Jordan Allard
CSCI 264-01
Homework 4

HW4 Problem 3 Writeup

a) Heart of the Solution

I) WHAT: S[j, u, v] = the max total cost of two subsets of items chosen from the first j items, where the weight of the first subset <= u and the weight of the second subset is <= v.

II) HOW:

```
S[j, u, v] = 0 if j = 0, u = 0, v = 0
S[j, u, v] = max(
      S[j - 1, u, v],
      cj + S[j - 1, u - wj, v] if u - wj >= 0, else
      cj + S[j - 1, u, v - wj] if v - wj >= 0)
```

III) WHERE: S[n, W1, W2]

b) Pseudocode

```
Given a set of items with cost ci and weight wi
Given two max weights of W1 and W2
Let S be a 3D array with dimensions n x W1 x W2
For u = 0 to W1:
   For v = 0 to W2: S[0, u, v] = 0
For j = 0 to n: s[j, 0, 0] = 0
For j = 1 to n:
   For u = 1 to w1:
      For v = 1 to w2:
         S[j, u, v] = S[j - 1, u, v]
            If (u - wj >= 0) and (cj + S[j - 1, u - wj, v] > S[j, u, v]):
                 S[j, u, v] = S[j - 1, u - wj, v] + cj
            Else if (v - wj >= 0) and (cj + Spj - 1, u, v - wj]
             > S[j, u, v]):
                 S[j, u, v] = S[j - 1, u, v - wj] + cj
Return S[n, W1, W2]

Reconstructing the solution:
Let b1 and b2 be arrays of items contained in bag 1 and bag 2 respectively
u = W1
v = W2
For i = n to 0:
   If S[i, u, v] != S[i - 1, u, v]:
         If (u - wj >= 0) and (S[i, u, v] - cj == S[i - 1, u - wj, v]):
             Add i to b1
             u -= wj
         Else if (v - wj >= 0) and (S[i, u, v] - cj == S[i - 1, u, v - wj]):
             Add i to b2
             v -= wj
Print the items in b1
Print the items in b2
```

c) Proof of Correctness (Explanation of HOW)

The algorithm is guaranteed to find the maximum cost possible for the given weights because it will check every item and always takes the maximum at each iteration. At each iteration, if the cost of the new item will improve the maximum, the algorithm will try to take it. First it checks if the item can fit in the first knapsack, and if not then it will try to fit it into the second knapsack. If taking the item will reduce space without increasing the cost, or if the item doesn't fit in either bag, it will pass over the item. In this way, the algorithm will always

find the maximum cost of items that will satisfy the weight limit of both bags.

d) Running Time Estimate
O(n * W1 * W2)

e) Running Time Estimate Reasoning
The running time estimate for this algorithm can be broken down as follows:
Read in initial input --> O(n)
Initialize the array with zeroes --> O(W1 * W2) + O(n)
> It technically takes O(n * W1 * W2) time to allocate memory for the array
Fill the solution array --> O(n * W1 * W2)
> The nested for loops run n times, W1 times, and W2 times respectively
Reconstruct the solution --> O(n)
> The items in b1 and b2 cannot exceed n, so finding and printing items is technically O(n) + O(n) + O(n) = O(n)
Thus, the total running time is O(n * W1 * W2).