

## 2. 0/1 Knapsack Problem (GFG)

# space Optimization sol.<sup>n</sup> (DP PROBLEM)

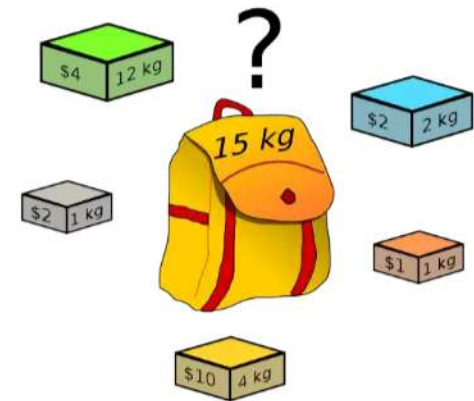
### Problem Statement:

Given  $N$  items where each item has some **weight** and **profit** associated with it and also given a **bag** with **capacity**  $W$ , (i.e., the bag can hold at most  $W$  weight in it).

### Return Kya Karana Hai:

The task is to put the items into the bag such that the **sum of profits** associated with them is the **maximum possible**.

**Note:** The constraint here is we can either **put an item completely into the bag** or cannot put it at all (It is not possible to put a part of an item into the bag).



Knapsack Problem

# **Approach 4: Space Optimization** Inclusive and Exclusive Pattern

## **DRY RUN**

Input:  
 $N = 3, W = 4, \text{weight[]} = \{4, 5, 1\}, \text{profit[]} = \{1, 2, 3\}$   
 Output: 3

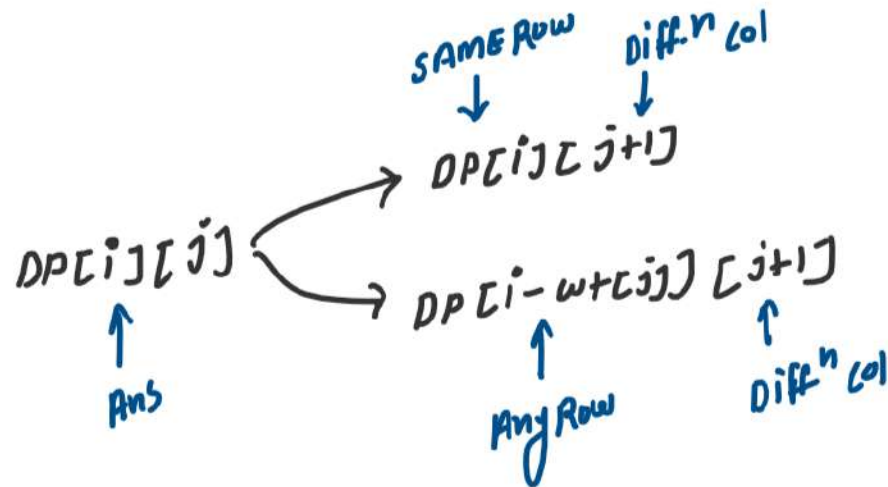


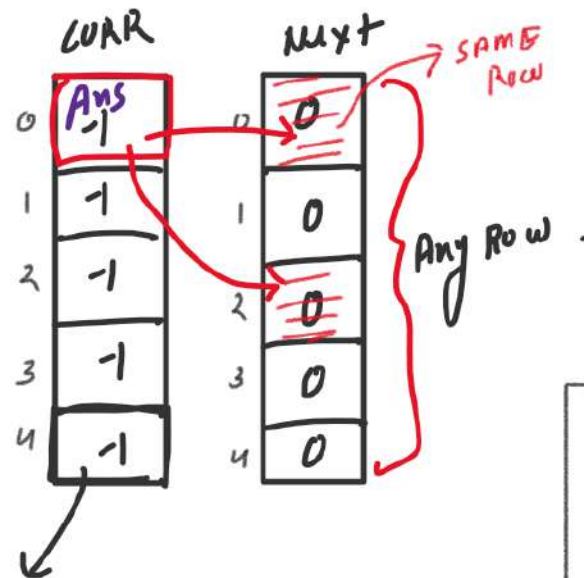
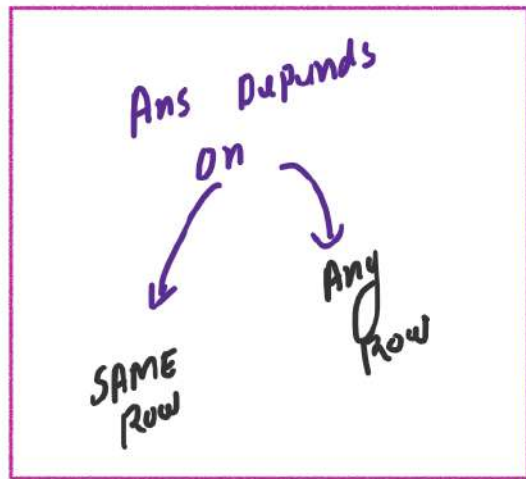
Diagram illustrating the DP table for the knapsack problem with space optimization:

Row Capacity (i) and Col Items N(j)

	Col 0	Col 1	Col 2	Col 3
Row 0	-1	-1	-1	0
Row 1	-1	-1	-1	0
Row 2	-1	-1	Ans	0
Row 3	-1	-1	-1	0
Row 4	-1	-1	-1	0

Annotations:

- Row 0 is labeled as **Row Capacity (i)**.
- Col 0 is labeled as **Col Items N(j)**.
- Col 2 is labeled as **Cur** (Current).
- Col 3 is labeled as **Next** (Next).
- Row 2, Col 2 is labeled as **Ans** (Answer).
- Row 2, Col 3 is labeled as **SAME ROW**.
- Row 3, Col 3 is labeled as **Any Row**.

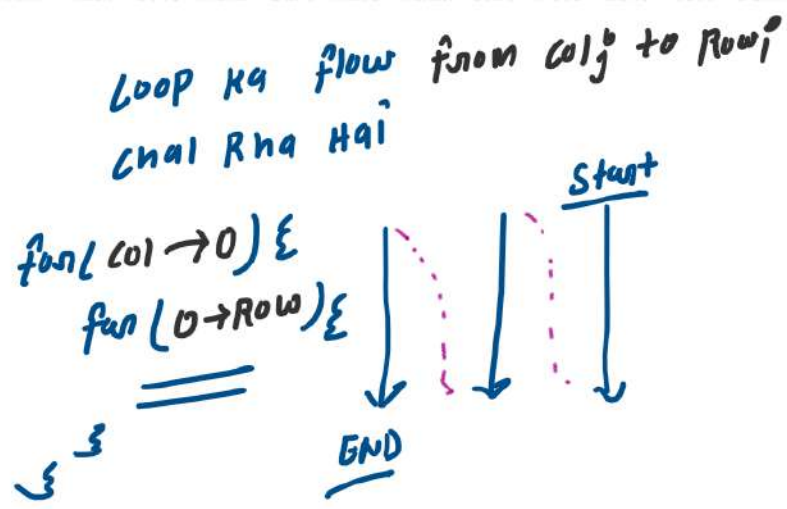


ARRAY CURR and Next ka size capacity + 1 Hona chahiye  
kyunki har EK Col per each Row ko fill krna pda Rha Hai.

jis Hisab se Loop chal Raha hai

Final Ans  
Yahaan per exist  
Karna

Why Loop  
Inten change?



```

// Problem 2: 0/1 Knapsack Problem (GFG)
// Approach 4: Space Optimization Approach

#include<iostream>
#include<vector>
using namespace std;

int solveUsingTabuS0(int capacity, int weight[], int profit[], int index, int n){
    vector<int> next(capacity+1, 0);
    vector<int> curr(capacity+1, -1);

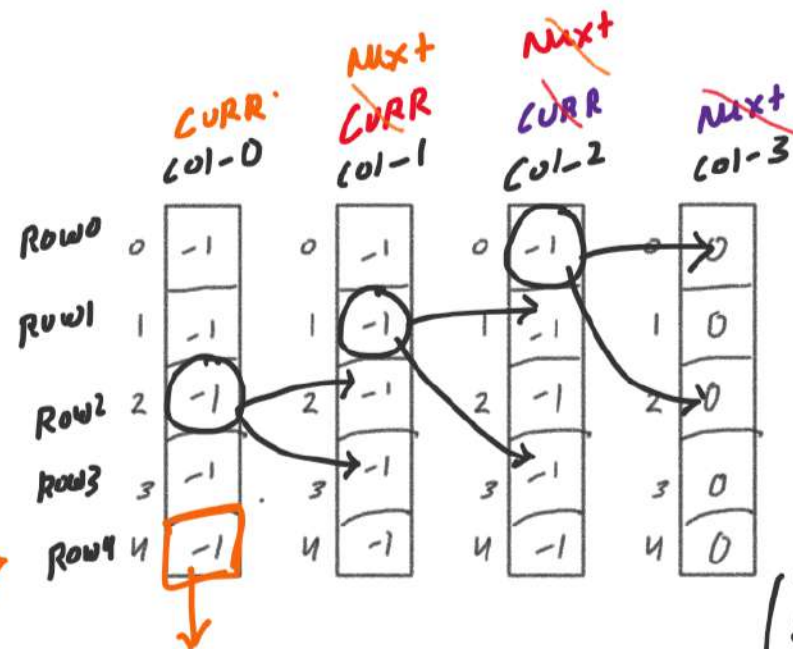
    for(int col=n-1; col>=0; col--){
        for(int row=0; row<=capacity; row++){
            // Recursive relation (inclusion or exclusion)
            int include = 0;
            if(weight[col] <= row){
                include = profit[col] + next[row - weight[col]];
            }
            int exclude = 0 + next[row];
            curr[row] = max(include, exclude);
        }
        // Shift Karna Bhoal Jata hu
        next = curr;
    }
    // return ans
    return curr[capacity];
}

int main(){
    int capacity = 6;
    int n = 3;
    int weight[] = {1,2,3};
    int profit[] = {10,15,40};
    int index = 0;

    int ans = solveUsingTabuS0(capacity, weight, profit, index, n);
    cout << "Max Profit: " << ans << endl;
    return 0;
}

```

→ why Return



[Max Profit  
loop ke according  
yahi hona chahiye]

shifting

Next = CURR

## Optimization Solution 2

Input:

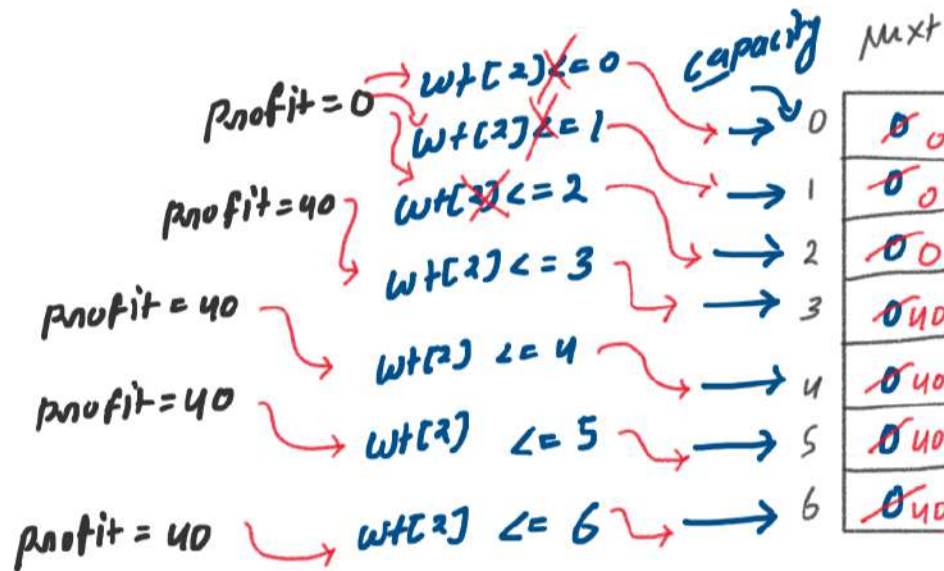
$N = 3, W = 6, \text{weight[]} = \{1, 2, 3\}, \text{profit[]} = \{10, 15, 40\}$

Output: 65

0 1 2

0 1 2

When item index = 2  
 $\rightarrow \text{wt}[2] = 3$



## Iteration 1

Inc =  $\text{profit}[\text{index}] + \text{max}[\text{capacity} - \text{wt}[\text{index}]]$

Excl =  $0 + \text{max}[\text{capacity}]$

$\text{max}[\text{capacity}] = \max \text{ of Inc and Excl}$



## Iteration 2

**Input:**

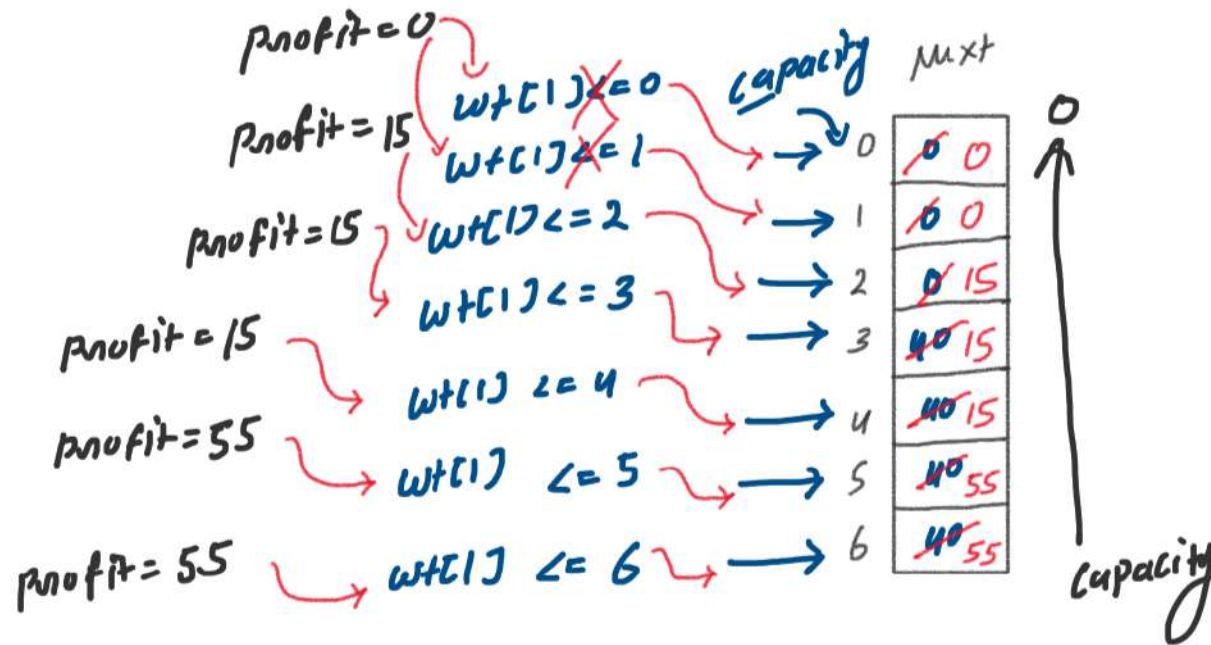
$N = 3, W = 6, \text{ weight}[] = \{1, 2, 3\}, \text{ profit}[] = \{10, 15, 40\}$

**Output:** 65

0 1 2

0 1 2

When item index = 1  
 $\rightarrow wt[2] = 2$



# Iteration 3

Input:

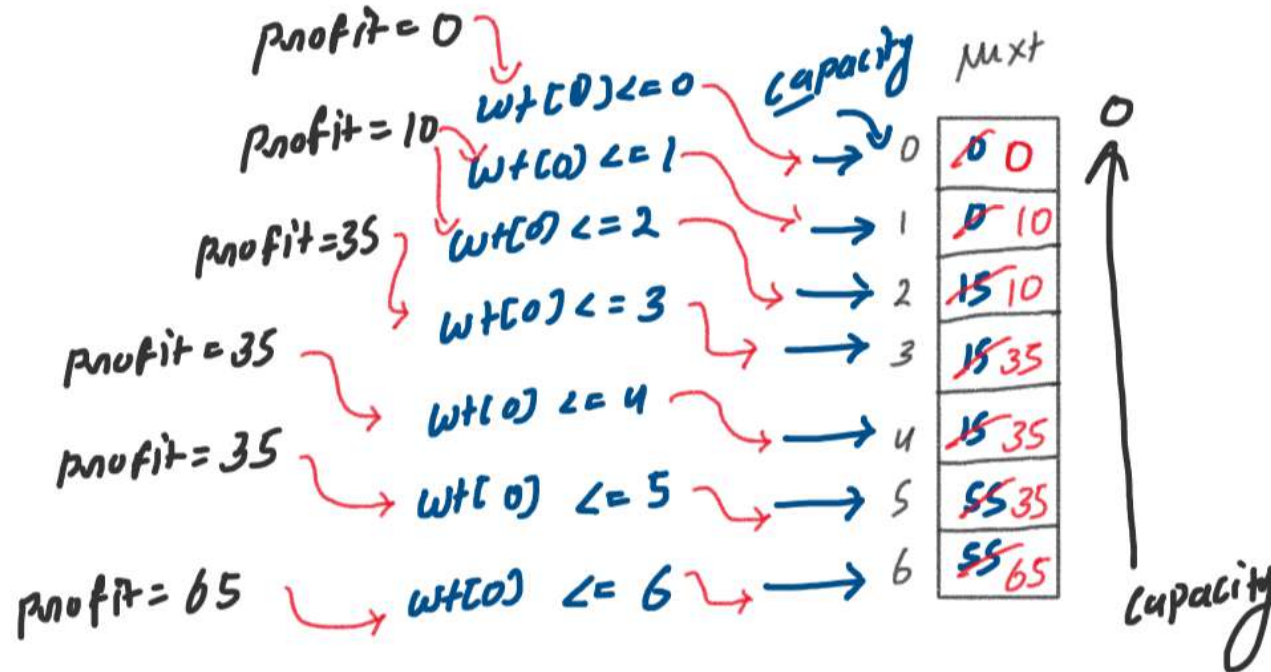
$N = 3, W = 6, \text{weight}[] = \{1, 2, 3\}, \text{profit}[] = \{10, 15, 40\}$

Output: 65

0 1 2

0 1 2

When item index = 0  
 $\rightarrow \text{wt}[0] = 1$



```

// Problem 2: 0/1 Knapsack Problem (GFG)
// Approach 5: Space Optimization Approach 2

#include<iostream>
#include<vector>
using namespace std;

int solveUsingTabuS02(int capacity, int weight[], int profit[], int index, int n){
    vector<int> next(capacity+1, 0);

    for(int col=n-1; col>=0; col--){
        for(int row=capacity; row>=0; row--){
            // Recursive relation (Inclusion or exclusion)
            int include = 0;
            if(weight[col] <= row){
                include = profit[col] + next[row - weight[col]];
            }
            int exclude = 0 + next[row];
            next[row] = max(include, exclude);
        }
    }
    // return ans
    return next[capacity];
}

int main(){
    int capacity = 50;
    int n = 3;
    int weight[] = {10, 20, 30};
    int profit[] = {60, 100, 120};
    int index = 0;

    int ans = solveUsingTabuS02(capacity, weight, profit, index, n);
    cout << "Max Profit: " << ans << endl;
    return 0;
}

```

Index = -1  
↳ STOP

This is maximum  
Profit  
return next  
[capacity]

next

0	0
1	10
2	10
3	35
4	35
5	35
6	65