) == lookup.end())
        {
                // 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]);

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

```
            // assign true if we can get subset by including or excluding the
            // current item
            lookup[key] = include || exclude;
}
```