Stephen Townsend
CS 325-400
10/15/17

Assignment 3

Problem 1:
15.1-2
Show, by 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 i 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 $1 \le i \le n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n - i.

i	1	2	3	4
p_i	1	38	60	60
p_i/i	1	19	20	15

Greedy	Counter
3 = 60	2 = 38
1 = 1	2 = 38
60 + 1 = 61	38 + 38 = 76

The greed example would take the highest density per cut. To do this with the above we would get a cut of 3 for 60 and have a remaining cut of 1 for 1. This totals 61. If we do not use greedy then we can do two cuts of 2 for 38 each and get a total of 76. As a result, the greedy strategy does not always work due to the counter example.

Problem 2:
15.1-3
Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, 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.

```
MODIFIED-BOTTOM-UP-CUT-ROD(p, n, c)
Let r[0..n] be a new array
r[0] = 0
for j = 1 to n
        q = p[j]                                    //No cuts are made
        for i = 1 to j
                q = max(q, p[i] + r[j-i] - c)
        r[j] = q
return r[n]
```

Because there is a fixed cost for cuts we need to add the cuts to the revenue equation. There will be a time where there are no cuts. As a result, this would require us not using the revenue equation but instead letting q just equal the price of rod (q = p[j]).

Stephen Townsend
CS 325-400
10/15/17

Problem 3:

a) Compute the product-sum

2, 1, 3, 5, 1, 4, 2 = 2 + 1 + (3 x 5) + 1 + (4 x 2) = 27

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

$OPT[j] = \{\max\{OPT[j-1] + v_j, OPT[j-2] + v_j * v_{j-1}\} \; if \; j \geq 2$

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

```
Prod-Sum(value[ ], n)
        if (n == 0)
                return 0;
        value[ ] OPT = value [n + 1];
        for j = 2 to n
                OPT[j] = max(OPT[j − 1] + v[j], OPT[j − 2] + v[j] ∗ v[j − 1]);
        return OPT[n];
```

Given if we use a bottom-top approach and use iteration the time complexity would be Θ(n) for having to iterate through the array.

Problem 4:

a) Using the bottom-up method with iteration. The program can be designed using two arrays. One array is used for total coins and the other is a tracker coin array. The function will go through each coin and iterate through the total array to determine if a coin can make the total. The total array goes from 0 to total change requested. If the total can be made with the current coin then the second array will hold that coin's position. This is a way to track and remember what coins can make what totals. If there is a coin that can make the total in less than the second array will hold the location of the lesser option. The end result is to take the last element in the array and subtract that with the coins location from the second array. By subtracting the totals of the coins, we are able to gather what the coin combination is. This will be the minimum amount of coins needed to make the designated total.

<u>Pseudocode</u>

```
minCoins(total, coins[])
        totalCArr of (total +1) = ∞
        trackCArr of (total+1) = -1
for i = 0 to n
        for j = 1 to coins.length
                if(j >= coins of (i))
                        if(totalCArr of (j-coins of (i)) +1 < totalCArr of (j))
                                totalCArr of (j) = 1 + totalCArr  of (j – coins of (i));
                                trackCArr of (j) = j
return totalCArr of (total)
```

Stephen Townsend
CS 325-400
10/15/17

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

$$T[i] = \{\min\{T[i], T(i - coins[j]{+}1\}$$

The theoretical running time for the Iterative bottom-up algorithm is Θ(n)

Problem 5:
Making Change Implementation
Submit a copy of all your files including the txt files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named amount.txt.

Problem 6:

a) Collect experimental running time data for your algorithm in Problem 4. Explain in detail how you collected the running times.

To obtain running times I needed a change amount that would be high enough along with a span of coin denomination that would not solve to quickly. Instead of just guessing on numbers I had a system of the coin denominations that went from 1 to 20 counting by 2 starting with 2. Ex 1, 2, 4, 6 … 20

To have a high enough number for change I decided to use the limit for an int in c. This would give me 2,147,483,647. I had an issue using max int because the code would use INT_MAX + 1 and this would give an overflow with an int array. As a result, I decided to use INT_MAX less 40. I then needed at least 7 numbers. To create a constant number generator, I took the change number and took 10% of the number as the next number (change - (change * 10%)).

I attempted this on my tablet because my laptop crashed a few days ago. I came to the problem that I ran out of RAM and I could not run the test locally since I did not have access to the 32gb of RAM from my laptop. As a result, I ran the results on the school flip server.

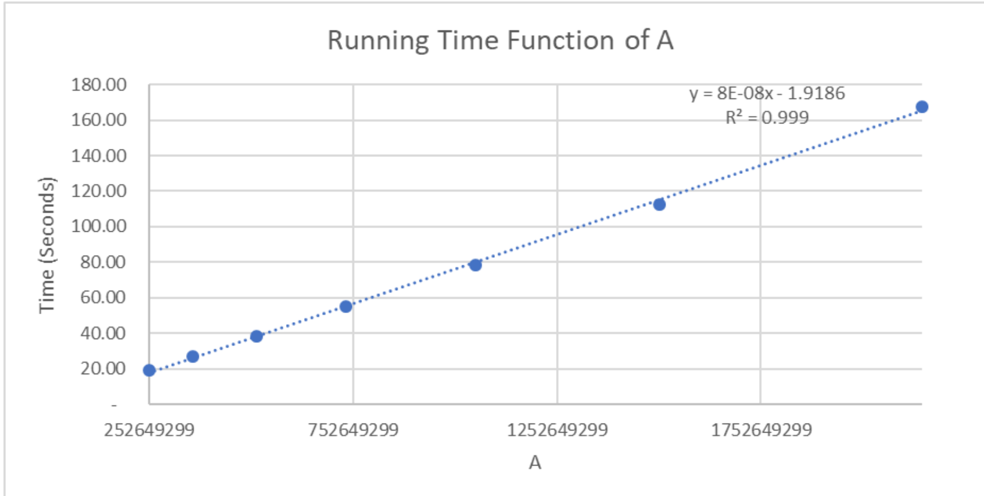
The times after running took a few minutes per coin change. I then decided to change the change numbers. Instead of 10% I attempted 30%. This gave me a pretty linear function.

b) On three separate graphs plot the running time as a function of A, running time as a function of n and running time as a function of nA. Fit trend lines to the data. How do these results compare to your theoretical running time? (Note: n is the number of denominations in the denomination set and A is the amount to make change)

Stephen Townsend
CS 325-400
10/15/17

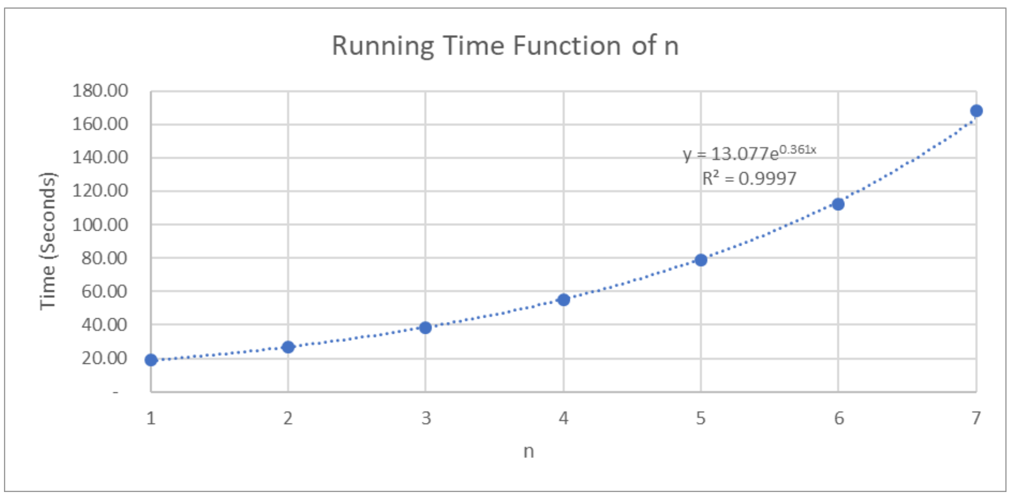
Function of A

252649299	18.99
360927570	26.95
515610814	38.47
736586877	54.98
1052266967	78.57
1503238525	112.48
2147483607	167.82



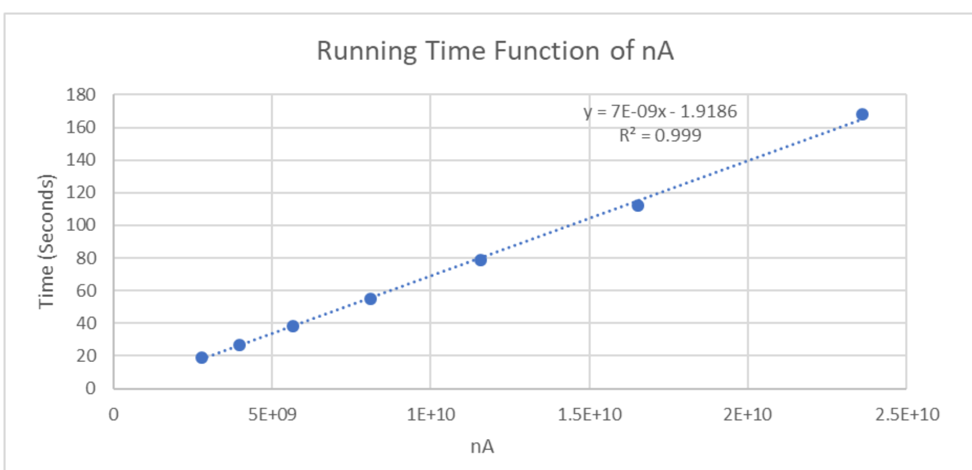
Function of n

1	18.99
2	26.95
3	38.47
4	54.98
5	78.57
6	112.48
7	167.82



Function of nA

2779142288	18.99
3970203268	26.95
5671718954	38.47
8102455649	54.98
11574936642	78.57
16535623774	112.48
23622319677	167.82



Stephen Townsend
CS 325-400
10/15/17

These results are in line with the theoretical running time. It was assumed that using bottom-top iteration that the time complexity would be $\Theta(n)$. This means that it is linear. Looking at the data from what I ran, I achieved that result. My actual results ran pretty linear and are thus in line with the $\Theta(n)$ complexity.