Coin Change problem

An example of Dynamic programming

Question-

Given an unlimited supply of coins of given denominations, find the minimum number of coins required to get a desired change.

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn't matter.

Example:

N = 4 and S = {1,2,3}

There are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}

So output should be 4.

N = 10 and S = {2, 5, 3, 6}

There are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}

So the output should be 5.

It is a special case of Subset Sum problem.

1. The idea is use recursion to solve this problem.

2. For each coin of given denominations, we recur to see if total can be reached by including the coin or not.

3. If choosing the current coin resulted in solution we update the minimum coins needed.

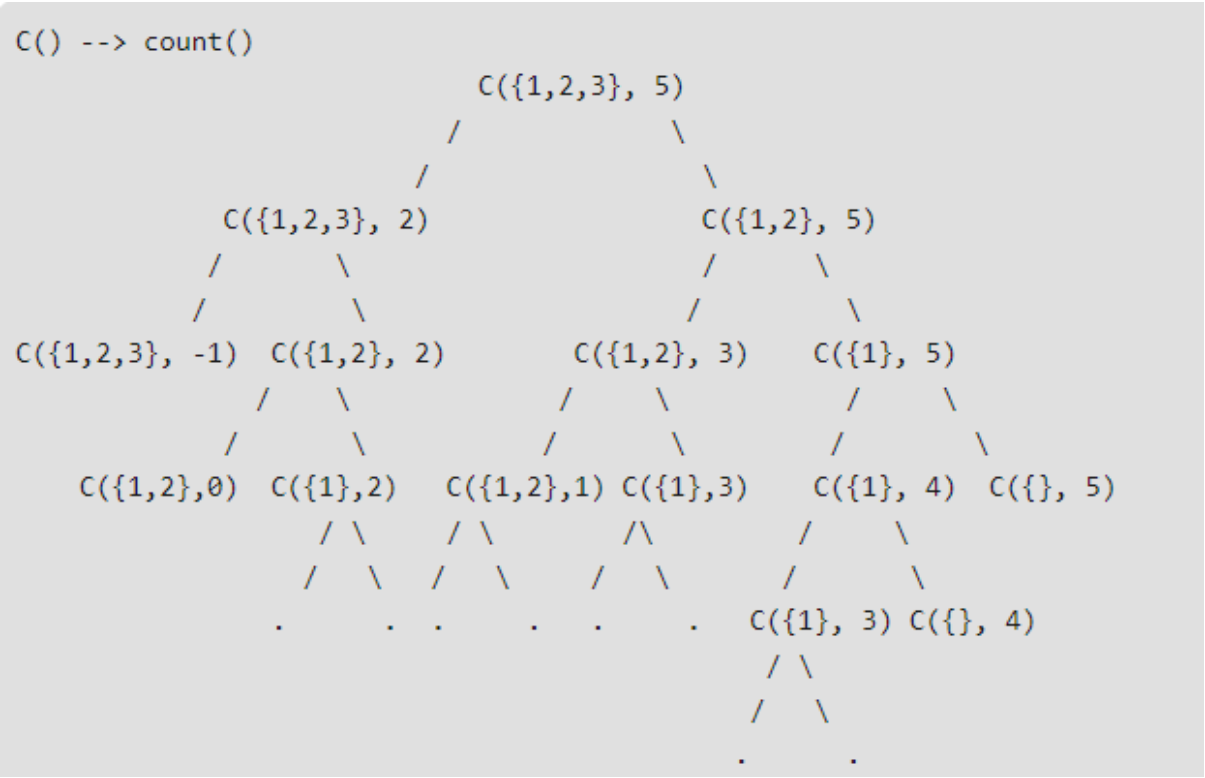
4. Finally we return the value we get after exhausting all combinations.

Recursion Algorithm:

```
int findMinCoins(int S[], int n, int N)
```

```
{
	// if total is 0, no coins are needed
	if (N == 0)
		return 0;

	// return INFINITY if total become negative
	if (N < 0)
		return INT_MAX;

	// initialize minimum number of coins needed to infinity
	int coins = INT_MAX;

	// do for each coin
	for (int i = 0; i < n; i++)
	{
		// recur to see if total can be reached by including
		// current coin S[i]
		int res = findMinCoins(S, n, N - S[i]);

		// update minimum number of coins needed if choosing current
		// coin resulted in solution
		if (res != INT_MAX)
			coins = min(coins, res + 1);
	}

	// return minimum number of coins needed
	return coins;
}
```

The time complexity of the above solution is exponential as each recursive call is making n recursive calls. The problem has an optimal substructure as the problem can be broken down into yet smaller sub problems and so on. It also has an overlapping substructure so we will end up solving the same sub problem again and again. We can see that by drawing the recursion tree.



Memoization algorithm:

```
int findMinCoins(int S[], int n, int N)
{
	// T[i] stores minimum number of coins needed to get total of i
	int T[N + 1];
	T[0] = 0;	// 0 coins are needed to get total of i

	for (int i = 1; i <= N; i++)
	{
		// initialize minimum number of coins needed to infinity
		T[i] = INT_MAX;
		int res = INT_MAX;

		// do for each coin
		for (int c = 0; c < n; c++)
		{
			// check if index doesn't become negative by including
			// current coin c
			if (i - S[c] >= 0)
				res = T[i - S[c]];

			// if total can be reached by including current coin c,
			// update minimum number of coins needed T[i]
			if (res != INT_MAX)
				T[i] = min(T[i], res + 1);
		}
	}

	// T[N] stores the minimum number of coins needed to get total of N
```

```
        return T[N];
}
```

The time complexity of above solution is O(n x N) ,where N is the total change required.