Lab 10 Dynamic Programming

Q1.
Consider the Knapsack problem algorithm we covered in class i.e.

$$V[i,j] = \begin{cases} \max\{V[i\text{-}1,j],\ v_i + V[i\text{-}1,j\text{-}w_i]\} & \text{if } j\text{-}w_i \geq 0 \\ V[i\text{-}1,j] & \text{if } j\text{-}w_i < 0 \end{cases}$$

Initial conditions: $V[0,j] = 0$ and $V[i,0] = 0$.

This does not list the items selected when a final optimized solution is obtained.

Use this algorithm and modify it by adding a new array Keep[i,w] which will keep track of the items selected. Write a pseudo code for this updated algorithm. Also, make sure you print the results showing items selected [Hint: use similar approach as in Print-Cut_Rod_Soln].

Ans.
(Next page)

The Complete Algorithm for the Knapsack Problem

```
KnapSack(v, w, n, W)
{
    for (w = 0 to W) V[0, w] = 0;
    for (i = 1 to n)
        for (w = 0 to W)
            if ((w[i] ≤ w) and (v[i] + V[i − 1, w − w[i]] > V[i − 1, w]))
            {
                V[i, w] = v[i] + V[i − 1, w − w[i]];
                keep[i, w] = 1;
            }
            else
            {
                V[i, w] = V[i − 1, w];
                keep[i, w] = 0;
            }
    K = W;
    for (i = n downto 1)
        if (keep[i, K] == 1)
        {
            output i;
            K = K − w[i];
        }
    return V[n, W];
}
```

Q2.
Determine the running time complexity of your pseudo code. Has Dynamic Programming approach of solving Knapsack problem changed the exponential time complexity of the original brute force solution?

Ans.
The running time is O(nW) where n is the number of items and W is the limit on the weight.
No, in this case DP has not changed the exponential running time although it has minimized the running time of the brute force method.

Q3.
Use the answer of Q2 to explain how Knapsack problem is exponential (Hint - consider both inputs and see how they [or at least one] can cause Knapsack problem to be exponential).

The running time complexity as shown above is O(nW). Inputs have two parameters in general - value and size. The number of items, n usually is not very large (although it can be theoretically). But the weight W can be very large. To represent this W we would need
size = lg b(W) where "b" is the base and "size" is the number of bits to represent "W". This means that W = b ^size. If b = 2, we have the running time as

O(n x 2^ size) which is exponential.

Another way to explain - there are TWO real inputs - items and Total Weight capacity. The items (their combination) is limited in general and is polynomial for this problem (as we can only put limited combinations e.g. we cannot possibly take all items etc). However, the weight can be any number and hence representation of it can be exponential etc.