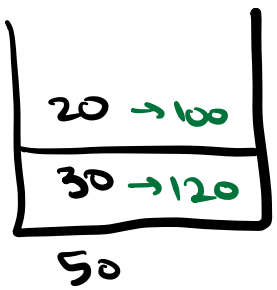
- Fractional Knapsack
- 0-1 Knapsack
- Unbounded Knapsack
- Minimum Difference

Given N items with profit $v_i$ and weight $w_i$. A bag is given with capacity W to carry some objects such that total weight $\leq W$ and maximize profit in bag.

1) Fractional knapsack (can pick every item only once and pick the item fractionally)

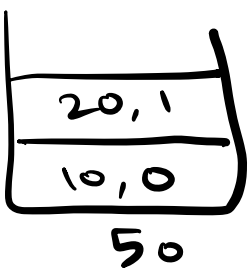
| | | 0 | 1 | 2 |
|---|---|---|---|---|
| N = 3 | $w_i$ | 10 | 20 | 30 |
| W = 50 | $v_i$ | 60 | 100 | 120 |
| | Profit per kg | 6 | 5 | 4 |

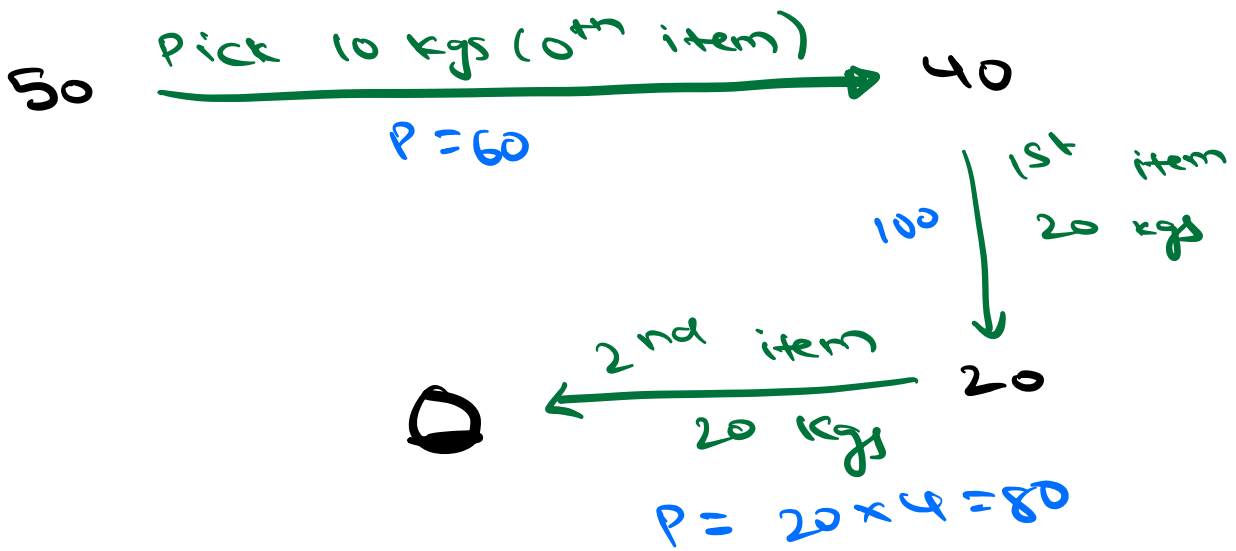
profit = 220




10 kg → 60

1 kg → $\frac{v_i}{w_i}$




max Profit → 240

Idea : Calculate profit per kg for every item. Sort items based on the ratio

```
class Item {
    int wt
    int val
    double profitperunit
}
```

1. Create an array of items and populate it
2. sort (items) → descending order of profit ratio
3. double ans = 0

```
for (i=0 ; i < n ; i++) {
    if (items[i].wt ≤ W) {
        ans += items[i].profitperunit * W
        W = 0
        break
    }
    else { // ith item is put in bag
        ans += items[i].val
        W = W - items[i].wt
    }
}
return ans
```

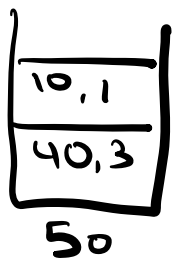
TC: O(NlogN)

SC: O(N)

# 0/1 Knapsack

- You can pick every item only once.
- You will either pick an item or not.

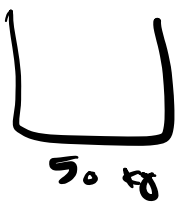
| | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| N = 4 | Wt: | 20 | 10 | 30 | 40 |
| W = 50 | V: | 100 | 60 | 120 | 150 |
| | Profit per unit | 5 | 6 | 4 | 3.75 |




Idea 1: Choose by max V

<3, 1> X

150 + 60 = 210




Idea 2: Choose by max profit per unit

<1, 0> X

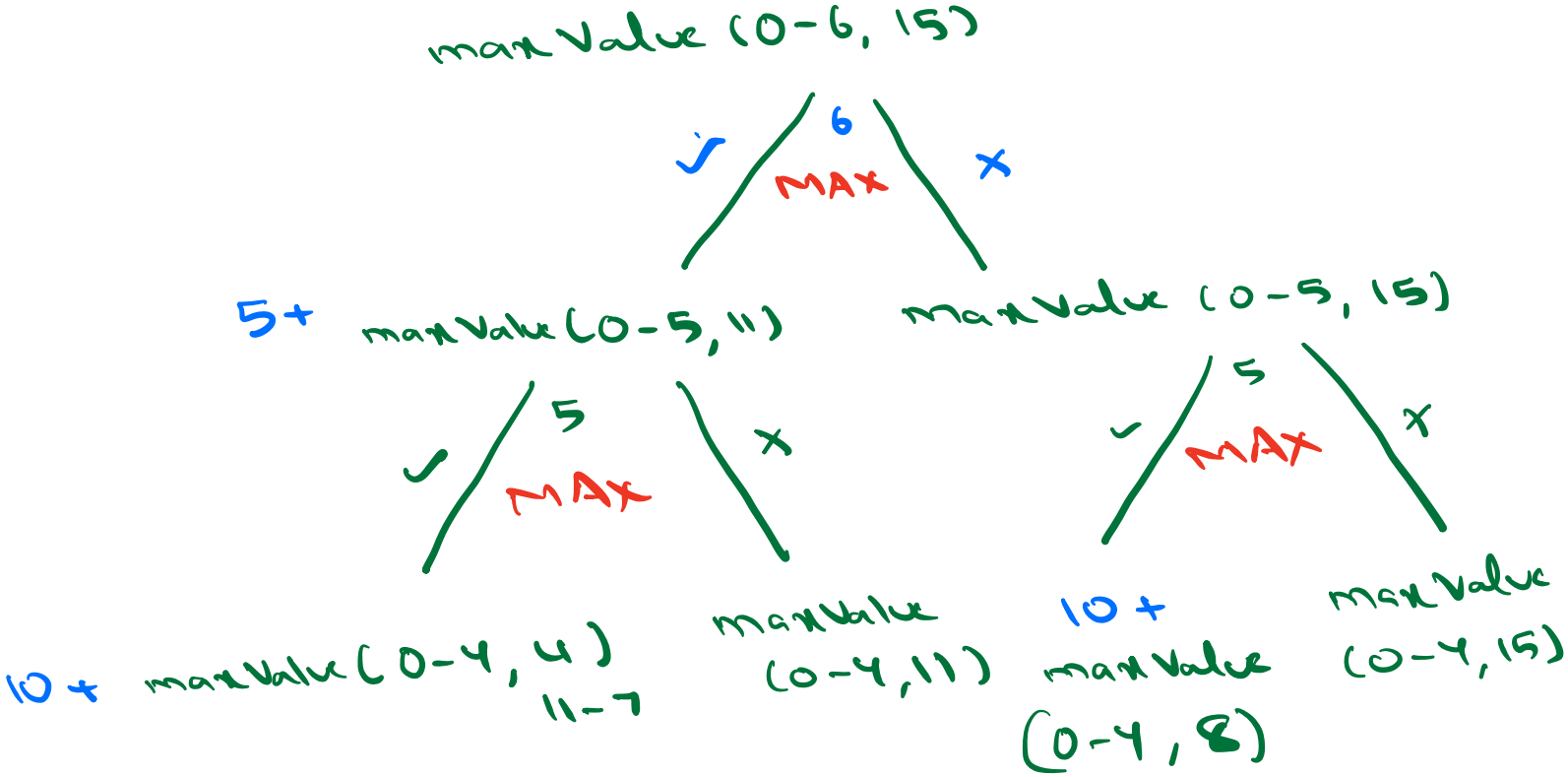
60 + 100 = 160

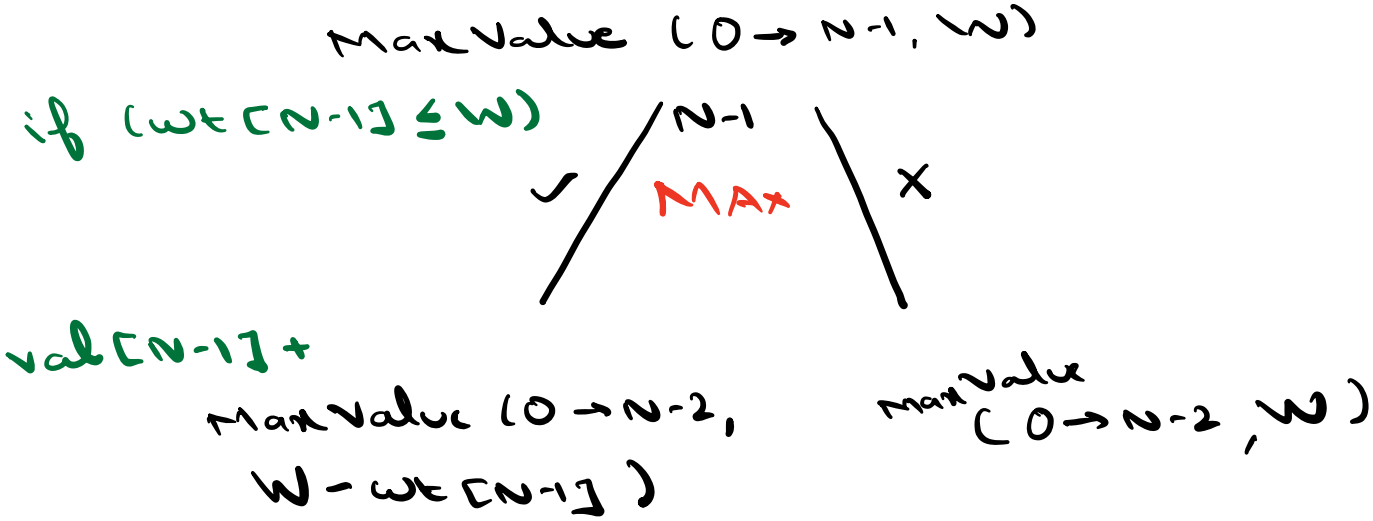

ans → <0, 2>

100 + 120 = 220

Idea 3: Go to all subsets with weight ≤ W and take max profit

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| N = 7 | Wt | 4 | 1 | 5 | 4 | 3 | 7 | 4 |
| W = 15 | V | 3 | 2 | 8 | 3 | 7 | 10 | 5 |




MaxValue (end, W)

Overlapping subproblems

N = 12
W = 50

Wt = _ _ _ _ _ _ 5 3 8 9 2 4
V = _ _ _ _ _ _ _ _ _ _ _ _
✗ ✗ ✗ ✓ ✓ ✗
✓ ✗ ✗ ✗ ✓ ✓

maxValue (12, 50)

→ maxValue (6, 39)  maxValue (6, 39)


End index, W

end idx
dp [N] [W+1]

0-4, 10

Base Cases

if W == 0 ⇒ no space value = 0

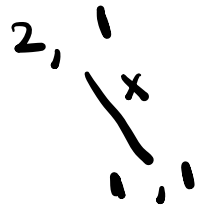
if E == -1 / N == 0 ⇒ no items value = 0

no of items
↓
dp [i] [j] = max value we can get from first i items in a bag of j weight
↑
capacity

dp[5+1][8+1] = dp[6][9]

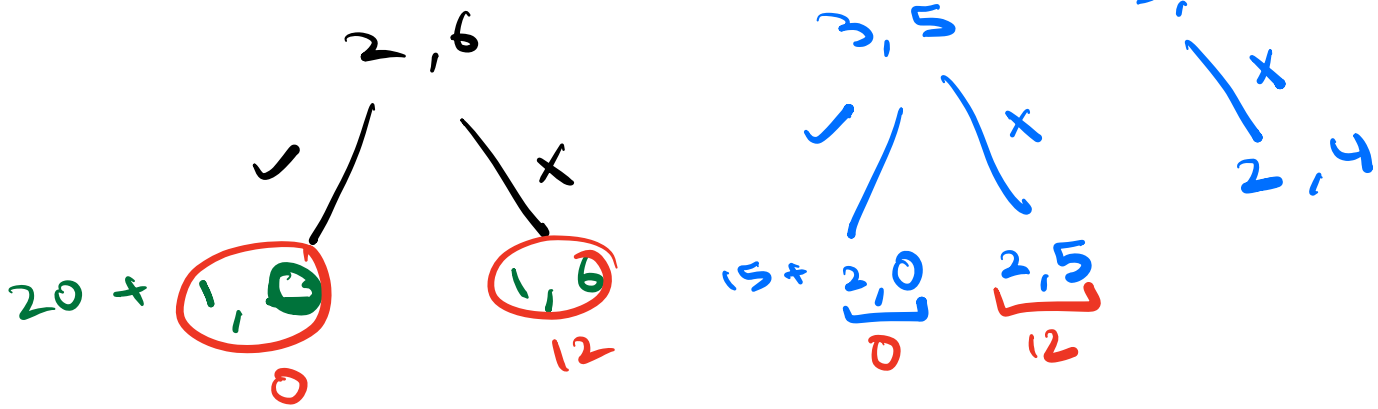
N = 5
W = 8

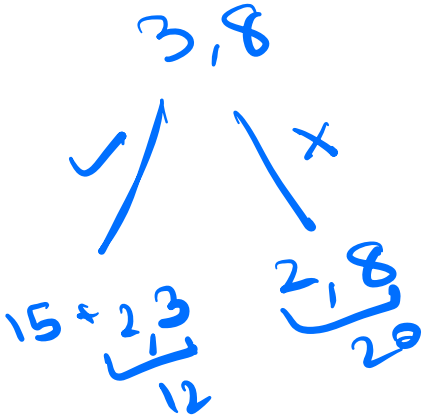
| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| wt | 3 | 6 | 5 | 2 | 4 |
| v | 12 | 20 | 15 | 6 | 10 |

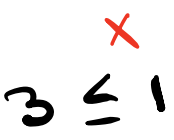


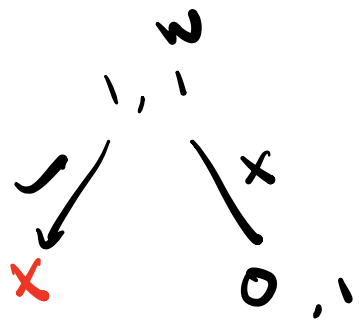

capacity / W →

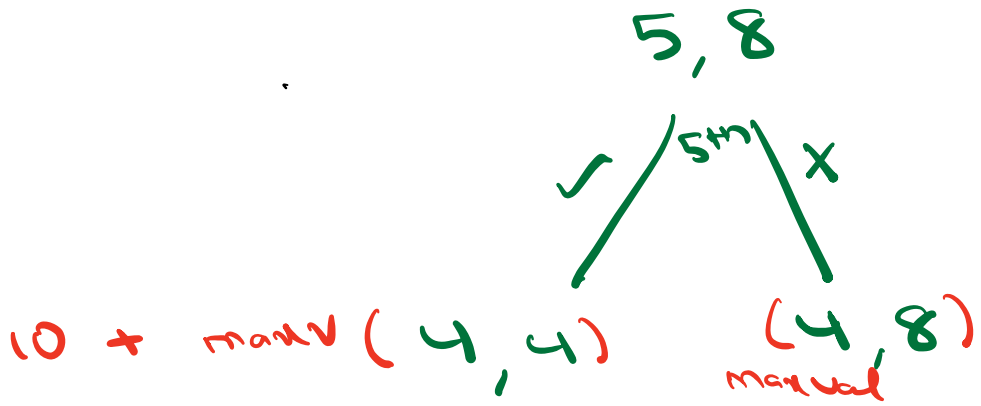
| val | wt | cnt ↓ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 3 | 1 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 |
| 20 | 6 | 2 | 0 | 0 | 0 | 12 | 12 | 12 | 20 | 20 | 20 |
| 15 | 5 | 3 | 0 | 0 | 0 | 12 | 12 | 15 | 20 | 20 | 27 |
| 6 | 2 | 4 | 0 | 0 | 6 | 12 | 12 | 18 | 20 | 21 | 27 |
| 10 | 4 | 5 | 0 | 0 | 6 | 12 | 12 | 18 | 20 | 22 | 27 |

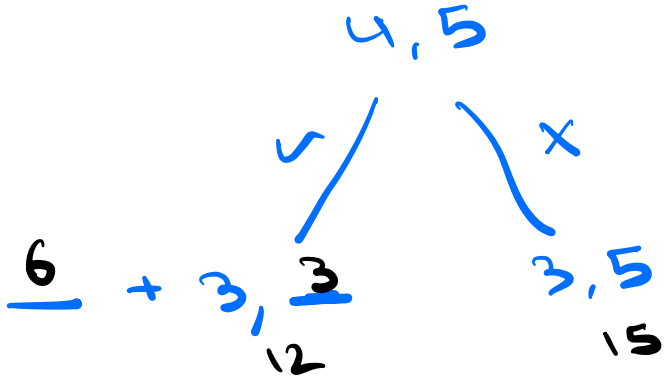



5, 8
✓
SM
X
10 + max ( 4, 4 )
( 4, 8 )
max val


4, 5
✓
X
6
+ 3, 3
12
3, 5
15

```
// N, W, wt[], val[]
int dp[N+1][W+1]
```

dp[i][j] → max value from first i items in a bag of j weight

```
for (i=0; i ≤ N; i++) {
  for (j=0; j ≤ W; j++) {
    if (i==0 || j==0)
      dp[i][j] = 0
    else {
      // i items, j weight
      int exclude = dp[i-1][j]
      int include = 0
      if (wt[i-1] ≤ j) {
        include = val[i-1] +
                  dp[i-1][j-wt[i-1]]
      }
      dp[i][j] = max(exclude, include)
    }
  }
}
return dp[N][W]
```


TC : O(N * W)

SC : O(N * W)

11:15

# Unbounded Knapsack (0-∞ Knapsack)

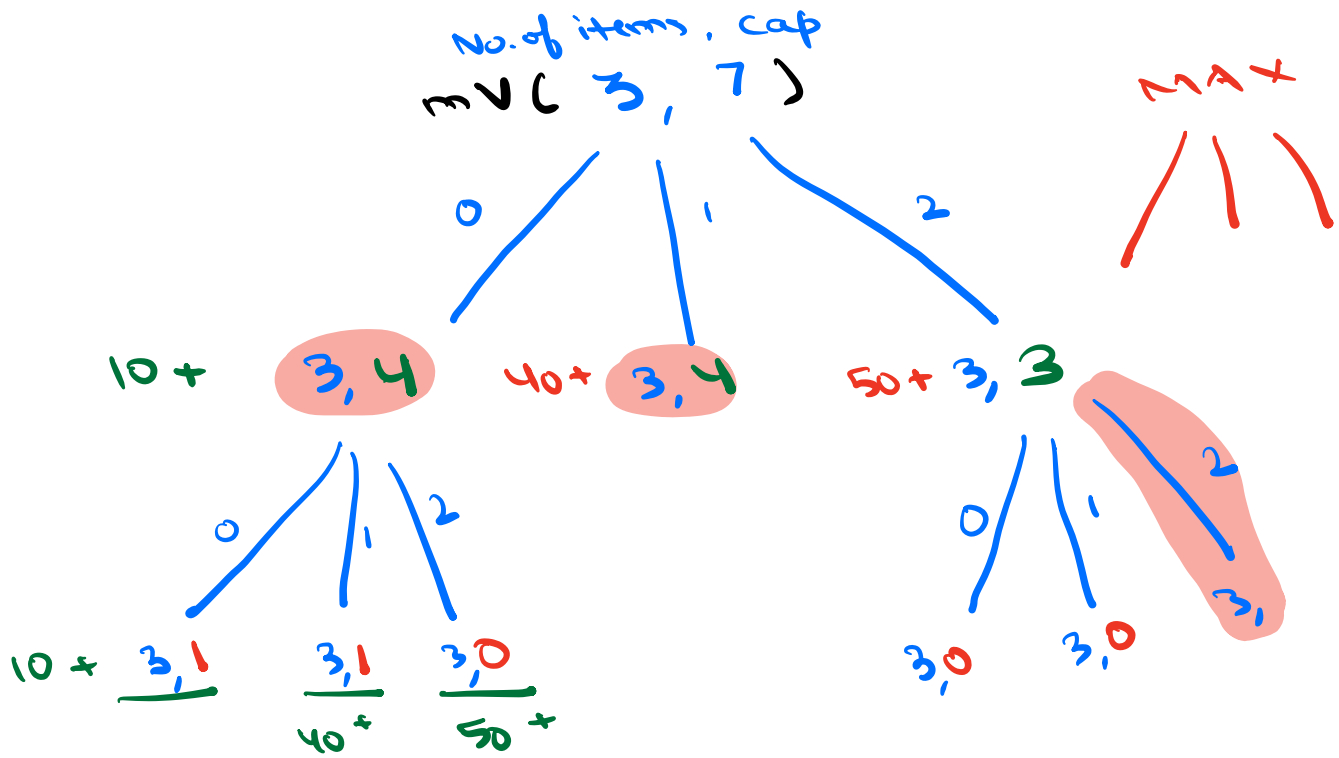
- Can select 1 item any no. of times
- You can't select any item partially

| | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| N = 4 | wt | 20 | 13 | 10 | 40 |
| W = 50 | val | 100 | 66 | 40 | 150 |

< 0, 0, 2 >
100 + 100 + 40

ans → 240

| | | 0 | 1 | 2 |
|---|---|---|---|---|
| N = 3 | val | 10 | 40 | 50 |
| W = 7 | wt | 3 | 3 | 4 |




dp[W+1]

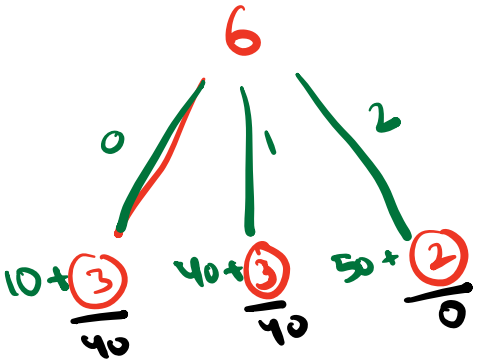
1D → weight of knapsack

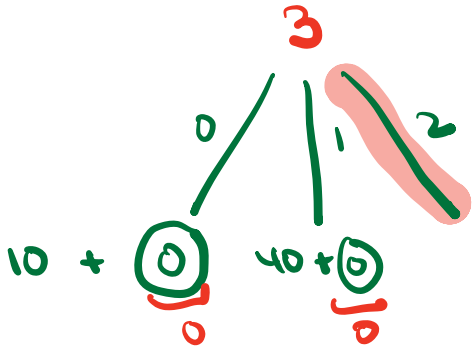
dp[i] → Max profit that we can get in a bag of i weight

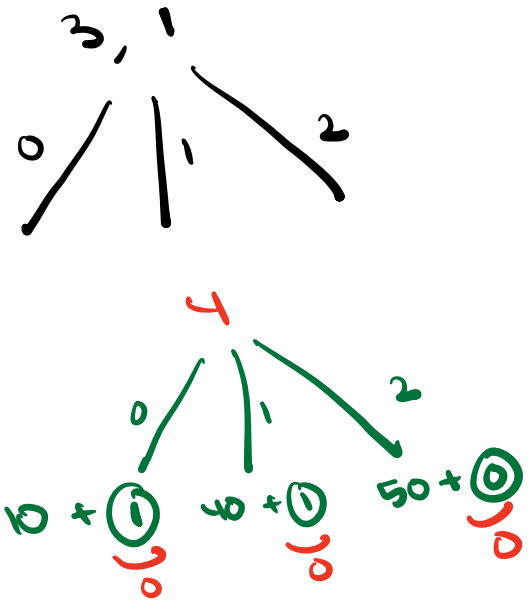
| | | 0 | 1 | 2 |
|---|---|---|---|---|
| N = 3 | val | 10 | 40 | 50 |
| W = 7 | wt | 3 | 3 | 4 |

dp[7+1]

| idx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| dp[] | 0 | 0 | 0 | 40 | 50 | 50 | 80 | 90 |




```
int dp[W+1]
dp[0] = 0

for (int i=1; i<=W; i++) {
    // i -> capacity of bag
    int maxValue = 0
    for (int j=0; j<N; j++){
        if (wt[j] <= i)
            maxValue = max(maxValue,
                           val[j] + dp[i - wt[j]])
    }
    dp[i] = maxValue
}
return dp[W]
```

TC : O(W)

SC : O(W)