CS 325 Project 2: Coin Change
Brandon Swanson
10/28/15

1)
Two tables are used in the dynamic programming algorithm both are a single array of length A+1 indexed from 0 to A, one for storing the minimum coins needed to make each value less than and equal to A (array T) and another for storing the last denomination of coin used to reach this value in an optimum way (array C) stored as an index into the set V.

Both tables are filled starting from 1 (with a 0 value placed in T[0] as it takes 0 coins to reach amount 0). determining the value of T[i] depends on the values of T[0..i-1], the minimum number of coins to make value i is min(T[i-V[j]]) + 1 where i-V[j] >= 0. An inductive proof of this way of determining T[i] will follow in question 3 but informally we can see that this is equivalent to choosing what the first valued coin should be used first by evaluating the relevant subproblems of the minimum amount of coins needed to generate A-V[i] for the chosen i.

This optimum choice of first coin is what is recorded in the C[1...A] array, at each optimum solution calculated for T[i] the denomination used to achieve that minimum is recorded in C[i]. Then when the minimum number of coins required a solution is back traced by starting with index A of C and then subtracting this index by the value of the coin indexed in V at C[backtrace-Index], tallying the amount used for each denomination.

2)

ChangeSlow Algorithm Pseudo Code:

```
slowHelper(Values[1...N],Amount)
     for i = 1 to N
          if Amount == Values[i]
               return new array [I]

     bestKI = INFINITY
     bestKICoins = NULL

     for i = 1 to k//2 + 1
          leftCoins = slowHelper(Values,i)
          rightCoins = slowHelper(Values,amount-i)
          if leftCoins.length + rightCoins.length < bestKI
               bestKI =  leftCoins.length + rightCoins.length
               bestKICoins = leftCoins + rightCoins

     return bestKICoins

Slow(Values[1...N],Amount)
     let Result[1..N] be a new array [0,0..,0]

     coinsUsed = slowHelper(Values,Amount)

     for i = 1 to coinsUsed.length
          result[coinsUsed[i]] += 1

     return result
```

GreedyChange Algorithm Pseudo Code:

```
Greedy(Values[1...N],Amount)
     let Result[0..N] be a new array

     for i=N down to 1
          Result[i]= Amount//Values[i]
          Amount = Amount%Values[i]

     return Result
```

ChangeDP Algorithm Pseudo Code:

```
DP(Values[1...N],Amount)

     let minCoins[0..Amount] be a new array
     minCoins[0] = 0
     minCoins[1-Amount] = INFINITY

     let trace[1..Amount] be a new array


     for i = 1 to Amount
          coinIndex = 0
          while coinIndex <= N and Values[coinIndex] <= I

               if minCoins[i-Values[coinIndex]] + 1 < minCoins[i]
                    minCoins[i] = minCoins[i-Values[coinIndex]] + 1
                    trace[i] = coinIndex

               coinIndex += 1


     let Result[1..N] be a new array [0,0..,0]
     traceIndex = amount
     while traceIndex > 0:
          coinUsed = trace[traceIndex]
          result[coinUsed] +=1
          traceIndex -= coins[coinUsed]
     return result
```

3)

Base case: T[0] = 0

Strong Inductive Hypothesis: T[0-k] = fewest number of coins needed to make change for T[i]

0 <= i <= k

fewest coins needed to make change for k+1 using any individual coin as the last coin will be the that one coin plus the fewest needed to make k+1 minus the value of that coin. Formally

T [k+1] using V[i] = 1 + fewest(k+1 - V[i])

the fewest possible using any coin will be the min of the above formula using all coins that are less than or equal to k+1, Therefore:

$T[k+1] = \min_{v[i]<=k+1} \{T[k+1 – V[i]] +1\}$

proving the inductive hypothesis.

EXPERIMENTAL ANALYSIS
please see attached recorded results and times and plot charts, all times recorded in milliseconds

It can be seen by comparing the output of the ChangeSlow algorithm to that of the ChangeDP algorithm for low values that it is providing optimum solutions. However due to its running time the number of coins as a function of A for problems 4-6 are compared on amounts less than 36. but are not plotted as they have the same relationship to ChangeGreedy as ChangeDP

4) With these denominations of coins the dynamic programming and greedy algorithm generate the same values of minimum number of coins, In other words they both find an optimal solution.

5) The greedy algorithm produces a non-optimal solution for both sets of denominations, the amount of non-optimal answers was measured experimentally by comparing the output of the dynamic programming algorithm. Both sets of values produced very similar ratios of optimal to non optimal solutions.
V_1 greedy non-optimal: 41 out of: 201 tests for an average of: 20.39% non-optimal solutions
V_2 greedy non-optimal: 37 out of: 201 tests for an average of: 18.40% non-optimal solutions

however the greedy algorithm produced non-optimal solutions repeatedly in groups of 12 for sequentially inputs of Amount on the denomination set V_1 however on values set V_2 they were more sporadicly dispersed.

6) Once again the greedy algorithm always produces optimal solutions for this set of denominations of coins.

7) see attached plots of recorded running time. ChangeSlow has such an explosive runtime that only amounts less than 36 could be recorded (12 hours of computer time were expended attempting to record the time required to calculate the minimum coins for amount=40). A separate comparison of all 3 algorithms for amounts less than 36 on all problems denomination sets is included. At these low values there are outliers for the ChangeSlow algorithm as if the amount being calculated is one of the coin denominations then the base case will return that single coin in constant time without ever entering the base case. ChangeSlow on Problem 6, with the even coin denominations up to 30 has an apparent constant running time for amount 1 through 30, but after amounts greater than the largest denomination the exponential time would begin.
Therefore the 3 algorithms are only plotted for problem V=[1,5,10,25,50] and V=[1] for amount 1 to 35.

It can be determined from the experimental analysis that the running times can be characterized by a function of A as such:

ChangeSlow: $f(A) = O(C^A)$ Exponential
ChangeDP = $f(A) = O(A)$ - Linear Time
ChangeGreedy = $f(A) = O(1)$ – Constant Time

The plot of ChangeGreedy appears to have a number of outliers but upon careful look at the scale of the plot it can be seen that the running times never vary by more than a hundredth of a millisecond. This much variance is insignificant and it can be accurately claimed that A has no effect on the running time of the ChangeGreedy algorithm.

8) The size of n has an effect on both ChangeGreedy and ChangeDP, for changeDP it is only one of the factors, but for ChangeGreedy n is the only determinant of its running time.

Evaluating the algorithms experimental and theoretically it can be seen that the running time of ChangeGreedy is $\Theta(n)$, and the running time of ChangeDP is $O(An)$ there are A iterations to fill the dynamic programming table and on each one at most n comparisons are made.

9) In such a case the greedy algorithm would always produce an optimal solution. This can be ascertained by considering that a locally optimal solution will be globally optimal.
For whatever original or remaining amount being consider, A, it will be $p^k <= A < p^{k+1}$ and for any value exponent k, to sum up to p^k with the values $p^{k-1} .. p^0$ the sum of the coefficients will always be greater than 1, at a minimum they will be p, therefore the globally optimal strategy is to repeatedly divide by the greatest possible denomination.

Therefore the greedy algorithm would be a preferable algorithm as its running time would be better. As the greedy algorithms runtime is dependent on the number of denominations (and therefore the number of divisions and mod operations to perform) its runtime is $\Theta(n)$.
the number of denominations in this imaginary scenario could be very large, larger than A, but if the greedy algorithm was modified to start its divisions at the largest denomination less than or equal to A then it would be upper bound by the number of denominations less than A, which is $\Theta(\log_p A)$, making the greedy algorithm $O(\log_p A)$ while the dynamic programming solution would still be $\Theta(An)$