Chapter 45: Knapsack Problem

Section 45.1: Knapsack Problem Basics

The Problem: Given a set of items where each item contains a weight and value, determine the number of each to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Pseudo code for Knapsack Problem

Given:

1. Values(array v)
2. Weights(array w)
3. Number of distinct items(n)
4. Capacity(W)

```
for j from 0 to W do:
    m[0, j] := 0
for i from 1 to n do:
    for j from 0 to W do:
        if w[i] > j then:
            m[i, j] := m[i-1, j]
        else:
            m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
```

A simple implementation of the above pseudo code using Python:

```
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]
    return K[n][W]
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(knapSack(W, wt, val, n))
```

Running the code: Save this in a file named knapSack.py

```
$ python knapSack.py
220
```

Time Complexity of the above code: `O(nW)` where n is the number of items and W is the capacity of knapsack.

Section 45.2: Solution Implemented in C#

```
public class KnapsackProblem
{
    private static int Knapsack(int w, int[] weight, int[] value, int n)
    {
        int i;
        int[,] k = new int[n + 1, w + 1];
        for (i = 0; i <= n; i++)
        {
            int b;
            for (b = 0; b <= w; b++)
            {
                if (i==0 || b==0)
                {
                    k[i, b] = 0;
                }
                else if (weight[i - 1] <= b)
                {
                    k[i, b] = Math.Max(value[i - 1] + k[i - 1, b - weight[i - 1]], k[i - 1, b]);
                }
                else
                {
                    k[i, b] = k[i - 1, b];
                }
            }
        }
        return k[n, w];
    }

    public static int Main(int nItems, int[] weights, int[] values)
    {
        int n = values.Length;
        return Knapsack(nItems, weights, values, n);
    }
}
```