IT-081 Pankhania Aanandi R.
DAA -

Knapsack Algo.

i) Analysis of the problem:-

i/p :- total no. of objects & their weights will be given & capacity of the knapsack i.e W max. & also each objects profit.

process :- We have to calculate which object we should place in knapsack so that we get the max. profit.

o/p :-

s[i,j] : max value of knapsack of [w] weight capacity j with ith object.

e.g:- Max. capacity of knapsack = 5
Total items = n = 4

item i	1	2	3	4
value val	100	20	60	40
weight wt	3	2	4	1

creating value table V[i, wt] when
i → no. of items
wt → weight of items.
Row → items
col → weight.

4 items ⇒ 5 rows i.e. 0 to 4.

as W = 5 so we have 6 cols i.e 0 to 5

→ fill in first row i=0 with 0. ⇒ 0 item considered so weight = 0.
i=0 fill col. w=0 means when weigh 0 items = 0.

Rule to fill the [V[i, w]] table:-

if wt[i] > w then v[i, w] = V[i-1, w]
else if wt[i] <= w then
v[i, w] = max (V[i-1, w], val[i] + V[i-1, w-wt[i]])

after calculation, value table :-

V[i, w]	w=0	1	2	3	4	5
i=0						
0	0	0	0	0	0	0
1	0	0	0	100	100	100
2	0	0	20	100	100	120
3	0	0	20	100	100	120
4	0	40	40	100	140	140

max value earned Max Value = V[n, W]

= V[4, 5]

= 140

// items that were put inside the knapsack are found using the following rule.

set i=n & w=W

while i and w > 0 do
    if V[i,w] != V[i-1, w] then
        mark the item (ith)
        set w = w - wt[i]
        set i = i-1
    else
        set i = i-1
    endif
endwhile

Algorithm: (final combined)

```
Algorithm knapsack (value [1...n]
                    weight [1...n]
                    n, maxWeight)

for i ← 1 to n
    for j ← 1 to maxweight
        if (j < weight[i])
            b[i][j] = b[i-1][j]
        else
            b[i][j] = max { b[i-1][j],
                            { b[i-1][j - weight[i]] + v[i]
        // b[i-1][j], b[i-1][j - wt[i]] + v[i]
return b[n, maxWeight].
```

Time Complexity :- $O(n * W)$

n → no. of items

W → total weight

Space Complexity :- $O(W)$

W → total weight

time complexity will be n.W for this as from our code (Algo.)