CS 325 - HW 3 – Selected solutions

Problem 1: *(3 points)* **Rod Cutting**: (from the text CLRS) 15.1-2 **– Many possible solutions**

Here is a counterexample for the "greedy" strategy:

length i	1	2	3	4
price p_i	1	20	33	36
p_i/i	1	10	11	1

Let the given rod length be 4. According to a greedy strategy, we first cut out a rod of length 3 for a price of 33, which leaves us with a rod of length 1 of price 1. The total price for the rod is 34. The optimal way is to cut it into two rods of length 2 each fetching us 40 dollars.

Problem 2: **(3 points)** Modified Rod Cutting: (from the text CLRS) 15.1-3

```
MODIFIED-CUT-ROD(p, n, c)
  let r[0..n] be a new array
  r[0] = 0
  for j = 1 to n
      q = p[j]
      for i = 1 to j - 1
          q = max(q, p[i] + r[j - i] - c)
      r[j] = q
  return r[n]
```

The major modification required is in the body of the inner **for** loop, which now reads $q = \max(q, p[i] + r[j-i] - c)$. This change reflects the fixed cost of making the cut, which is deducted from the revenue. We also have to handle the case in which we make no cuts (when i equals j); the total revenue in this case is simply $p[j]$. Thus, we modify the inner **for** loop to run from i to $j-1$ instead of to j. The assignment $q = p[j]$ takes care of the case of no cuts. If we did not make these modifications, then even in the case of no cuts, we would be deducting c from the total revenue.

Problem 3: ***(6 points)*** **Making Change**: Given coins of denominations (value) $1 = v_1 < v_2 < \ldots < v_n$, we wish to make change for an amount A using as few coins as possible. Assume that v_i's and A are integers. Since $v_1 = 1$ there will always be a solution.

a) Describe **(1 pt)** and give pseudocode for a dynamic programming algorithm to find the minimum number of coins to make change for A.

Sample pseudocode example **(3 pts)**

coins[k] is the number of coins used to make change for k cents
sol[k] is the index of last denomination V[sol[k]] used to obtain change for k cents

Formula

coins[i] = inf if i < 0
coins[0] = 0
coins[1]= 1
coins[j] = $\min_{1 \le i \le n}\{1 + coins[j - v_i]\}$**, sol[j] = i**

```
minCoins(A, V[], n)
        coins[0]=0; coins[1] = 1
        for j = 2 to A do {
                min = inf
                for i = 1 to n do {
                        if ( j >= V[i] )
                           if (coins[j-V[i]] < min )
                           {
                                min = coins[j-V[i]]
                                index = i
                            }
                }
                coins[j] = min + 1
                sol[j] = index
        }
        return coins[A], sol[A]
```

To reconstruct the series of coins used to obtain change for A with minimum number of coins. Call MakeChange(sol, V, A).

```
MakeChange(sol[], V[],C[], m)
        While m > 0 {
                C[sol[m]] = C[sol[m]] + 1
                Print V[sol[m]]
                m = m – V[sol[n]]
        }
```

b) What is the theoretical running time of your algorithm? **(2 pts)**

The running time of MakeChange is O(A)
The running time of minCoins is Θ(nA) since the outer loop is j = 2 ..A and inner loop is i = 1 ..n.
Overall Θ(nA)

Problem 4:

a) The Shopping Spree problem is modified version of the Knapsack problem. First we find the family member who can carry the most weight and call that weight M_{max}. We then use the DP algorithm to solve the knapsack problem with n items of price P_i, weight W_i and a knapsack with capacity of M_{max}. The running time of the DP knapsack algorithm is $\Theta(n\ M_{max})$. The maximum total price of goods that family member k can carry is in cell (n, M_k) where M_k is the maximum weight that member k can carry. The maximum value for each family member can be obtained from the table in constant time. The time to backtrack in the DP table to determine the items a single family member takes is O(n) and for all family member this takes O(nF).

b) The overall running time is $\Theta(n\ M_{max})$ + O(nF) or $O(n\ (M_{max} + F))$ or $\Theta(nM_{max})$ if $M_{max} >> F$