Week 4

Problem 4

In order to perform this problem, we will use dynamic programming. To solve the problem by dynamic programming, we prepare a table bag[1..n,0..W] that has a row for each available treasure, and a column for each weight from 0 to "capacity". In the table, bag[i ,j] will be the maximum value of the objects that we can transport if the weight limit is j, with 0≤ j ≤ capacity, and if we only include the objects numbered from 1 to i, with 1 ≤ i ≤n. The solution to this case can be found in V[n,W].

We will fill the entries in the table using the general rule: bag[i, j] max(bag[i 1, j],bag[i 1, j w] v).

Once we obtain the values for 1 bag, we delete the treasures used and then obtain the values for the next one. We are only working with the treasure that is left (the treasure that is not in the first bag).

In this example:

First Bag

Values: `{ 60, 100, 120, 140, 210 }`
Weights: `{ 10, 20, 30, 40, 50 }`

Capacity: 50

Weight = 10; Value = 60
Weight = 20; Value = 100
Weight = 30; Value = 120
Weight = 40; Value = 140
Weight = 50; Value = 210

Weight = 20 + 10; Value = 160
Weight = 30 + 10; Value = 180
Weight = 40 + 10; Value = 200
Weight = 30 + 20; Value = 220

Content(weights): 20 and 30

Second Bag

Values: `{ 60, 140, 210 }`
Weights: `{ 10, 40, 50 }`

Capacity: 50

Weight = 10; Value = 60
Weight = 40; Value = 140
Weight = 50; Value = 210
Weight = 10 + 40; Value = 200

Content in weights: 50

Problem 7

In order to solve the problem, we have used Dynamic Programming, because generating all sub-sequences of both given sequences and find the longest matching subsequence is exponential in terms of time complexity.

In this problem we encounter 2 properties: Optimal Substructure Property and Overlapping Substructure property.

Let the input sequences be A[0..m-1] and B[0..n-1] of lengths m and n respectively. If last characters of both sequences match (or A[m-1] == B[n-1]) then LS(A[0..m-1], B[0..n-1]) = 1 + LS(A[0..m-2], B[0..n-2]). If last characters of both sequences do not match (or A[m-1] != B[n-1]) then LS(A[0..m-1], B[0..n-1]) = max(LS(X[0..m-2], Y[0..n-1]), LS(A[0..m-1], B[0..n-2])).We can see that the common subsequence problem has Optimal Substructure property, so the main problem can be solved using optimal solutions to subproblems.

While resolving this problem we saw that there were many subproblems which were solved repeatedly, which indicated the Overlapping Substructure property.

In order to obtain and print the longest common subsequence, we first constructed the L[m+1][n+1] table. As the value LS[m][n] contains the length of subsequence we created a character array lcs[lcs length + 1] with one extra slot to store \0. After that we went through the array where every cell was studied to be part of the subsequence. If the characters corresponding to LS[i][j] where the same (Or A[i-1] == B[j-1]), then it was included as part of the subsequence, if not, we compared the values of LS[i-1][j] and LS[i][j-1] and followed the greater value.

Like that we obtained the longest common subsequence of two given sequences.

In the given example we introduced the following sequences

```
String A = "01101010";  //First sequence
String B = "101001001"; //Second sequence
```

Through which we can obtain a lot of sub-sequences, however the longest common sub-sequence is the “101010”. If we changed the first sequence to “011010101” the longest common sub-sequence would be “1010101”. Both result are obtained with the implemented algorithm.