Thanhbinh Truong
CS325
933233558

Problem 1:
Let given rod length = 5, with a price array that has 1 = 1, 2 = 6 and 3 = 8.
Using the greedy strategy and looking at the density, we would have pieces of 2, 2, 1 which equals $13.
Using the dynamic programming, we would have pieces of 2 and 3, which equals $14.

Problem 2:

```
MODIFIED−CUT−ROD(p, n, c)
    let r[ 0 . . n ] be a new array
    r [0] = 0
    for j = 1 to n
        q = p[j]
        for i= 1 to j −1
            q = max(q , p[i] + r [ j−i ] − c)
    r [j] = q
    return r [n]
```

Problem 3:

a) The DP algorithm that would solve this problem would be to use a 2-degree matrix that contains the value of the score received that corresponds to certain questions answered that is against the total time T. The algorithm would then be run through the combinations possible and compare it against the highest score ultimately saving the two highest scores. At the end, it will save the highest score.

b)
```
A[i][j] = 0;                        // I = question, j = time used
for j = 0 to T                      // T = total time allowed
        A[0][j] = P[0][j]           // initialize array
for i is 1 to n (# of questions)
        for j = 0 to T
                high = 0            // highest score
                for k is 0 to j
                        score = A[i-1][j-k] + P[i][k]
                if score > high
                        high = score
                A[i][j] = high      // find the combination of the highest score
```

c) Runtime = O(n*T) = because you run through all the possible questions and possible time in each question.

d) No, Benny would definitely not use this algorithm if the professor gave partial credit. The algorithm depends on the time t used for each question n, and doing this for partial credit on an exam would require way more information, how questions would be split due to time, etc.

Problem 4:

A) Pseudocode:
   1) Assume V as a global variable
   2) Initialize and loop through 0^{th} row and 0^{th} column of array and find the number of coins using recursion

```
changeFunction(V,A)
        array1[] = new array
        array2[] = new array
        array3[] = new array
        array1[0] = 0 array2[0] = -1

        for int i = 1 to A
                array1[i] = information
                array2[i] = -1

        for int i = 0 to length(V)
                array3[i] = 0

        for int j = 0 to length of V
                for i = 1 to A
                        if i >= V[j]
                        if (array1[i-V[j]] + 1 < array1[i])
                                array1[i] = 1 + array1[i-V[j]]
                                array2[i] = j
        total = A
        while total != 0
                array3[array2[total]] += 1
                total = total - V[array2[total]]
return array3
```

B) Theoretical running time = Θ(V*A)

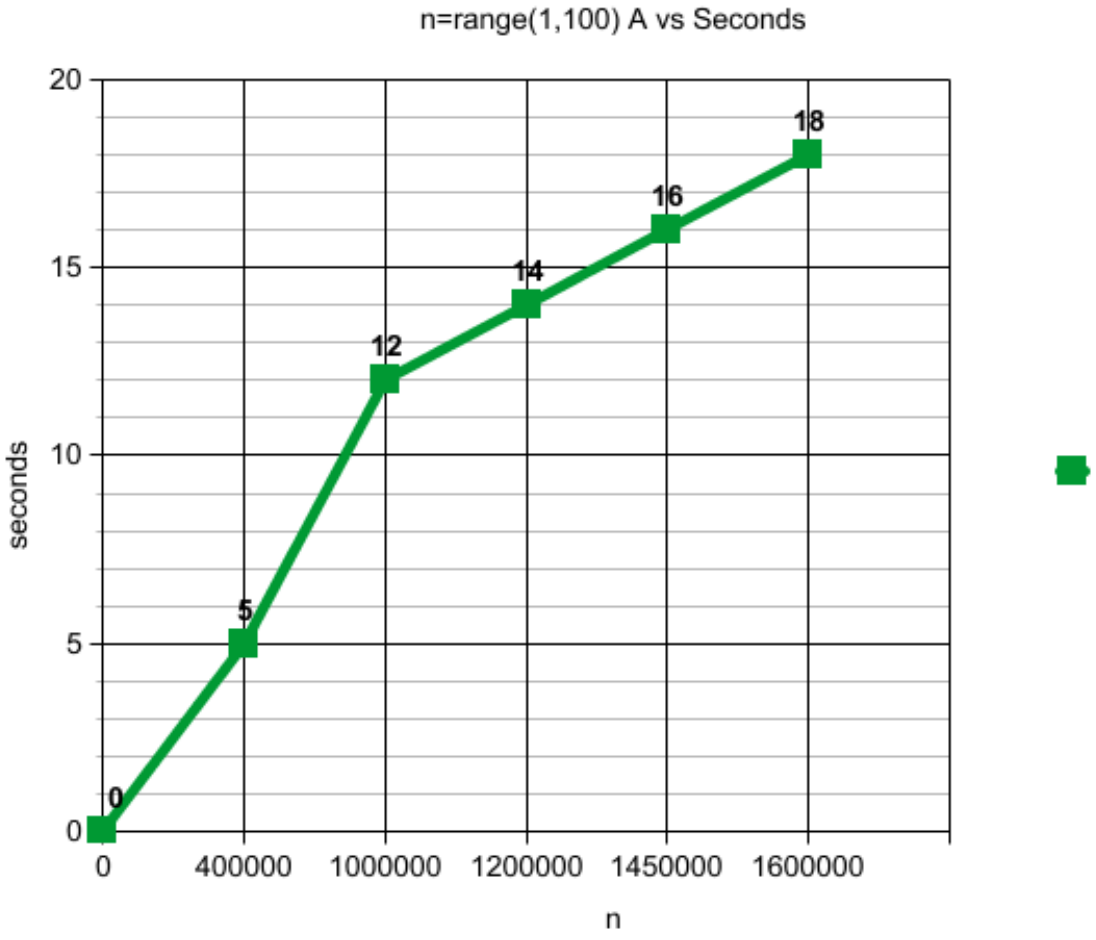
Problem 5: (done)

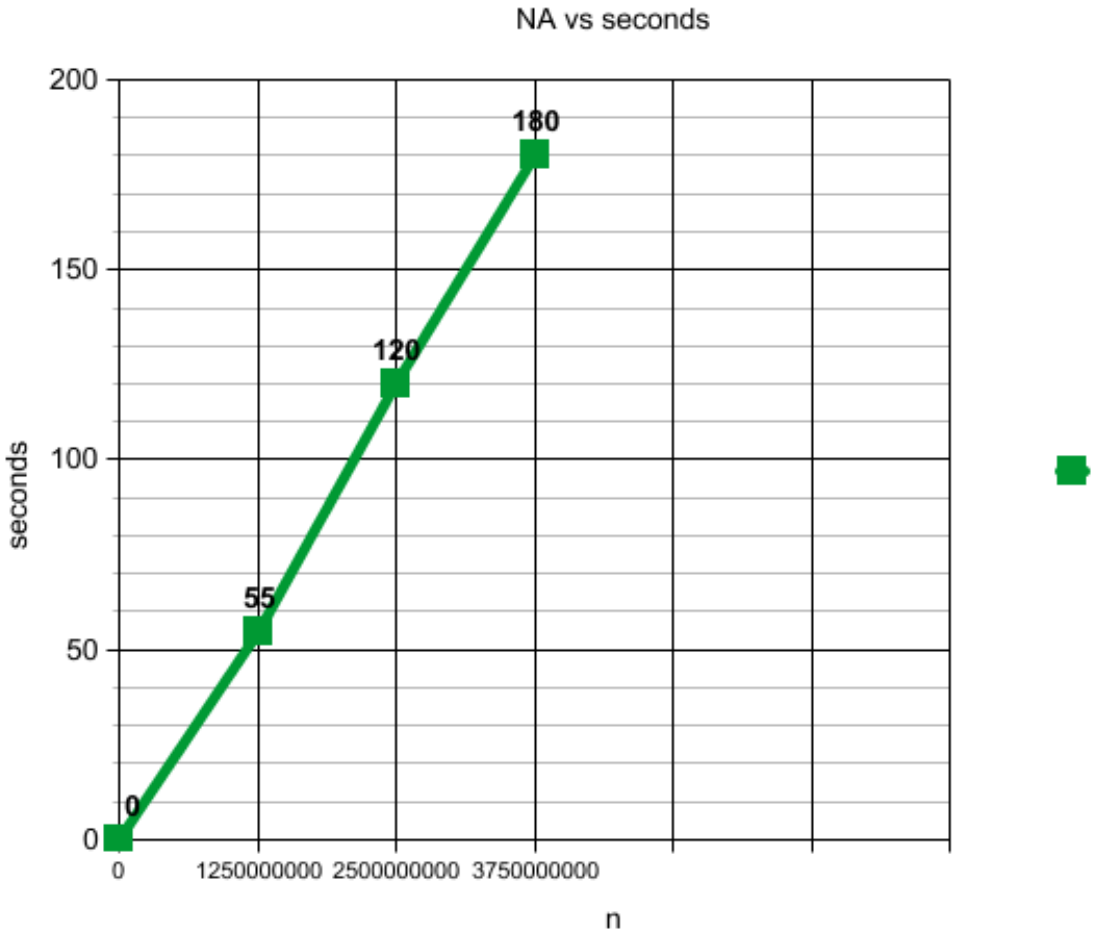
Problem 6:

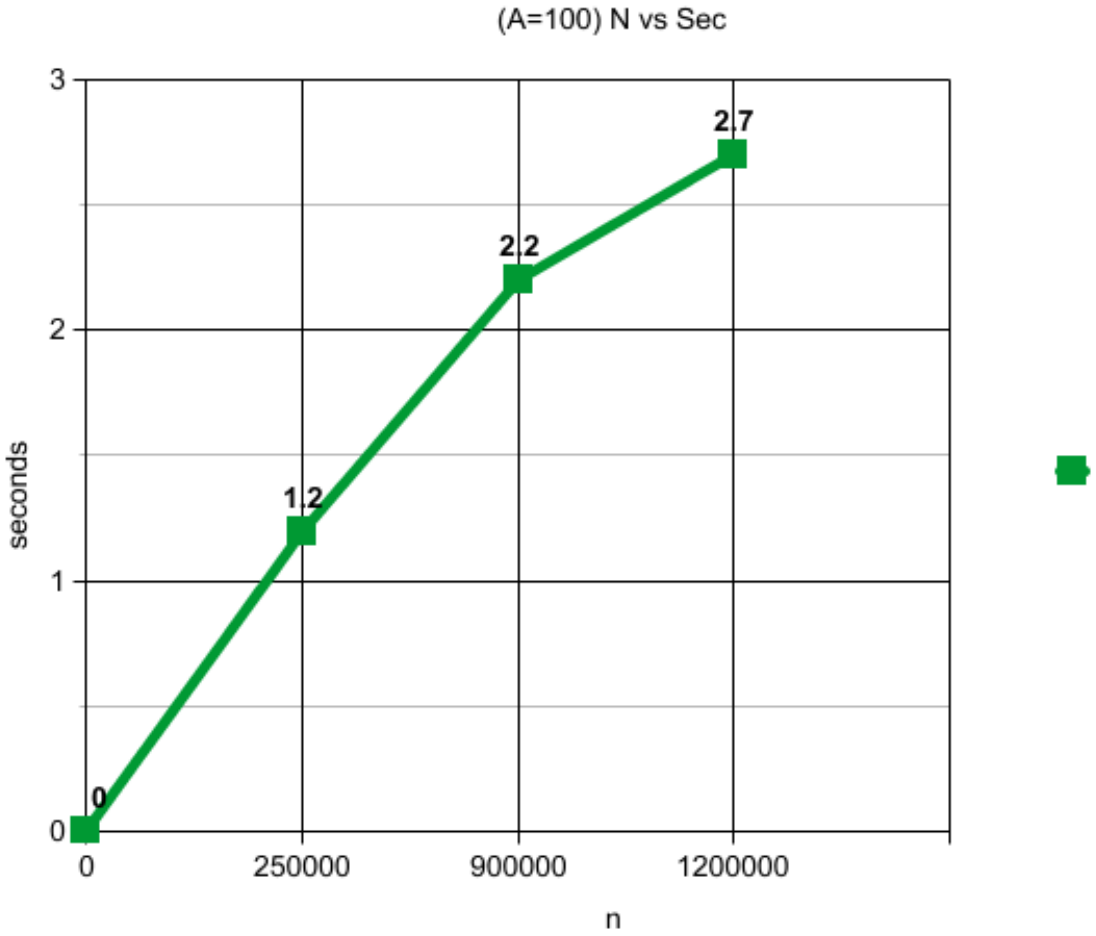
A) Using the following cases:

1. Large array for V that is increasing with a small A
2. Small array for V with increasing size and large A
3. Avg array of V w increasing size and avg A, increasing in size

b)


n=range(1,100) A vs Seconds
20
15
10
5
0
seconds
0
5
12
14
16
18
0
400000
1000000
1200000
1450000
1600000
n


NA vs seconds
200
150
100
50
0
seconds
180
120
55
0
0
1250000000
2500000000
3750000000
n



The results seem to be pretty close to the theoretical running time. If A was small and n was large, then n would dominate and make the running time be O(n) and vice versa. If A and n were large and somewhat close in value, the running time would then go towards $O(k^2)$, where k is approx. == n & a.