Data Structures (CS 1520) Lab 6 Name:_____

**Objective:** To gain experience with backtracking via recursion, and writing a dynamic programming solution

**To start the lab:** Download and unzip the file at: www.cs.uni.edu/~fienup/cs1520f12/labs/lab6.zip

**Part A:** The `coin_no_globals.py` file contains the recursive backtracking algorithm discussed in lecture to implement the "coin-change" problem: "Given a set of coin types and an amount of change to be returned, it determines the **fewest number** of coins for this amount of change."

a) Run the `coin_no_globals.py` program with the following input:

| Change Amount | Order of Coin Types | Run-Time (seconds) | Number of Backtracking Nodes |
|---|---|---|---|
| 29 | 1 5 10 12 25 50 | | |
| 29 | 50 25 12 10 5 1 | | |

b) Explain why the algorithm performs better for the descending order of coins.

c) Draw the complete search-space recursion tree for 29 cents change and coin types: 50 25 12 10 5 1

d) Each node of the search-space (recursive-call) tree maintains the state of a partial solution. In general the partial solution state consists of potentially large arrays that change little between parent and child. For example, the current `coin_no_globals.py` program copies the `numberOfEachCoinType` array and updates one spot:

```
        else:
            # call child with updated state information
            smallerChangeAmtNumberOfEachCoinType = [] + numberOfEachCoinType
            smallerChangeAmtNumberOfEachCoinType[index] += 1
```

To avoid having to make multiple copies of the `numberOfEachCoinType` array, a reference to a single "global" array can be maintained which is updated before we go down to the child (via a recursive call) and undone when we backtrack to the parent. Modify the `coin_no_globals.py` program to include this improvement.

e) Uncomment the `print` statement which is the next to last line of `solveCoinChange`. Explain the resulting output of this `print` statement.

**After you have completed part A, raise your hand and explain your results**.

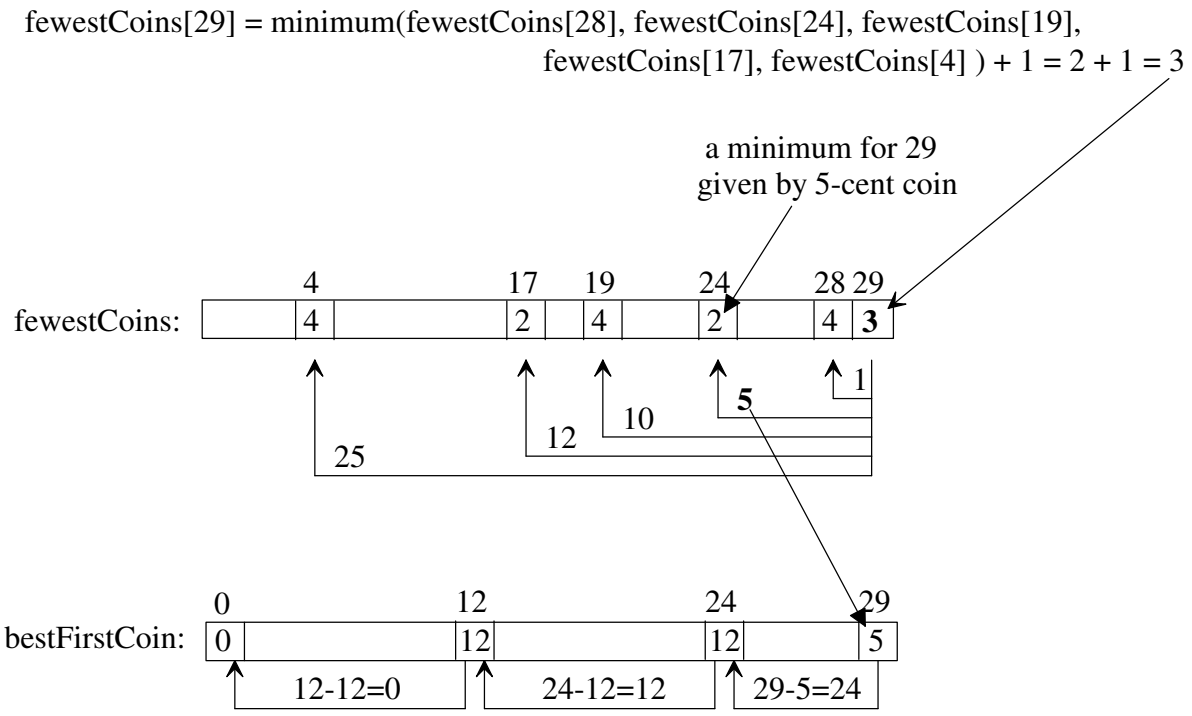
**Part B.** The `coinDynPgmming.py` file contains a partial solution to the dynamic programming coin-change problem. Your job is to complete the dynamic programming solution by having it calculate the fewestCoins and bestFirstCoin arrays.

Recall that the dynamic programming algorithm is:

I. Fills an array fewestCoins from 0 to the amount of change. An element of fewestCoins stores the fewest number of coins necessary for the amount of change corresponding to its index value.

For 29-cents using the set of coin types {1, 5, 10, 12, 25, 50}, the dynamic programming algorithm would have previously calculated the fewestCoins for the change amounts of 0, 1, 2, ..., up to 28 cents.

II. If we record the best, first coin to return for each change amount (found in the "minimum" calculation) in a list bestFirstCoin, then we can easily recover the actual coin types to return.




**EXTRA CREDIT: In main, complete the code to extract the coins in the solution.**

Extract the coins in the solution for 29-cents from bestFirstCoin[29], bestFirstCoin[24], and bestFirstCoin[12]

**After you have completed your dynamic programming solution and debugged it, raise your hand and demonstrate your code.**