George Lenz
HW3
7/15/2018

Problem 1:

This does not necessarily work because it may not give back the maximum answer. For example if you had a rod of length 7 and you had cuts with values such their values are 1= 2, 2=6, 3=8, and 4=11 5=14 6 = 15 and 7 = 16 The highest pi / i would be of length 2 with a density of 3. so the cuts would be 2,2,2,1which gives 19 but with dynamic programming we can see that a rod of length 7 cut up into pieces of 5 and 2 would give a total value of 20 which is greater.

Problem 2:

Pseudocode1:

```
Bottom-Up-Cut-Rod(p, n)
  let r[0…n] be a new array
  r[0] = 0
  for I = 1 to n
    q = -∞
    for I = 1 to j
      q = max(q, (p[i] – cost) + r[j-i])
    r[j] = q
  return r[n]
```

Pseudocode 2:

```
opt[0] = 0
sol[0] = []
for k = 1 to price.length
 opt[k] = 0
 sol[k] = null
 for i = 1 to k
 if opt[k] < opt[k-i] + (price[i] - cost)
 opt[k] = opt[k-i] + (price[i] - cost)
 sol[k] = sol[k-i] + [i]
```

Problem 3:

a) This is similar to the knapsack or rod cutting algorithms in which you have a maximum weight (or in this case time) and you have a number of items (problems) each with an amount of weight and a benefit, and you want to maximize the benefit (or points) within the amount of total time.

b)

```
for t = 0 to T
   P[0, time] = 0
for  q = 0 to Q
  P[time, 0] = 0
for t = 0 to T
   if time <= T
      if p_q + P[q-1, t-t_q] > P[q-1, t]
         P[q, t] = p_q + P[q-1, t-t_q]
      else
         P[q, t] = P[q-1, t]
else P[q, t] = P[q-1, t]
```

c) the running time of this algorithm is O[qt] which is the time to fill the table. It is pseudo-polynomial.

d) No, if the professor gave partial credit, it is no longer similar to an 0,1 knapsack problem and rather becomes more similar to the fractional knapsack algorithm.

Problem 4:

a)

```
change(V, A)
   minimumChagne[A+1]
   coinAmountList[A+1]
   NumberOfEachCoin[Vsize]

   minimumChange[0] and coinAmountList[0]= 0
   set minimunChange[1…A] = infinity
    for i…V.size
       for j…A
          if j >=V[i]
             min(minimumChange[i], 1+ minimumChange[j-V[i]]
             coins[j] = i
     a = A

     while a > 0
        NumberOfEachCoin[coinAmountList[a]] + 1
        a = a- V[coinAmountList[a]]
```

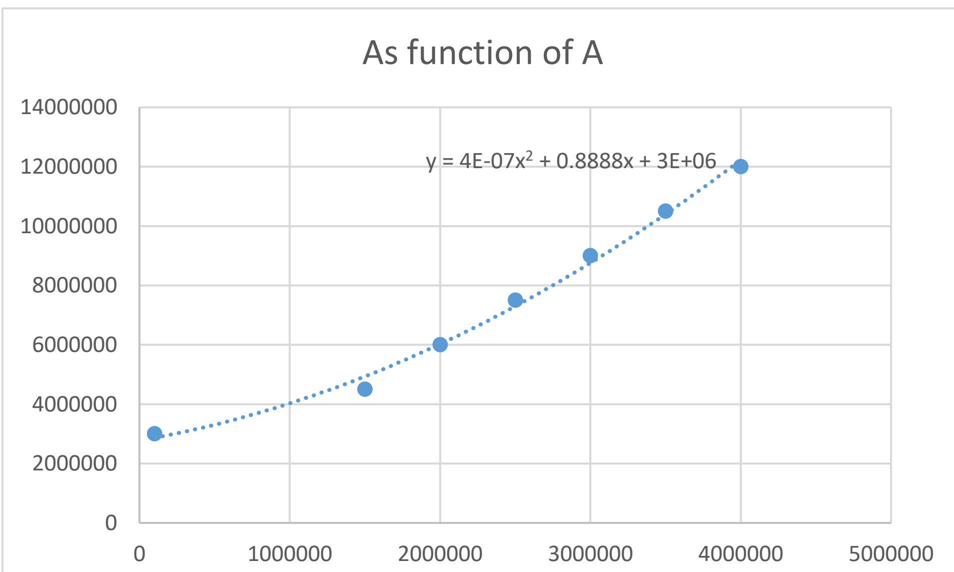
b)

the theoretical running time is [mn] where m is the size of V and n is the amount of change to be made.

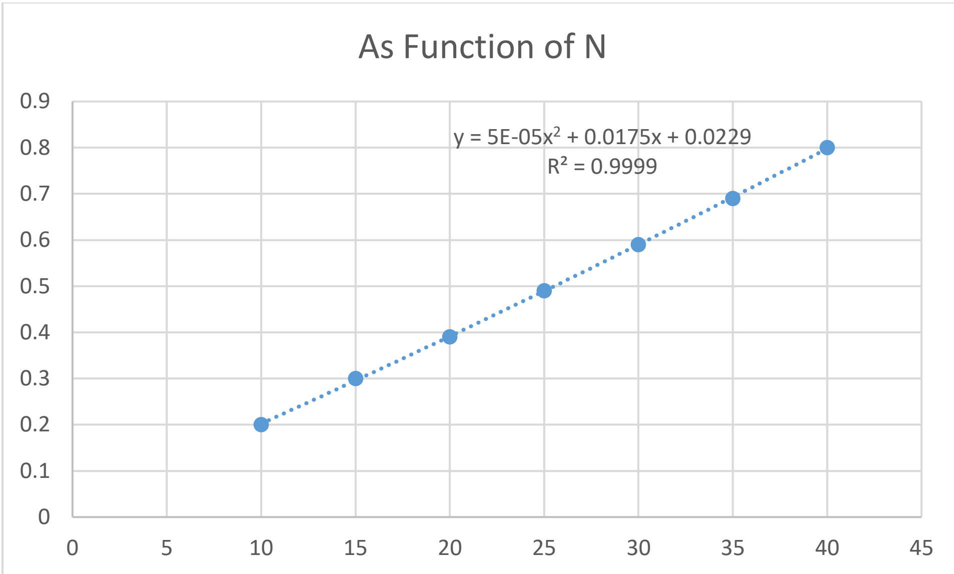
Problem 6:

a) to collect the data I modified the program to ask for an amount size, the amount of different coins, and the denominations of each coin. I then put in each denomination manually from 1-N, starting from a denomination of coin 1 with value 1, up to coin N with value N.

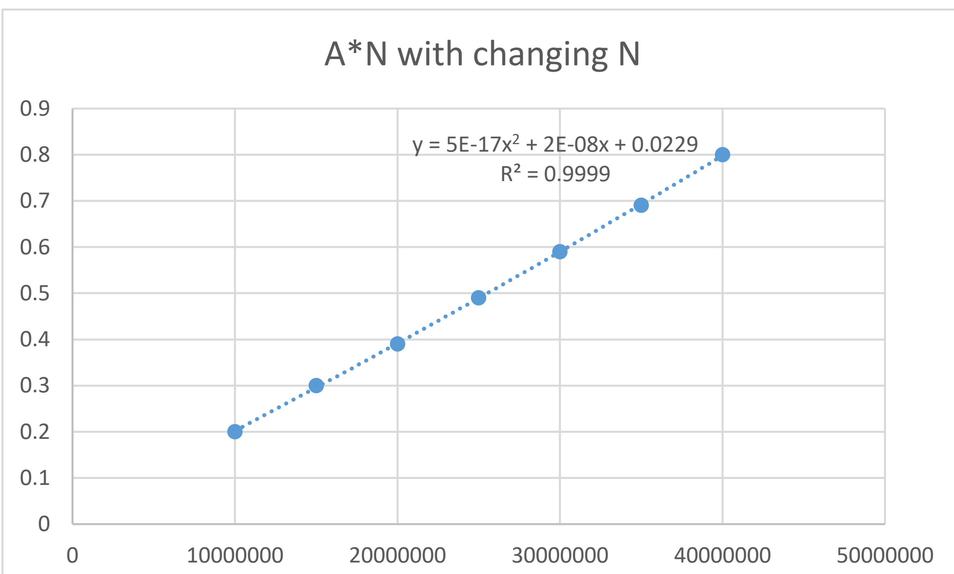
A	N	Time
100000	3	0.06
1500000	3	0.1
2000000	3	0.14
2500000	3	0.17
3000000	3	0.2
3500000	3	0.24
4000000	3	0.28

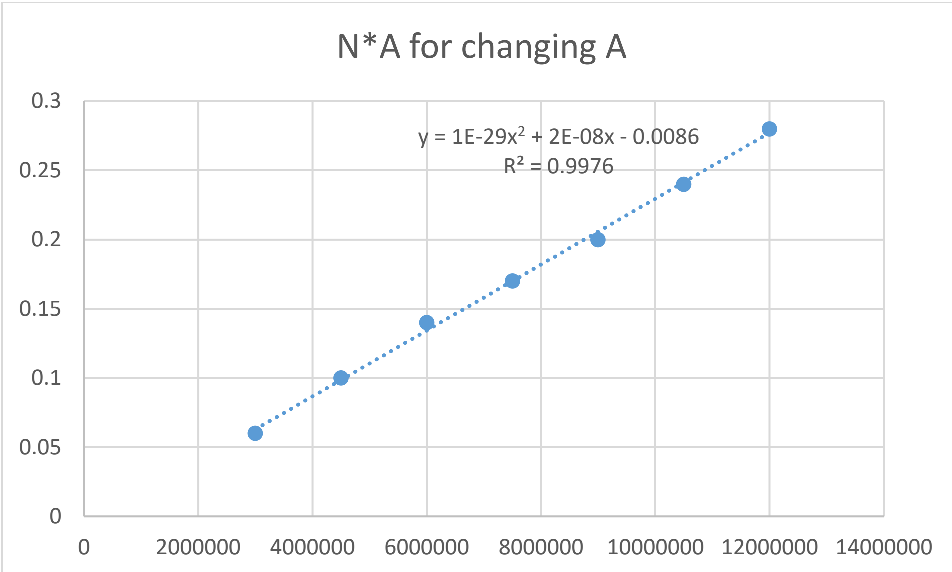


A	N	Time
1000000	10	0.2
1000000	15	0.3
1000000	20	0.39
1000000	25	0.49
1000000	30	0.59
1000000	35	0.69
1000000	40	0.8



As Function of A*N for both





b) These results fit the data as polynomial fit best for all the graphs which was as expected. The only catch was that on the last graph of A*N with a changing A, a linear graph also fit equally as well.