Casey Levy – CS 325 – HW 3

Problem 1

- For *i* = 1, 2, 3, 4…..
- The "greedy" strategy does not always determine an optimal way because a rod of length 4 (*i* = 4) can be our counterexample

i	p_i	p_i / i
1	1	1
2	20	10
3	33	11
4	36	9

Problem 2

- def modified(p, n, c)
  Make a new array A[0..n]
  A[0] = 0
  for i = 1 to n:
  j = p[i]
  for x = 1 to i – 1:
  j = max(j, p[x] + A[i – x] – c)
  A[i] = j
  Return A[n]

- The cost of a cut, *c,* would still be deducted from total revenue even if a cut was not made if these modifications are not made
- We add "*j = max(j, p[x] + A[i – x] – c*)" to represent the cost of the cut
- Also, the case of no cuts made is handled when we include "*j = p[i]*"

Problem 3

a)

- Some work inspired by Coin Change Problem Using Dynamic Programming (codesdope.com)

```
def change(v, n, c)
    C[n + 1]
    C[0] = 0


    for i in 1 to n:
        min = INF
```

```
for k in 1 to c:
    if i ≥ v[i]
        if min < (1 + C[i – v[k]])
            return min
        else
            min = (1 + C[i – v[k]])
            return min

C[i] = min

return C[n]
```

- We first must make a new array, starting at n+1, to store the minimum number of coins needed for each value
- We then set a min to INF so that all other values are less than min in the beginning, so that we can store the minimum number of coins for each value
- For each k, we must choose the minimum where d_i will change from d_1 to d_k
- We continue and then hit our conditional statement where we may or may not change the value of min
- Array C's value is changed and then returned

b) Running time is **O(n * A)**

Problem 4

a) The problem given is similar to the 0/1 Knapsack problem and must be altered to fit the entire family. Using bottom up iteration, we compare the weight value to the total amount allowed and use this to fill in our dynamic programming table. Backtracking loop sorts the order of items chosen to tell us the actual number and their values.

- shoppingSpree(W, weight, vals, n)
    - create matrix and loop through ranges of weight
        - if item doesn't fit, give 0
        - if it does, take max of previous item in same column
- Continue through until matrix is filled
- Run backtracking to get items/values then write to output file

b) **O(nk)**