CS 1510: Homework 10

John Hofrichter
jmh162@pitt.edu

Kyra F. Lee
kfl5@pitt.edu

Zach Sadler
zps6@pitt.edu

September 19, 2013

Dynamic Programming Problem 11a

Explanation

To backtrace our solution out of the Value table, we start at the bottom-right most corner of the value table, at Value[N, L]. Then we compare the value at that position to the value one row above and in the same column as that position. If they are equal then we move upwards again. Continue this process until the value at position Value[i, j] is not equal to the position at Value[i-1, j].
Once we have the case where the value at a position is not equal to the value above that position, we look to the row above our position and go left equal to the weight of the object at row i-1.
The reasoning is that we added that item to our solution set when we were creating the Value table, so when constructing Value[i, j], we referenced one row higher $i - 1$ which represents one fewer items in the array, as well as $j - W(i - 1)$ which represents the weight of the previous object. We access this position in the table to get the optimal value at weight i-1, and add object $i - 1$'s value to it.
Thus at this point we should add object i-1 to our solution set.
We repeat this process of going up if we get the same value, up to the left if we get a different value, until eventually we reach the top left corner and have our complete solution set.

Pseudo-code follows:

Algorithm

```
s = L
for k = N down to 1 do
  if Value[k-1, s] != val[k, s] then
    solutions.add(objects_{k-1})
    s = s - objects_{k-1}.weight
  end if
end for
```

Dynamic Programming Problem 11b

Explanation

Using the solution given in the class notes for the Knapsack problem, we can see it is clear that we have 2 nested for loops, one with n iterations and one with L iterations, and $O(1)$ operations inside the inner for loop. Thus it is clear that we have $O(nL)$ time required.
To use $O(L)$ memory we'll actually use $2L$ memory; a two-row, L-column matrix. To do this we have one row representing the previous values we've calculated up to this point (or $-\infty$ in the first step), and the other row representing the current values for this iteration. For each iteration, we take our current object and insert its value into it's weight index in the current values row. Then we go through each position of our current object array and replace its value with the maximum of it's current value, the value at the previous row in that index, and the value of the index in the previous row which is the current object's weight to the left of our current index.
Since we only need to reference the previous row when constructing the current row in our column, we can effectively only keep track of two rows at a time in our algorithm. Thus we can use $O(2L) = O(L)$ memory to solve the Knapsack problem.

Dynamic Programming Problem 12a

We also go through each non-zero value in the previous values row, and add that value plus the current object's value to the current row, then replace the value at index current object weight + previous value weight with the

maximum of . After every weight index has been calculated for the current row, we use that as our previous values row for the next iteration.