**Problem link**: [Minimum Coins](https://www.codingninjas.com/codestudio/problems/minimum-elements_3843091?leftPanelTab=0)

**Approach**: We will try all combination of coins to get the min number of coins used having total sum = target.

Base cases:

* index==0, we can only take coins which can divide target. (12/4= 3 coins) else whatever the coins is can never sum up to target.

**1) recursion: //** tc = >>O(2^n) because we have 2 choices for every coin + single coin cab be chosen multiple times.

//sc = >>O(N) for recursion stack space

**//recursive soln**

**//TC = O(2^n)**

**int f(int ind, int target, vector<int> &nums){**

**if(ind==0){**

**if(target % nums[ind]==0)**

**return target/nums[ind];**

**else**

**return 1e9;**

**}**

**int notTake = 0 + f(ind-1, target, nums);**

**int take = INT\_MAX;**

**if(nums[ind]<= target)**

**take = 1 + f(ind, target-nums[ind], nums);**

**return min(take, notTake);**

**}**

**int minimumElements(vector<int> &nums, int x)**

**{**

**int n= nums.size();**

**int num = f(n-1, x, nums);**

**if(num==1e9)**

**return -1;**

**return num;**

**}**

2) **Memoization**: dp[n][target+1]

tc= O(n\* target)

sc = O(n\* target) + O(N)

**//Memoization soln**

**//TC = O(n\*target)**

**int f(int ind, int target, vector<int> &nums, vector<vector<int>> &dp){**

**if(ind==0){**

**if(target % nums[ind]==0)**

**return target/nums[ind];**

**else**

**return 1e9;**

**}**

**if(dp[ind][target] != -1) return dp[ind][target];**

**int notTake = 0 + f(ind-1, target, nums, dp);**

**int take = INT\_MAX;**

**if(nums[ind]<= target)**

**take = 1 + f(ind, target-nums[ind], nums, dp);**

**return dp[ind][target] = min(take, notTake);**

**}**

**int minimumElements(vector<int> &nums, int x)**

**{**

**int n= nums.size();**

**vector<vector<int>> dp(n, vector<int>(x+1, -1));**

**int num = f(n-1, x, nums, dp);**

**if(num==1e9)**

**return -1;**

**return num;**

**}**

3) **Tabulation**: build a dp[n][target+1] table from bottom to up.

tc = O(n\*target)

sc = O(n\*target)

**//tabulaion(space optimization)**

**//tc = O(n\*target)**

**//sc = O(n\*target)**

**int minimumElements(vector<int> &nums, int x)**

**{**

**int n= nums.size();**

**vector<vector<int>> dp(n, vector<int>(x+1, 0));**

**//index==0**

**for(int target=0; target<=x; target++){**

**if(target % nums[0]==0)**

**dp[0][target] = target/nums[0];**

**else**

**dp[0][target] = 1e9;**

**}**

**//dp**

**for(int ind=1; ind<n; ind++){**

**for(int target=1; target<=x; target++){**

**int notTake = 0 + dp[ind-1][target];**

**int take = INT\_MAX;**

**if(nums[ind]<= target)**

**take = 1 + dp[ind][target-nums[ind]];**

**dp[ind][target] = min(take, notTake);**

**}**

**}**

**int num = dp[n-1][x];**

**if(num>=1e9)**

**return -1;**

**return num;**

**}**

4) **Space optimization**:

* **Using Two arrays**: use prev[W+1] to store [i-1]th row values & curr[W+1] to store current row values.

**Two arrays:**

tc = O(n\*target)

sc = O(2\*target)

**//tabulaion(space optimization)**

**//tc = O(n\*target)**

**//sc = O(n\*target)**

**int minimumElements(vector<int> &nums, int x)**

**{**

**int n= nums.size();**

**vector<int> prev(x+1, 0), curr(x+1, 0);**

**//index==0**

**for(int target=0; target<=x; target++){**

**if(target % nums[0]==0)**

**prev[target] = target/nums[0];**

**else**

**prev[target] = 1e9;**

**}**

**//dp**

**for(int ind=1; ind<n; ind++){**

**for(int target=1; target<=x; target++){**

**int notTake = 0 + prev[target];**

**int take = INT\_MAX;**

**if(nums[ind]<= target)**

**take = 1 + curr[target-nums[ind]];**

**curr[target] = min(take, notTake);**

**}**

**prev = curr;**

**}**

**int num = prev[x];**

**if(num>=1e9)**

**return -1;**

**return num;**

**}**