Student: Joaquin Saldana
Course: CS325 – Summer 2017
Homework 3 Submission

1. Show, by means of counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods.

   Below is a counterexample for the "greedy" strategy:

   Assume the following values for price and density …

length i	1	2	3	4
price (pi)	1	20	33	35
density (pi/i)	1	10	11	8.75

   if the rod length is 4, per the greedy strategy, we first cut out a rod of length 3 for the price of 33, which only leaves us with a rod of length 1 for the price of 1 and the total price of the length 4 rod to $34.

   While, if we had cut the rod into 2 pieces, the total price it would have fetched is $40. Proving this greedy strategy does not always gives the optimal solution.

2. Was provided a list of n integers, (v1, v2, …., vn), that product sum is the largest sum that can be formed by multiplying adjacent elements in the list. Each element can be matched with at most one of its neighbors.

   a. Compute the product sum of 1, 4, 3, 2, 3, 4, 2

      The largest product sum is 1 + (4 x 3) + 2 + (3 x 4) + 2 = 29

   b. Give the DP optimization formula OPT[j] for computing the product-sum of the first j elements.

$$OPT[j]\begin{cases}\max\{OPT[j-1]+v_j, OPT[j-2]+v_j * v_{j-1} & if \geq 2\\ v_1 \quad if\ j=1\\ 0 \quad if j=0\end{cases}$$

   c. What would be the asymptotic running time of a dynamic programming algorithm implemented using the formula in part b.

      In quickly brainstorming pseudocode for the algorithm, I can immediately think that at the very least 2 loops will be necessary to iterate through the array of n elements. As a result I believe the asymptotic run time is polynomial an is $O(n^2)$

3. Making Change:

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

```
makeChange(n, d, k)
        C[0] = 0
        for j = 1 to n do
                C[j] = ∞
                For i = 1 to k do
                        if j ≥ d_i and 1 + C[j − d_i] then
                        C[j] = 1 + C[j - d_i]
                        denom[j] = d_i

        return c
```

   b. Demonstrate your algorithm "by hand" when A = 11, v1 = 1, v2, = 3, and v3 = 4

   First we create an array (or in this case a table) to index the coins at our disposal

	coin index	value
coins	0	1
	1	3
	2	4

   Next we create 2 arrays, one of which is indexed up to the total amount (11) in this case and is initialized with the value of infinity

Array to keep the minimum number of coins											
0	1	2	3	4	5	6	7	8	9	10	11
0	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

   And the second array is track the index of the value that was used to find the minimum. Here we use the index of the coins in the table above. It's initialized to the value of -1.

0	1	2	3	4	5	6	7	8	9	10	11
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

   Now we a start to fill in the array by iterating through each coin in the table starting w/ index 0 and up to index 2 which has the value 4. If the value fits into the index then we store the number of coins by using the optimization formula

$$T[i] = \min\big(T[i], \quad T\big[i - coins[j]\big]\big)\ if\ i \geq coins[j]$$

So after the first iteration of our loop, w/ the coin value of 1 we have the following array:

Array to keep the minimum number of coins											
0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	3	4	5	6	7	8	9	10	11

And the array that holds the index value of the coin at

0	1	2	3	4	5	6	7	8	9	10	11
-1	0	0	0	0	0	0	0	0	0	0	0

After the second iteration, which in this case we are now evaluating the coin at index 1 which has the value of 3, we have the following array values:

Array to keep the minimum number of coins											
0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	1	2	3	2	3	4	3	4	5

Array that holds the index value of the coin:

0	1	2	3	4	5	6	7	8	9	10	11
-1	0	0	1	0	1	1	1	1	1	1	1

And finally after the third iteration we have the following arrays:

Array to keep the minimum number of coins											
0	1	2	3	4	5	6	7	8	9	10	11
0	1	2	1	1	2	2	2	2	3	3	3

array that holds the index value of the coin:

0	1	2	3	4	5	6	7	8	9	10	11
-1	0	0	1	2	2	1	2	2	2	2	2

Now to we know that in order to get to 11 coins we need the following coins

Coin[2] = 4

11 – 4 = C[7] = 2 = Coin[2] = 4

7 – 4 = C[3] = 1 = Coin[1] = 3

3 – 3 = 0 , which means we've arrived at the end of our algorithm.

So to get the amount of 11, the minimum we need is 3 coins and of denominations 4 and 3.

4 + 4 + 3 = 11

c. The theoretical running time of this algorithm is O(CN) where C = the number of denominations and N is the amount to make change for.

4. Code submitted via OSU Teach site

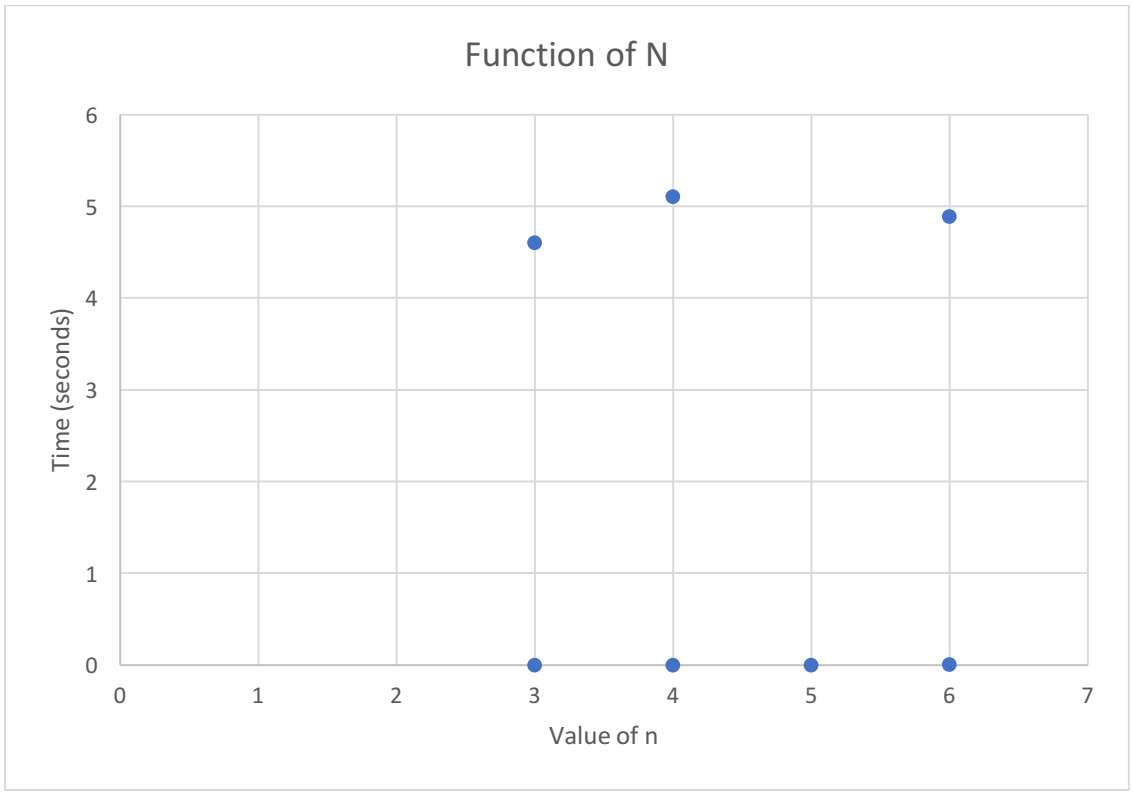
5. Making Change Experimental Running Time
   a. Experimental time data for the algorithm

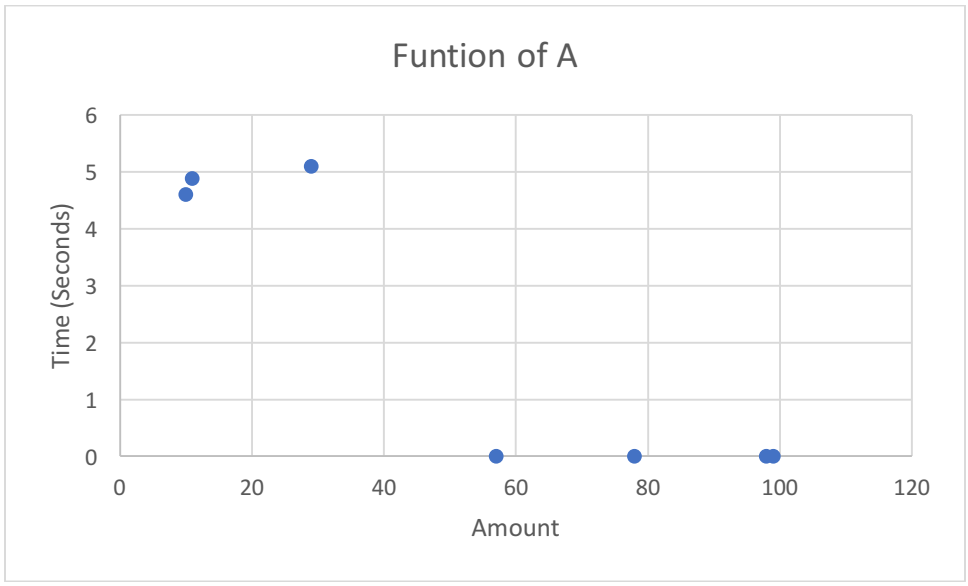
      Below is a table with the experimental running times I collected:

amount	n	time
10	3	4.601478
99	3	0.000156
78	4	0.000176
29	4	5.102157
57	5	0.000189
11	6	4.887581
98	6	0.002031

      My logic behind the chosen values is incrementing the n amount by one each time and trying it with a small amount and a large amount. What I found was that the when the amount was small it would take longer than a 0 second to find the minimum amount of coins to return the change.

   b. Plot on three separate graphs the running time as a function of A, running time as a function of n and running time as a function of nA.


Function of N
Time (seconds)
Value of n
0
1
2
3
4
5
6
7


Funtion of A
Time (Seconds)
Amount
0
20
40
60
80
100
120