P6 a) Implement in Java, the 0/1 knapsack problem using Dynamic Programming method.

```
import java.util.*;
public class knapsack
 {
   public void solve(int[] wt, int[] val, int W,
     int n){
   int i, j;
   int sol[][] = new int[n+1][W+1];
   for (i=0; i<=n; i++)
   { for (j=0; j<=W; j++)
     { if (i==0 || j==0)
         sol[i][j] = 0;
       else if (wt[i] > j)
         sol[i][j] = sol[i-1][j];
       else
         sol[i][j] = Math.max(sol[i-1][j],
         (sol[i-1][j-wt[i]] + val[i]));
   } }
   System.out.println("The optimal solution
     is: "+ sol[N][W]);
   int[] selected = new int[N+1];
   for (i=0; i<N+1; i++)
     selected[i]=0;
```

```
i=N; j=W;
while (i>0 && j>0)
{
    if (sol[i][j] != sol[i-1][j]) {
        selected[i] = 1;
        j = j - w[i];
    }
    i--
}
System.out.println("\n Items Selected: ");
for (i=1; i<n+1; i++)
    if(selected[i] == 1)
        System.out.print(i+" ");
}

public static void main(String[] args)
{
    Scanner in = new Scanner(System.in);
    Knapsack ks = new Knapsack();

    System.out.println("Enter no. of Elements: ");
    int n = in.nextInt();
    int[] wt = new int[n+1];
    int[] val = new int[n+1];
```

```
System.out.println("\n Enter the weights: ");
for(int i=1; i<=n; i++)
    w[i] = in.nextInt();
System.out.println("\n Enter knapsack weight: ");
int W = in.nextInt();
System.out.println(" Enter values: ");
for(int i=1; i<=n; i++)
    v[i] = in.nextInt();
ks.solve(wt, val, W, n); }}
```

Output:

```
Enter number of Elements
4
Enter weights
2 1 3 2
Enter knapsack weight
5
Enter values
12 10 20 15
The optimal solution is: 37
Items selected:
1 2 4
```

| Item | weight | value |
|---|---|---|
| 1 | 2 | $12 |
| 2 | 1 | $10 |
| 3 | 3 | $20 |
| 4 | 2 | $15 |

capacity $w = 5$

How to decide on total no. of rows & columns?

no. of rows = no. of items + 1

no. of colums = capacity + 1

here rows = 4+1 = 5 col = 5+1 = 6

(considered) when capacity of bag = 0 → can't fill anything

| i \ capacity j | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 12 | 12 | 12 | 12 |
| 2 | 0 | 10 | 12 | 22 | 22 | 22 |
| 3 | 0 | 10 | 12 | 22 | 30 | 32 |
| 4 | 0 | 10 | 15 | 25 | 30 | 37 |
| 5 | 0 | | | | | |

$w_1 = 2$ $v_1 = 12$

$w_2 = 1$ $v_2 = 10$

$w_3 = 3$ $v_3 = 20$

$w_4 = 2$ $v_4 = 15$

$w_5 =$

Saathi

Date ___/___/___

V[4,5]

1) V[1,1] check j-wt ⇒ 1-2 = -1 ≯ 0

V[1,1] = V[0,1] = 0

[i.e capacity = 1, weight of item = 2, we can't fill (we can't consider part of item)]

2] V[1,2] j-w[1] ⇒ 2-2 ≥ 0

max{V[0,2], v2 + V[0,0]}

max{0, 12} = 12

V[1,2] = 12

1 Item, cap = 2
wt = 2, cap = 2
can be filled
Profit = profit of item

3] V[1,3] j-w[1] ⇒ 3-2 ≥ 0 ⇒ 1 ≥ 0

max[V[0,3], 12 + V[0,1]] = 12

V[1,3] = 12.

4] V[1,4] j-w[1] ⇒ 4-2 ⇒ 2 ≥ 0

max{V[0,4], v1 + V[0,2]}

max{0, 12+0} = 12

5] V[1,5] j-w[1] ⇒ 5-2 ⇒ 3 ≥ 0

max{V[0,5], v1 + V[0,3]}

max{0, 12+0} = 12

Page No.

Saathi

Date ___/___/_____

⇒ 1) V[2,1] $j - w_2 = 1 - 1 \geq 0 \Rightarrow 0 \geq 0$ ⇒ true.

max[V[1,1], $v_2$ + V[1,0]]

max[0, 10+0] = 10

2) V[2,2] $2 - 1 = 1 \geq 0$

max[V[1,2], $v_2$ + V[1,1]]

max[12, 10+0] = 12

3) V[2,3] $3 - 1 = 2 \geq 0$

max[V[1,3], $v_2$ + V[1,2]]

max[12, 10+12] = max[12, 22] = 22

4] V[2,4] $4 - 1 = 3 \geq 0$

max[V[1,4], 10 + V[1,3]] = max[12, 10+12] = 22

5] V[2,5] $5 - 1 = 4 \geq 0$

max[V[1,5], 10 + V[1,4]] = max[12, 10+12] = 22

⇒ 1] V[3,1] $1 - w_3 = 1 - 3 = -2 \not\geq 0$

V[2,1] = 10

2] V[3,2] $2 - 3 = -1 \not\geq 0$

V[2,2] = 12

3] V[3,3] $3 - 3 = 0 \geq 0$

Page No.

max[ V[2,3] , $v_3$ + V[2,0] ]
max[ 22, 20 + 0 ] = 22

4] V[3,4] 4 - 3 = 1 ≥ 0
max[ V[2,4] , $v_3$ + V[2,1] ]
max[ 22, 20 + 10 ] = 30

5] V[3,5] 5 - 3 = 2 ≥ 0
max[ V[2,5] , $v_3$ + V[2,2] ]
max[ 22, 20 + 12 ] = 32

⇒ 1] V[4,1] 1 - $w_4$ = 1 - 2 = -1 ≱ 0
V[3,1] = 10

2] V[4,2] 2 - 2 = 0 ≥ 0
max[ V[3,2] , $v_4$ + V[3,0] ]
max[ 12, 15 + 0 ] = 15

3] V[4,3] 3 - 2 = 1 ≥ 0
max[ V[3,3] , $v_4$ + V[3,1] ]
max[ 22, 15 + 10 ] = 25

4] V[4,4] 4 - 2 ≥ 0 ⇒ 2 ≥ 0
max[ V[3,4] , $v_4$ + V[3,2] ] = V[30, 15 + 12] = 30

Saathi

Date ___/___/_____

5) V[4,5] 5-2 = 3 > 0

max[ V[3,5], V4 + V[3,3] ]

max[ 32, 15 + 22 ] = 37

Back tracking:

↳ Start from the last value & compare it with previous row (same col.) value.

If values are different ⇒ last value is selected to put in bag. (repeat) ←

↳ check for remaining cap. & go to that col & compare item

*(a) here last value = 37 37 & 32 ⇒ are different

Item 4 is selected.

(b) Now check if 32 (item 3 can be selected)

total cap = 5 , item 4 = 2

rem. cap = 5 - 2 = 3

Now check col. 3 of row 3 (prev. row of item 4)

i.e 22

* Now 22 is compared with V[2,3] = 22

values are not different. Hence item 3 is not selected.

remaining capacity

*(a) Now check col (3) of row 2

22 & 12 are different. Item 2 is selected

Page No.

Saat

Date __/__/____

(a) remainy cap => 3-1=2
check col 2 & row 1
* (a) 12 & 0 different
hence item 1 is selected.

* (b) remary cap => 2-2=0
col 0 & row 0 =0

Sol. vect : | 4 | 2 | 1 |

profit = 15+10+12 = 37

37 is the profit obtained
Thus verified.

* Time complexity: O(n·w)
n = no. of items
w = knapsack capacity