**Problem link**: [0/1 Knapsack](https://www.codingninjas.com/codestudio/problems/0-1-knapsack_920542?source=youtube&campaign=striver_dp_videos&utm_source=youtube&utm_medium=affiliate&utm_campaign=striver_dp_videos&leftPanelTab=1)

**Approach**: We will try all combination of values to get the max value of items having total weight (<=W)

Base cases: i) index==0, we can take this element only if weight[0] <= W.

ii) W==0, that means bag is full, we can’t add value of items in it.

**1) recursion:** tc = O(2^n), sc = O(N) for recursion stack space

**//recursive soln**

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

**int f(int ind, int W, vector<int> &weight, vector<int> &value){**

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

**if(W >= weight[ind])**

**return value[ind];**

**return 0;**

**}**

**if(W==0) return 0; //bag is full, we cant add up items**

**int notTake = 0 + f(ind-1, W, weight, value);**

**int take = INT\_MIN;**

**if(weight[ind]<= W) take = value[ind] + f(ind-1, W-weight[ind], weight, value);**

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

**}**

**int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)**

**{**

**return f(n-1, maxWeight, weight, value);**

**}**

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

tc= O(n\* target)

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

**//Memoization soln**

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

**int f(int ind, int W, vector<int> &weight, vector<int> &value, vector<vector<int>> &dp){**

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

**if(W >= weight[ind])**

**return value[ind];**

**return 0;**

**}**

**if(W==0) return 0; //bag is full, we cant add up items**

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

**int notTake = 0 + f(ind-1, W, weight, value, dp);**

**int take = INT\_MIN;**

**if(weight[ind]<= W) take = value[ind] + f(ind-1, W-weight[ind], weight, value, dp);**

**return dp[ind][W] = max(notTake, take);**

**}**

**int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)**

**{**

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

**return f(n-1, maxWeight, weight, value, dp);**

**}**

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

tc = O(n\*W)

sc = O(n\*W)

**//Tabulation**

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

**int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)**

**{**

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

**for(int w= weight[0]; w<=maxWeight; w++) //index= 0**

**dp[0][w] = value[0];**

**//dp**

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

**for(int w=0; w<=maxWeight; w++){**

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

**int take = INT\_MIN;**

**if(weight[ind]<= w)**

**take = value[ind] + dp[ind-1][w-weight[ind]];**

**dp[ind][w] = max(notTake, take);**

**}**

**}**

**return dp[n-1][maxWeight];**

**}**

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.
* **Using Singlw array:** to calculate curr[w] we need curr[w] & curr[w-wt[ind]] so if we fill values from the back we can do this task using a single array because at a time to calculate curr[w] we need that value or the previous values not the next values.

**Two arrays:**

tc = O(n\*W)

sc = O(2\*W)

**//Tabulation(space optimised 1d)**

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

**int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)**

**{**

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

**for(int w= weight[0]; w<=maxWeight; w++) //index= 0**

**prev[w] = value[0];**

**//dp**

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

**for(int w=0; w<=maxWeight; w++){**

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

**int take = INT\_MIN;**

**if(weight[ind]<= w)**

**take = value[ind] + prev[w-weight[ind]];**

**curr[w] = max(notTake, take);**

**}**

**prev = curr;**

**}**

**return prev[maxWeight];**

**}**

**Single array:**

**//Tabulation(space optimised with single array)**

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

**//sc = O(W)**

**int knapsack(vector<int> weight, vector<int> value, int n, int maxWeight)**

**{**

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

**for(int w= weight[0]; w<=maxWeight; w++) //index= 0**

**prev[w] = value[0];**

**//dp**

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

**for(int w=maxWeight; w>=0; w--){**

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

**int take = INT\_MIN;**

**if(weight[ind]<= w)**

**take = value[ind] + prev[w-weight[ind]];**

**prev[w] = max(notTake, take);**

**}**

**}**

**return prev[maxWeight];**

**}**