Project 2

CS325 — Spring 2015

---

by Group 2
Vedanth Narayanan
Jonathan Merrill
Tracie Lee

May 10, 2015

1 Dynamic Programming Table

Starting with i = 0 and incrementing to the amount of change passed in, we fill in the table with the minimum number of coins needed to make i cents. As i increases, we use previous smaller values that are stored in the table to compute the minimum number of coin for i cents. By building the table from the smallest values to the largest, we can reuse the work to find optimal numbers of coins for each iteration until we eventually get to the value passed in.

2 Algorithm Pseudocode

```
-- Divide and Conquer --

Define changeslowhelper(currency[], amount)
    // currency = array of coin denominations
    // amount = int, total amount we're making change for

    if amount == 0
        return 0
    for each coin in currency
        if coin == amount
            return [coin]

    for i to amount/2
        temp.extend(changeslowhelper(currency, i))
        temp.extend(changeslowhelper(currency, amount - 1))
        numCoins = length of temp

        if numCoins < minCoins
            coins = temp

    return coins

Define changeslow(currency[], amount)
    coins = changeslowhelper(currency[], amount)

    for each coin in currency
        result.append(coins.count(coin))
```

```
    return result

================================================================================
================================================================================
-- Greedy --

Define changegreedy (currency[], amount)
    int num

    for i to currency.length

        if currency[i] <= amount
            num = amount / currency[i]
            numArray[i] = numArray[i] + num
            total = total + num
            amount -= num * currency[i]

    numArray.append(total)  // sending the total through on the end of the array
                            // then we'll strip it off in the print function
    return numArray

================================================================================
================================================================================
-- Dynamic Programming --

Define changedp (currency[], amount)
    table[0] = 0
    table2[0] = 0
    for j from 1 to amount
        for i from 0 to len(currency)
            if currency[i] <= j
                min = 1 + table[j - currency[i]]
                coin = i
        append min to table
        append coin to table2

    coins = []
```

```
while amount > 0
    append table2[amount] to result
    decrement amount by table2[amount]

result = []
for each coin in currency
append the number of each coin in coins to result

return result
```

3 Dynamic Programming Induction Proof

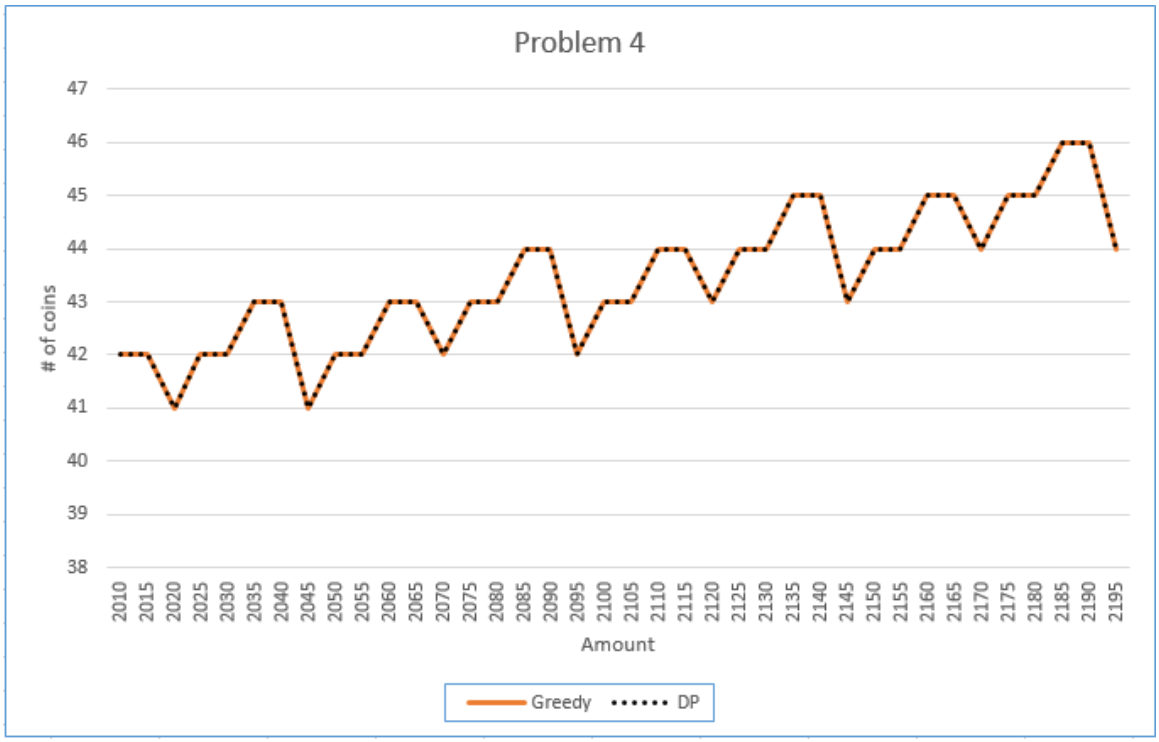
Proof by Induction:

- **Base Case**
  T[0] = 0. This is true because for 0 cents, the optimal number of coins used is 0.

- **Inductive Hypothesis**
  We assume that for some arbitrary value 'k', T[k] is the minimum number of coins used to make change for k cents. This also assumes that T[p] is correct, where p is any value less than or equal to k, due to the nature of the problem.

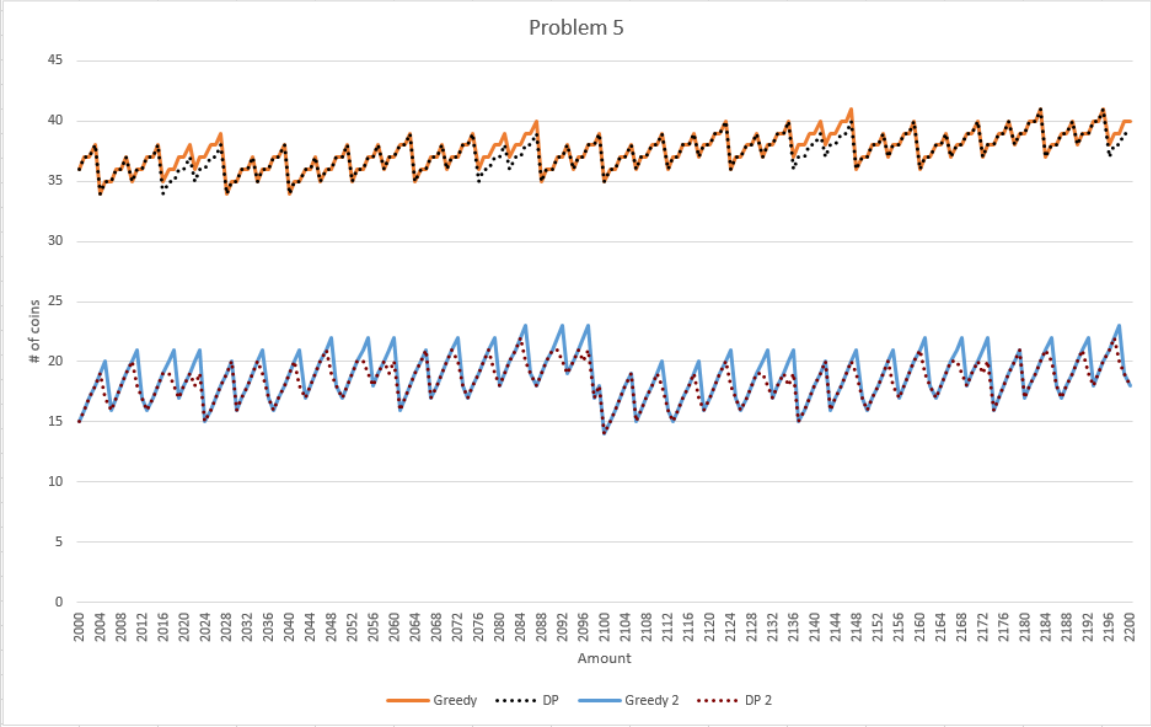
- **Inductive Step**
  We must now prove that T[k+1] is also correct:
  Since:
  T[k+1] = T[(k+1) - i] + 1; where i is some value less than or equal to k
  Then:
  T[k+1] = T[p] + 1; where p is some value less than or equal to k
  Since we know T[p] is the correct number of coins used for p cents, T[k+1] must also be the correct number of coins used for k+1 cents.

4 Greedy and DP Algorithms for V = [1, 5, 10, 25, 50]



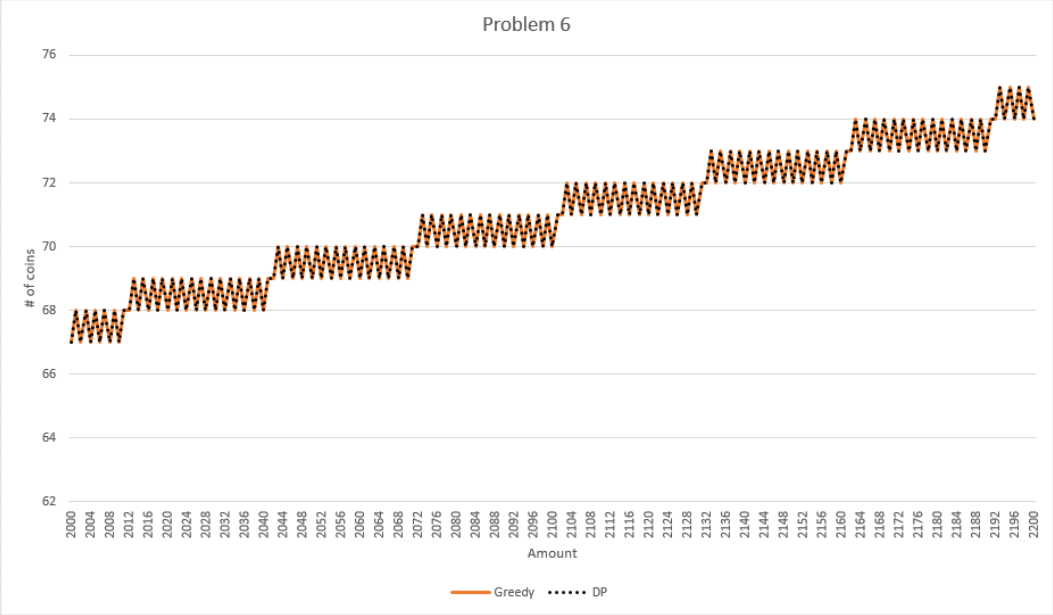
As you can see, both algorithms are identical in the results produced for V = [1,5,10,25,50].

5 Greedy and DP Algorithms for V = [1, 2, 6, 12, 24, 48, 60] and V = [1, 6, 13, 37, 150]



For some values in the Dynamic Programming algorithm, more minimized results are returned in comparison to the greedy algorithm. This is true for both V = [1, 2, 6, 12, 24, 48, 60] and V = [1, 6, 13, 37, 150].

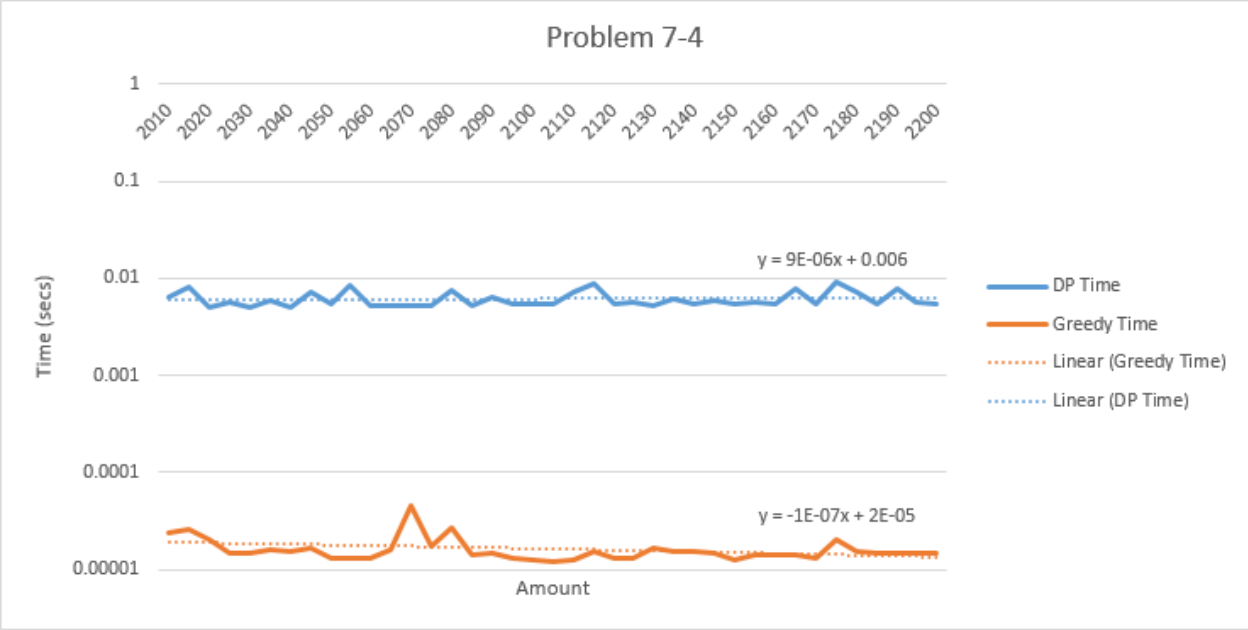
6 Greedy and DP Algorithms for V = [1, 2, 4, 6, 8, 10, 12, ..., 30]



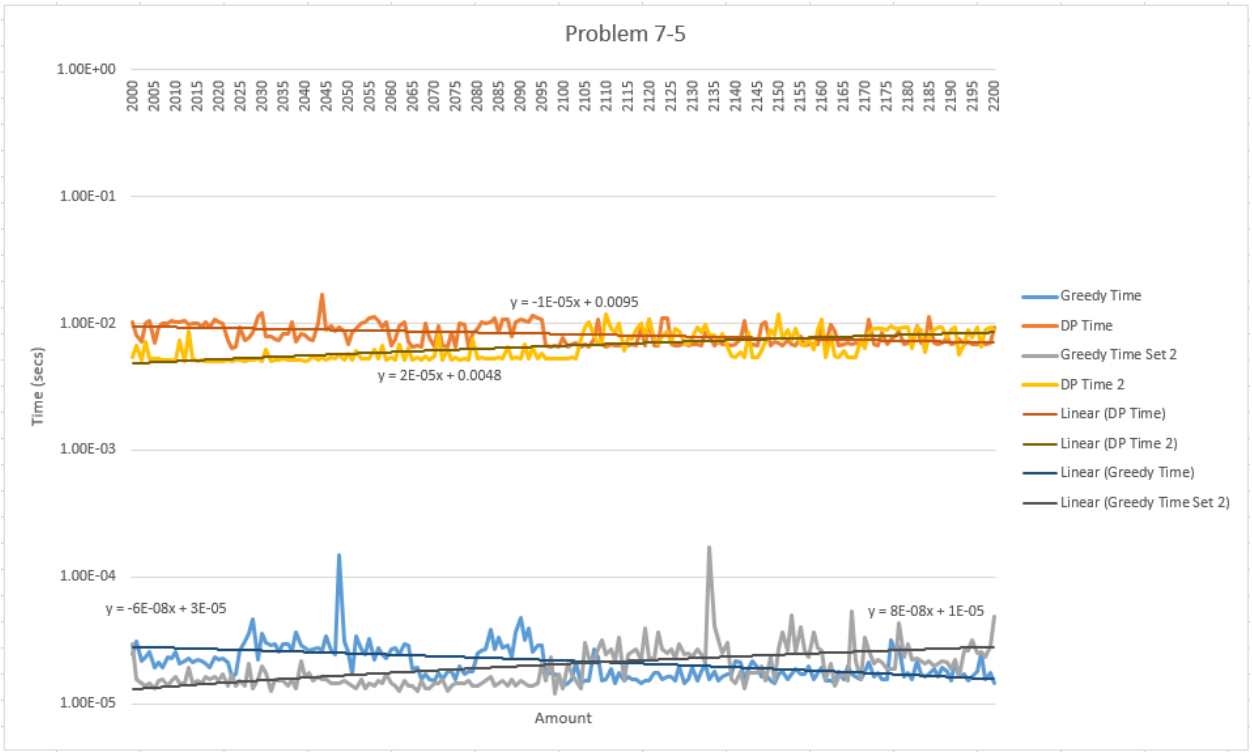
As you can see, both algorithms are identical in the results produced for V = [1, 2, 4, 6, 8, 10, 12, ..., 30].

7 Running Times for Greedy and DP Algorithms

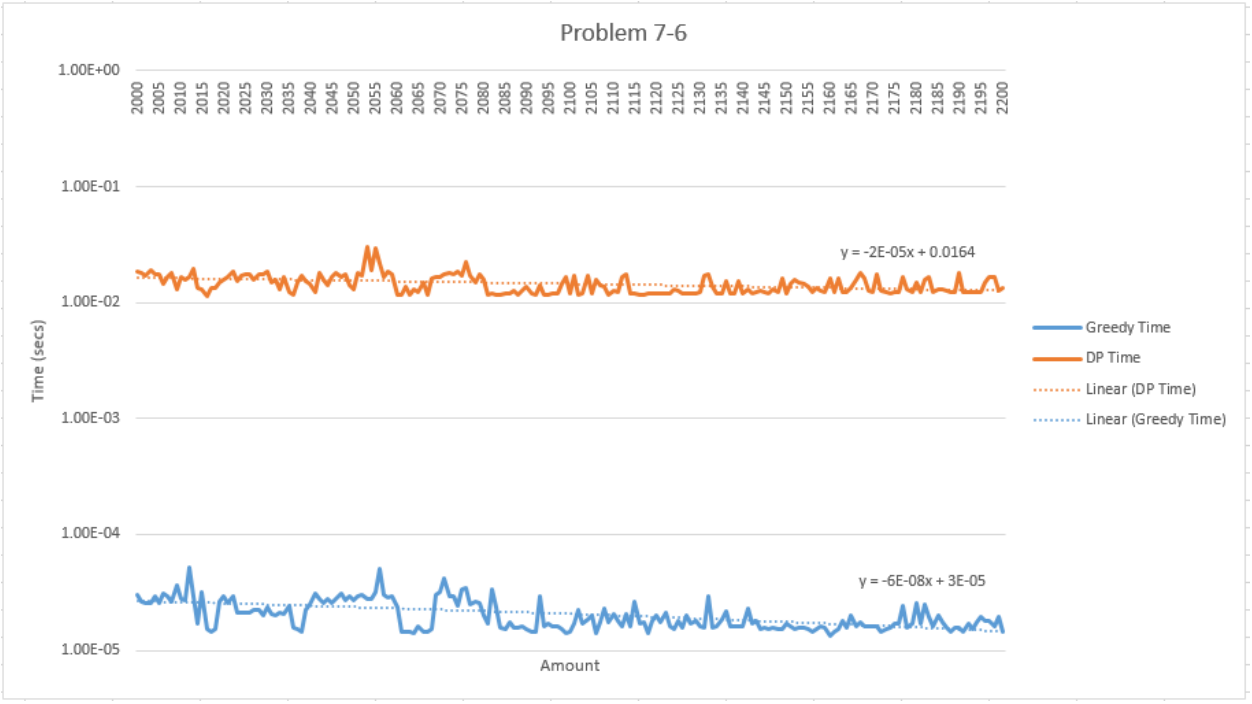
Problem 4



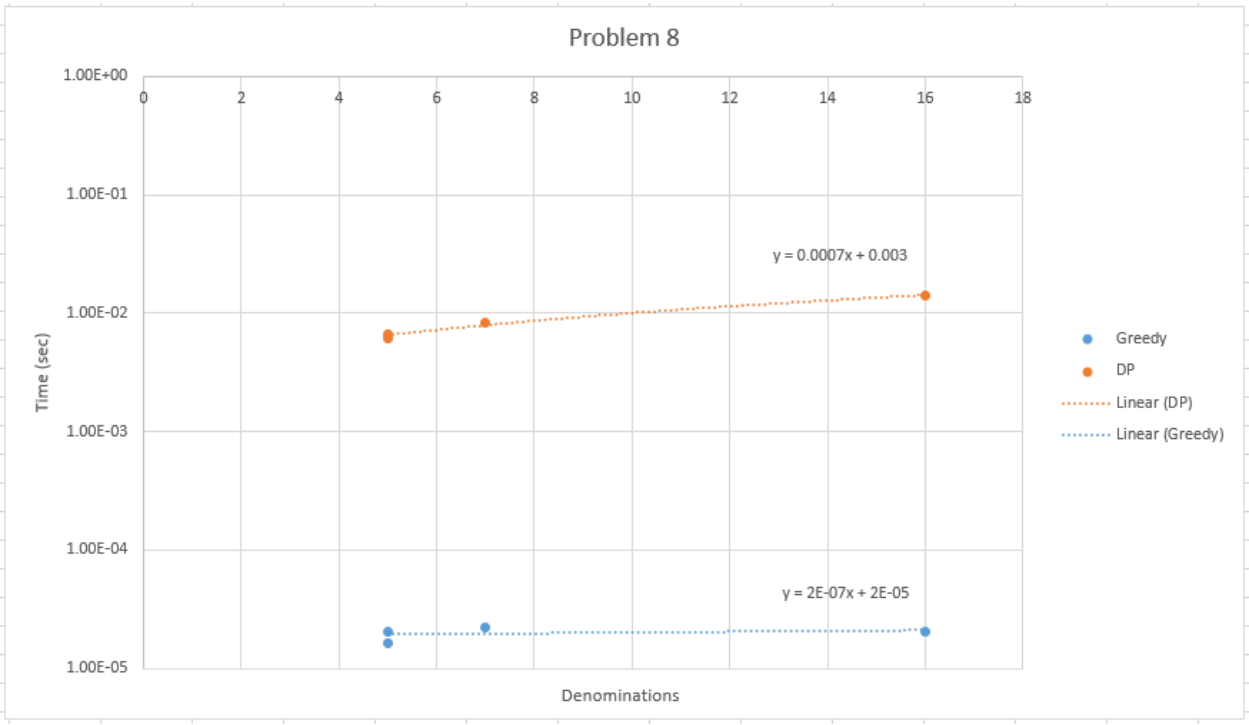
Problem 5


Problem 7-5
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
Time (secs)
Amount
y = -1E-05x + 0.0095
y = 2E-05x + 0.0048
y = -6E-08x + 3E-05
y = 8E-08x + 1E-05
Greedy Time
DP Time
Greedy Time Set 2
DP Time 2
Linear (DP Time)
Linear (DP Time 2)
Linear (Greedy Time)
Linear (Greedy Time Set 2)

Problem 6


Problem 7-6
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
Time (secs)
Amount
y = -2E-05x + 0.0164
y = -6E-08x + 3E-05
Greedy Time
DP Time
Linear (DP Time)
Linear (Greedy Time)

8 Running Times vs Number of Denominations for Greedy and DP Algorithms



The number of denominations (n) influences the running times of the two algorithms. The difference would be more prevalent with larger values for n.

9 Greedy vs DP for V = $[p^0, p^1, p^2, \ldots, p^n]$

The greedy algorithm will definitely operate faster because it looks for the largest element in V that it can subtract from A, where DP will check for all elements in V.