Coin Change :-

similar to 0-1 Knapsack Problem. (inclusion/exclusion principle)

Coin Change (Minimum Coins)

Difficulty: **Medium** Accuracy: **26.74%** Submissions: **247K+** Points: **4**

You are given an array **coins[]**, where each element represents a coin of a **different** denomination, and a target value **sum**. You have an **unlimited** supply of each coin type {coins1, coins2, ..., coinsm}.

Your task is to determine the **minimum** number of coins needed to obtain the target **sum**. If it is not possible to form the sum using the given coins, return **-1**.

Examples:

Input: coins[] = [25, 10, 5], sum = 30
Output: 2
Explanation: Minimum 2 coins needed, 25 and 5

Input: coins[] = [9, 6, 5, 1], sum = 19
Output: 3
Explanation: 19 = 9 + 9 + 1

Input: coins[] = [5, 1], sum = 0
Output: 0
Explanation: For 0 sum, we do not need a coin

Input: coins[] = [4, 6, 2], sum = 5
Output: -1
Explanation: Not possible to make the given sum.

As we know in 0-1 Knapsack problem, only single unit was given. But here, we have unlimited supply of items.

Brute force :-

parameters defined :-

(i, currSum)

Transition :- ans = INT_MAX

```
if(coins[i]<= currSum) { if (fn( i-1, currSum - coin[i]) != INT_MAX)) ans= 1 + x; }
ans= min(ans, fn(i-1, currSum)
   return ans ;
```

Base Case :-

```
if (currSum == 0) return 0;
   if(i<0) return INT_MAX;
```

```cpp
class Solution {
  public:
    int minCoins(vector<int> &coins, int sum) {
        // code here
        int n=coins.size();
        int ans=help(coins, n-1, sum);
        return ans==INT_MAX?-1:ans;
    }
    int help(vector<int> &coins, int i, int sum){
        if(sum==0)return 0;
        if(i<0)return INT_MAX;
        int ans=INT_MAX;
        if(sum>=coins[i]){
            int x=help(coins, i, sum-coins[i]);
            if(x!=INT_MAX)ans=1+x;
        }
        ans=min(ans, help(coins, i-1, sum));
        return ans;
    }
};
```

T.C. $O(2^n)$ S.C. $O(n)$ ← recursion stack

Optimized Solution :- Using Recursion + Memoization :-

T.C. O(size of coins[]) (sum)

S.C. O(size of coins[]) (sum) ← dp array

```cpp
// } Driver Code Ends

class Solution {
  public:
    int minCoins(vector<int> &coins, int sum) {
        // code here
        int n=coins.size();
        vector<vector<int>> dp(n, vector<int>(sum+1, -1));
        int ans=help(coins, n-1, sum, dp);
        return ans==INT_MAX?-1:ans;
    }
    int help(vector<int> &coins, int i, int sum, vector<vector<int>> &dp){
        if(sum==0)return 0;
        if(i<0)return INT_MAX;
        if(dp[i][sum]!=-1)return dp[i][sum];
        int ans=INT_MAX;
        if(sum>=coins[i]){
            int x=help(coins, i, sum-coins[i],  dp);
            if(x!=INT_MAX)ans=1+x;
        }
        ans=min(ans, help(coins, i-1, sum, dp));
        return dp[i][sum]=ans;
    }
};

```