Justin Phillips

spring 19 hw3

15.1-2 Show by the means of a counterexample, that the following 'greedy" strategy does not always determine an optimal way to cut rods. define the density of a rod of length to be p_i/i. that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where 1 <= I <= n, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n-i.

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	17	17	20
density	1	2.5	2.6	2.25	2	2.83	2.43	2.5

using the greedy strategy we start at the to down and in this case it would cut at 7 and have a piece 7 and 1 which total price is 18 with a density of 3.43 which isn't the best choice. where if we cut at 6 we would get a piece of 6 and 2 which price = 22 with a density of 5.33

15.1-3 Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic programming algorithm to solve this modified problem.

```
modCutRod(price, n, cost){
        array[ 0…n];
        array[0];
        for(int I =1; I < n; i++){
                int q = price[i];
                for(int j =1; j<=I;j++){
                q=max(q,price[j]+array[i-j]-cost);
                }
        array[i]=q;
        }
        return array[n];
}
```

3. Problem 3: (6points)Making Change:

Given coins of denominations (value) $1 = v_1 < v_2 < ... < v_n$, 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. Formally, an algorithm for this problem should take as input an array V where V[i] is the value of the coin of the i_{th} denomination and a value A which is the amount of change we are asked to make The algorithm should return an array C where C[i] is the number of coins of value V[i] to return as change and m the minimum number of coins it took. You must return exact change so

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

-using a dynamic programming approach and a bottom up approach, we iterate through each coin and use a min coin table to determine the most optimized solution for each total

```
pseudo
for value 0…A
    set current coin value to count
    set 1 = newcoin
        for each value && if value <=cents
            if minCoin value +1 is < coinCount
                coinCount = (amount – (value+1)
                newcoin = value
        put coinCount in minCOin table
        put newcoin in coinsUsed table
    return minCoins[A]
```

b) What is the theoretical running time of your algorithm?

The first loop runs A times at O(n). The 2nd loop runs inside the 1st and runs m times and m is the number of different coins and since we got nested loops we multiply them and the algorithm is pseudo polynomial O(n*m) is our run time theoretically.

Problem 4:Shopping Spree: (18 points)
Acme Super Store is having a contest to give away shopping sprees to lucky families. If a family wins a shopping spree each person in the family can take any items in the store that he or she can carry
out , however each person can only take one of each type of item. For example, one family member can take one television, one watch and one toaster, while another family member can take one television,

one camera and one pair of shoes. Each item has a price (in dollars) and a weight (in pounds) and each person in the family has a limit in the total weight they can carry. Two people cannot work together
to carry an item . Your job is to help the families select items for each person to carry to maximize the total price of all items the family takes.Write an algorithm to determine the maximum total price of items for each
family and the items that each family member should select.
Submit to Canvas
a) A verbal description and give pseudo-code for your algorithm. Try to create an algorithm that is
efficient in both time and storage requirements.

each member must carry their own items and cant carry more then one item of each. This code can be broken down into the knapsack problem where F is the number of the members. what needs to be done is to maximize the benefit of the capacity of each member using the discrete knapsack formula

pseudo code:

```
function knapsack(items selected(n), capacity(w), items, array trace)
    create 2d array[0…items],[0…capacity]
    initiate solution table with a for (I to n)
        fill 1st column with I and put 0 in the second [i][0]
    build table from the bottom up
        for I to n
            for j to w
            if ith is heavier then capacity
                use previous solution
            else
            take the I item or whatever item has the most benefit
initiate trace array
initiate an index of variables I and j
build the trace array using a while
    set to I if the current solution is different then optimal solution
    mark before going to the subproblem'
    then decrement I to backup a row
    return modified array
```

b) What is the theoretical running time of your algorithm for one test case given N items, a family of size
F, and family members who can carry at most M_i pounds for $1 \leq i \leq F$.

worst case complexity is O(FNW)

c) Implement your algorithm by writing a program named “shopping” (in C, C++or Python) that compiles
and runs on the OSU engineering servers. The program should satisfy the specifications below.