CS 325 Spring 2018 – HW 3

Sheng Bian

Problem 1:

The following is a counterexample for greedy strategy.

Length	1	2	3	4
Price	1	30	48	56
Density	1	15	16	14

If we use greedy strategy, we will first cut the rod of length 3 and then we will have another rod of length 1. The total price will be 49. If we cut the rod of length 2, we will get two rods of length 2. The price for two rods of length 2 is 60. Therefore, for this counterexample, the greedy strategy doesn't determine the optimal way.

Problem 2:

```
modified_cut_rod(p, n, c){
        array[0..n]
        array[0] = 0
        for(i=1, i <= n, i++){
                q = p[i]
                for(j = 1, j <= i-1, j++){
                        q = max(q, p[j] + array[i-j] – c)
                }
                r[i] = q
        }
        return array[n]
}
```

Problem 3:

(a) Verbally describe a DP algorithm to solve this problem.

This is a 0-1 Knapsack Algorithm problem. The solution is to consider all subsets of problems and calculate the total time and points of all subsets. Consider the only subsets whose total time is smaller than T. From all such subsets, pick the maximum points subset. To consider best subset of S_k that has the total points t, either contains problem k or not.

First case: $t_k>t$. Item k can't be part of the solution, since if it was, the total weight would be > t. So we select the "optimal" using items 1,.., k-1.

Second case: $t_k \leqslant t$. Then the item k can be in the solution, and we choose the case with greater value.

It means, that the best subset of S_k that has total time t is one of the two:

Do not contain problem k: the best subset of S_{k-1} that has total time t, or

Contain problem k: the best subset of S_{k-1} that has total time $t-t_k$ plus the problem k with point p_k

At last, print out all the selected problems.

(b) Give pseudo code with enough detail to obtain the running time, include the formula used to fill the table or array.

```
for t = 0 to T
        P[0, t] = 0
for i = 0 to n
        P[i,0] = 0
        for t = 0 to T
                if t_i <= t
                        print i     // this is the question that should be selected to answer
                        if p_i + P[i-1,t-t_i] > P[i-1,t]
                                P[i,t] = p_i + P[i-1,t- t_i]
                        else
                                P[i,t] = P[i-1,t]
                else P[i,t] = P[i-1,t]
```

(c) What is the running time of your algorithm? Explain.

The running time is O(nT) pseudo-polynomial. For "for t = 0 to T" it takes time of O(T). For "for i = 0 to n" and "for t = 0 to T", it takes time of O(nT). Therefore, the running is O(nT).

(d) Would Benny use this algorithm if the professor gave partial credit for partially completed questions on the exam? Discuss.

No, he can’t. The reason is the original question is a “0-1 knapsack problem”. If the professor gave partial credit for partially completed questions, this question becomes “Fractional knapsack problem”. There are different algorithm solutions to these two different questions.

Problem 4:

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

```
function makeChange(denominations, amount, minCoins, coinsUsed):
    for cent in range(amount + 1):
        coinCount = cent
        newCoin = 1
        for i in [c for c in denominations if c <= cent]:
            if minCoins[cent - i] + 1 < coinCount:
                coinCount = minCoins[cent - i] + 1
                newCoin = i
        minCoins[cent] = coinCount
        coinsUsed[cent] = newCoin
    return minCoins[amount]
```

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

The running time for “for cent in range(amount + 1)” is A,

the running time for “for i in [c for c in denominations if c <= cent]” is n,

therefore, the theoretical running time of my algorithm is O(nA)

Problem 6:

a)

The following code is used to collect running time

```
import time
import random

def makeChange(denominations, amount, minCoins, coinsUsed):
    for cent in range(amount + 1):
        coinCount = cent
        newCoin = 1
        for i in [c for c in denominations if c <= cent]:
            if minCoins[cent - i] + 1 < coinCount:
                coinCount = minCoins[cent - i] + 1
                newCoin = i
        minCoins[cent] = coinCount
        coinsUsed[cent] = newCoin
    return minCoins[amount]

def printRunningTimeA(amount):
    deno = [1,2,5]
    coinsUsed = [0]*(amount+1)
    coinCount = [0]*(amount+1)
    startTime = time.time()
    makeChange(deno, amount, coinCount, coinsUsed)
    print("The running time  as a function of A for amount = %s is %.5f seconds" %
(amount, (time.time() - startTime)))

printRunningTimeA(100000)
printRunningTimeA(200000)
printRunningTimeA(300000)
printRunningTimeA(400000)
printRunningTimeA(500000)
printRunningTimeA(600000)
printRunningTimeA(700000)
printRunningTimeA(800000)

def printRunningTimeN(n):
    deno = random.sample(range(0, 20), n)
    coinsUsed = [0]*(800000+1)
    coinCount = [0]*(800000+1)
    startTime = time.time()
    makeChange(deno, 800000, coinCount, coinsUsed)
    print("The running time  as a function of n for denomination size = %s is %.5f
seconds" % (n, (time.time() - startTime)))

printRunningTimeN(3)
printRunningTimeN(4)
printRunningTimeN(5)
printRunningTimeN(6)
printRunningTimeN(7)
printRunningTimeN(8)
printRunningTimeN(9)
printRunningTimeN(10)

def printRunningTimeNAndA(n, amount):
    deno = random.sample(range(0, 20), n)
    coinsUsed = [0]*(amount+1)
    coinCount = [0]*(amount+1)
    startTime = time.time()
```

```
    makeChange(deno, amount, coinCount, coinsUsed)
    print("The running time  as a function of nA for denomination size = %s amount
= %s nA = %s is %.5f seconds" % (n, amount, n*amount, (time.time() - startTime)))

printRunningTimeNAndA(3, 100000)
printRunningTimeNAndA(4, 200000)
printRunningTimeNAndA(5, 300000)
printRunningTimeNAndA(6, 400000)
printRunningTimeNAndA(7, 500000)
printRunningTimeNAndA(8, 600000)
printRunningTimeNAndA(9, 700000)
printRunningTimeNAndA(10, 800000)
```

The running time as a function of A for amount = 100000 is 0.12608 seconds

The running time as a function of A for amount = 200000 is 0.24817 seconds

The running time as a function of A for amount = 300000 is 0.36984 seconds

The running time as a function of A for amount = 400000 is 0.52106 seconds

The running time as a function of A for amount = 500000 is 0.61639 seconds

The running time as a function of A for amount = 600000 is 0.74150 seconds

The running time as a function of A for amount = 700000 is 0.86157 seconds

The running time as a function of A for amount = 800000 is 0.98466 seconds

The running time as a function of n for denomination size = 3 is 0.95162 seconds

The running time as a function of n for denomination size = 4 is 1.12375 seconds

The running time as a function of n for denomination size = 5 is 1.21381 seconds

The running time as a function of n for denomination size = 6 is 1.41694 seconds

The running time as a function of n for denomination size = 7 is 1.53403 seconds

The running time as a function of n for denomination size = 8 is 1.71915 seconds

The running time as a function of n for denomination size = 9 is 1.80321 seconds

The running time as a function of n for denomination size = 10 is 1.73316 seconds

The running time as a function of nA for denomination size = 3 amount = 100000 nA = 300000 is 0.11408 seconds

The running time as a function of nA for denomination size = 4 amount = 200000 nA = 800000 is 0.28119 seconds

The running time as a function of nA for denomination size = 5 amount = 300000 nA = 1500000 is 0.47832 seconds

The running time as a function of nA for denomination size = 6 amount = 400000 nA = 2400000 is 0.63142 seconds

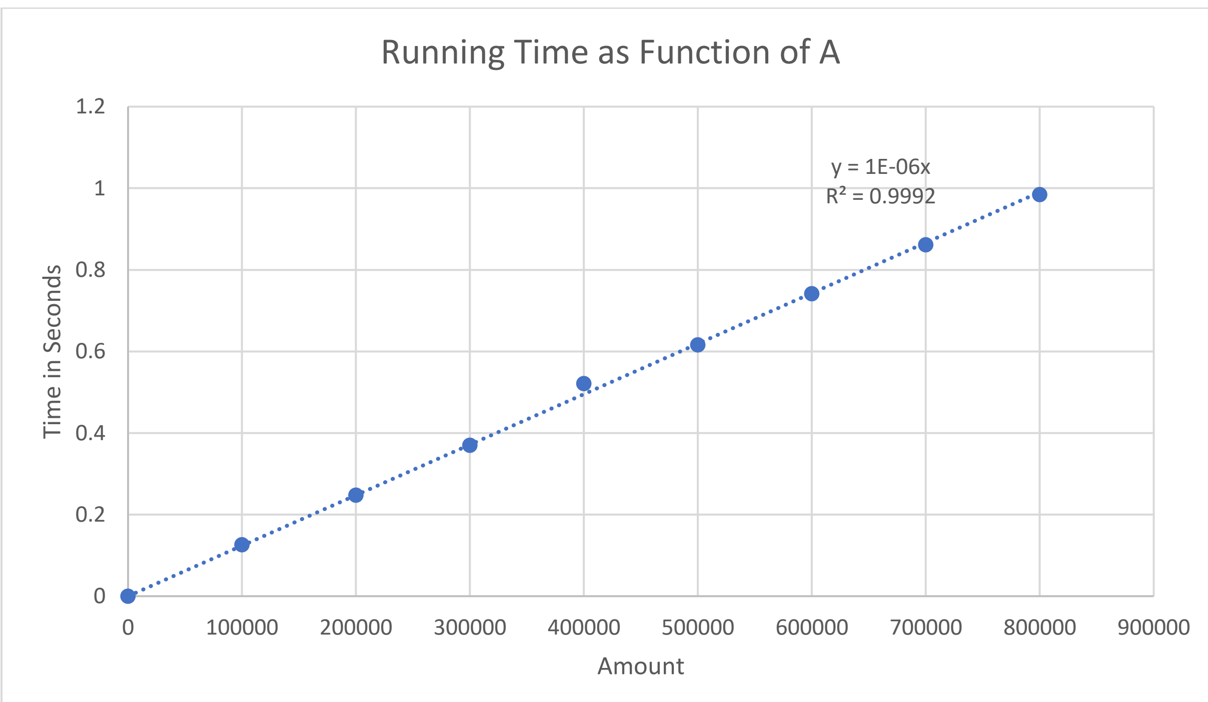
The running time as a function of nA for denomination size = 7 amount = 500000 nA = 3500000 is 0.95564 seconds

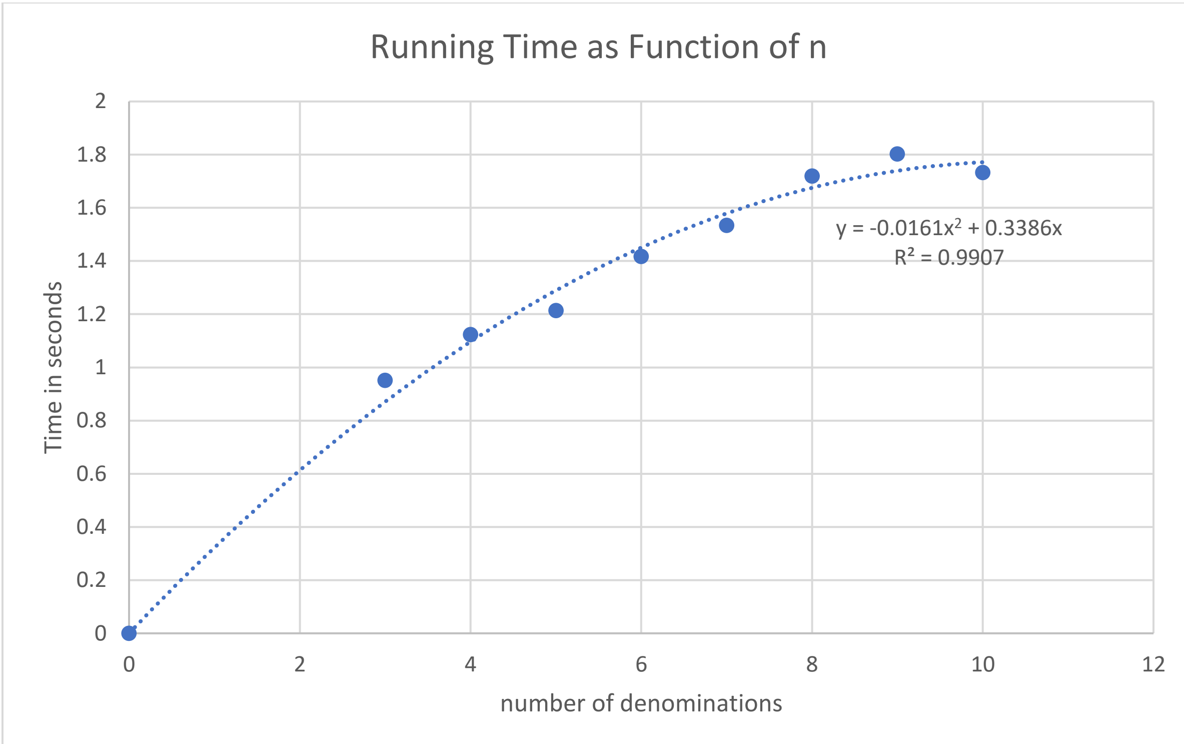
The running time as a function of nA for denomination size = 8 amount = 600000 nA = 4800000 is 1.11474 seconds

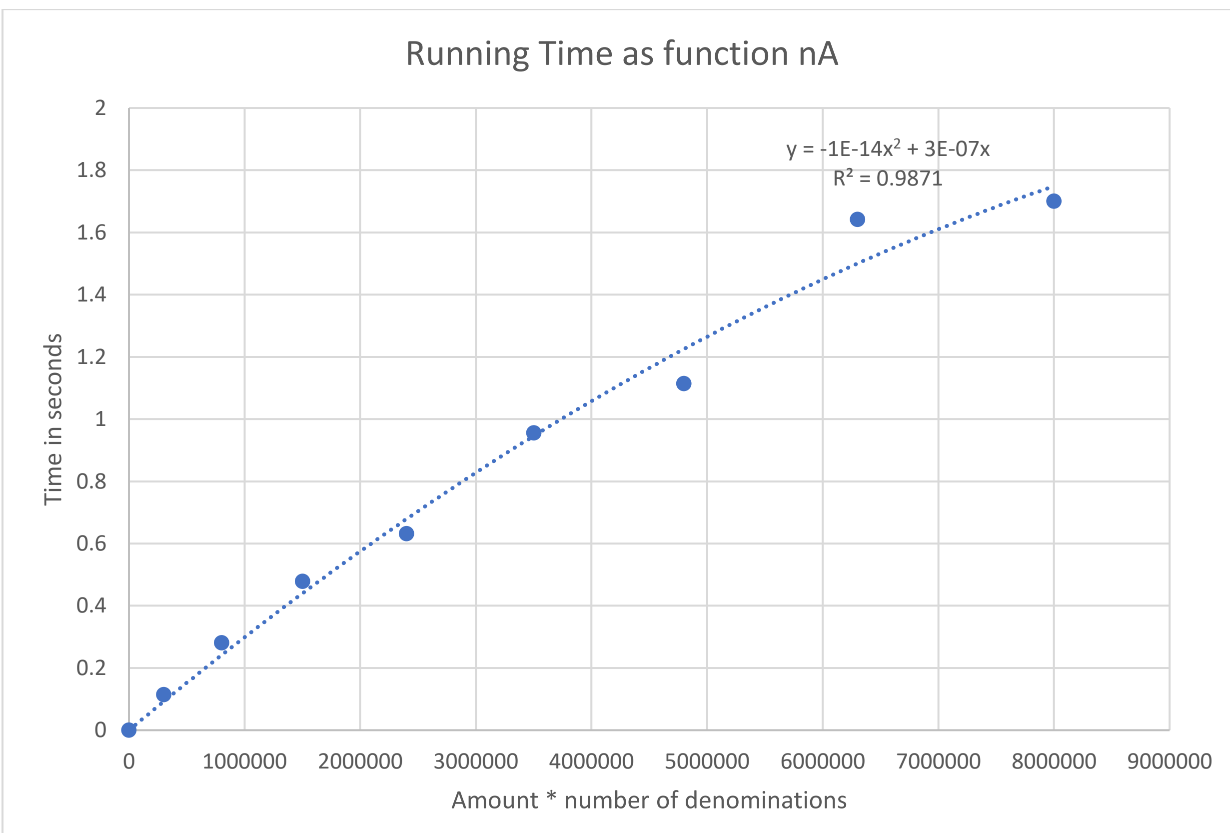
The running time as a function of nA for denomination size = 9 amount = 700000 nA = 6300000 is 1.64210 seconds

The running time as a function of nA for denomination size = 10 amount = 800000 nA = 8000000 is 1.70114 seconds

b)




Running Time as Function of n
y = -0.0161x² + 0.3386x
R² = 0.9907
Time in seconds
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
number of denominations


Running Time as function nA
y = -1E-14x² + 3E-07x
R² = 0.9871
Time in seconds
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
8000000
9000000
Amount * number of denominations

The equation of running time of function A is $y = 1E\text{-}06x$,

The equation of running time of function n is $y = -0.0161x^2 + 0.3386x$

The equation of running time of function nA is $y = -1E\text{-}14x^2 + 3E\text{-}07x$

My theoretical running time is O(nA), the closest equation of running time is graph of function of nA.