1) As in the maze problem of homework #3 we could use backtracking with a stack to maintain the list of unexplored choices, but we can also use recursion (with its run-time stack) to drive a backtracking algorithm. The general recursive backtracking algorithm for optimization problems (e.g., fewest number of coins) looks something like:

```
Backtrack( recursionTreeNode p ) {

    treeNode c;
    for each child c of p do                              # each c represents a possible choice
        if promising(c) then                              # c is "promising" if it could lead to a better solution
            if c is a solution that's better than best then   # check if this is the best solution found so far
                best = c                                  # remember the best solution
            else
                Backtrack(c)                              # follow a branch down the tree
            end if
        end if
    end for
} // end Backtrack
```

General Notes about Backtracking:

- The depth-first nature of backtracking only stores information about the current branch being explored so the memory usage is “low”
- 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. To avoid having multiple copies of these arrays, a single “global” state is maintained which is updated before we go down to the child (via a recursive call) and undone when we backtrack to the parent.

For the coin-change problem:

a) What defines the current state of a search-space tree node?

b) When would a “child” search-space tree node NOT be promising?

Coin-change backtracking code in Python:

```
# global current state of the backtrack
numberOfEachCoinType = []
numberOfCoinsSoFar = 0
solutionFound = False
bestFewestCoins = -1
bestNumberOfEachCoinType = None

# profiling
backtrackingNodes = 0

def main():
    changeAmt = int(raw_input("Enter the change amount: "))
    coinTypes = raw_input("Enter the coin types separated by spaces: ")
    coinTypes = coinTypes.split(" ")
    for index in xrange(len(coinTypes)):
        coinTypes[index] = int(coinTypes[index])
    print "Change Amount:",changeAmt,"   Coin types:", coinTypes
    fewestCoins, numberOfEachCoinType = solveCoinChange(changeAmt, coinTypes)
    print "Fewest number of coins",fewestCoins
    print "The number of each type of coin in the solution is:"
    for index in xrange(len(coinTypes)):
        print "number of %s-cent coins is %s"%(str(coinTypes[index]),
                                               str(numberOfEachCoinType[index]))
    return

def solveCoinChange(changeAmt, coinTypes):

    def backtrack(changeAmt):
        global numberOfEachCoinType
        global numberOfCoinsSoFar
        global solutionFound
        global bestFewestCoins
        global bestNumberOfEachCoinType
        global backtrackingNodes
        backtrackingNodes += 1

        for index in xrange(len(coinTypes)):
            smallerChangeAmt = changeAmt - coinTypes[index]
            if promising(smallerChangeAmt, numberOfCoinsSoFar+1):
                if smallerChangeAmt == 0:  # a solution is found
                    # check if its best
                   if (not solutionFound) or numberOfCoinsSoFar + 1 < bestFewestCoins:
                            bestFewestCoins = numberOfCoinsSoFar+1
                        bestNumberOfEachCoinType = [] + numberOfEachCoinType
                        bestNumberOfEachCoinType[index] += 1
                        solutionFound = True
                else:
                    # update global "current state" for child before call
                    numberOfCoinsSoFar += 1
                    numberOfEachCoinType[index] += 1

                    backtrack(smallerChangeAmt)

                    # undo change to global "current state" after backtracking
                    numberOfCoinsSoFar -= 1
                    numberOfEachCoinType[index] -= 1
    # end def backtrack
```

```
    def promising(changeAmt, numberOfCoinsReturned):
        if changeAmt < 0:
            return False
        elif changeAmt == 0:
            return True
        else:  # changeAmt > 0
            if solutionFound and numberOfCoinsReturned+1 >= bestFewestCoins:
                return False
            else:
                return True

    # set-up initial "current state" information
    global numberOfEachCoinType
    global numberOfCoinsSoFar
    global solutionFound
    global bestFewestCoins
    global bestNumberOfEachCoinType

    numberOfEachCoinType = []
    for coin in coinTypes:
        numberOfEachCoinType.append(0)
    numberOfCoinsSoFar = 0
    solutionFound = False
    bestFewestCoins = -1
    bestNumberOfEachCoinType = None

    backtrack(changeAmt)
    return bestFewestCoins, bestNumberOfEachCoinType

main()
print "Number of Backtracking Nodes:", backtrackingNodes
```

c) Consider the output of running the above backtracking code twice with a change amount of 399 cents.

```
>>>
Enter the change amount: 399
Enter the coin types separated by spaces: 1 5 10 12 25 50
Change Amount: 399    Coin types: [1, 5, 10, 12, 25, 50]
Run-time: 1289.869 seconds
Fewest number of coins 10
The number of each type of coin in the solution is:
number of 1-cent coins is 0
number of 5-cent coins is 0
number of 10-cent coins is 0
number of 12-cent coins is 2
number of 25-cent coins is 1
number of 50-cent coins is 7
Number of Backtracking Nodes: 302715760
>>> ============== RESTART =============
>>>
Enter the change amount: 399
Enter the coin types separated by spaces: 50 25 12 10 5 1
Change Amount: 399    Coin types: [50, 25, 12, 10, 5, 1]
Run-time: 8.360 seconds
Fewest number of coins 10
The number of each type of coin in the solution is:
number of 50-cent coins is 7
number of 25-cent coins is 1
number of 12-cent coins is 2
number of 10-cent coins is 0
number of 5-cent coins is 0
number of 1-cent coins is 0
Number of Backtracking Nodes: 2015539
>>>
```

a) Explain why ordering the coins from largest to smallest produced faster results.

```
backtrackingNodes = 0  # profiling variable to track number of state-space tree nodes

def main():
    changeAmt = int(raw_input("Enter the change amount: "))
    coinTypes = raw_input("Enter the coin types separated by spaces: ")
    coinTypes = coinTypes.split(" ")
    for index in xrange(len(coinTypes)):
        coinTypes[index] = int(coinTypes[index])
    print "Change Amount:",changeAmt,"   Coin types:", coinTypes
    fewestCoins, numberOfEachCoinType = solveCoinChange(changeAmt, coinTypes)
    print "Fewest number of coins",fewestCoins
    print "The number of each type of coin in the solution is:"
    for index in xrange(len(coinTypes)):
        print "number of %s-cent coins is %s"%(str(coinTypes[index]), str(numberOfEachCoinType[index]))
    return

def solveCoinChange(changeAmt, coinTypes):

    def backtrack(changeAmt, numberOfEachCoinType, numberOfCoinsSoFar, solutionFound, bestFewestCoins, bestNumberOfEachCoinType):
        global backtrackingNodes
        backtrackingNodes += 1

        for index in xrange(len(coinTypes)):
            smallerChangeAmt = changeAmt - coinTypes[index]
            if promising(smallerChangeAmt, numberOfCoinsSoFar+1, solutionFound, bestFewestCoins):
                if smallerChangeAmt == 0:  # a solution is found
                    if (not solutionFound) or numberOfCoinsSoFar + 1 < bestFewestCoins: # check if its best
                        bestFewestCoins = numberOfCoinsSoFar+1
                        bestNumberOfEachCoinType = [] + numberOfEachCoinType
                        bestNumberOfEachCoinType[index] += 1
                        solutionFound = True
                else:
                    # call child with updated state information
                    smallerChangeAmtNumberOfEachCoinType = [] + numberOfEachCoinType
                    smallerChangeAmtNumberOfEachCoinType[index] += 1

                    solutionFound, bestFewestCoins, bestNumberOfEachCoinType = backtrack(smallerChangeAmt, smallerChangeAmtNumberOfEachCoinType,
                                                                                         numberOfCoinsSoFar + 1, solutionFound,
                                                                                         bestFewestCoins, bestNumberOfEachCoinType)
        return solutionFound, bestFewestCoins, bestNumberOfEachCoinType
    # end def backtrack

    def promising(changeAmt, numberOfCoinsReturned, solutionFound, bestFewestCoins):
        if changeAmt < 0:
            return False
        elif changeAmt == 0:
            return True
        else:  # changeAmt > 0
            if solutionFound and numberOfCoinsReturned+1 >= bestFewestCoins:
                return False
            else:
                return True

    # set-up initial "current state" information
    numberOfEachCoinType = []
    numberOfCoinsSoFar = 0
    solutionFound = False
    bestFewestCoins = -1
    bestNumberOfEachCoinType = None

    numberOfEachCoinType = []
    for coin in coinTypes:
        numberOfEachCoinType.append(0)
    numberOfCoinsSoFar = 0
    solutionFound = False
    bestFewestCoins = -1
    bestNumberOfEachCoinType = None

    solutionFound, bestFewestCoins, bestNumberOfEachCoinType = backtrack(changeAmt, numberOfEachCoinType, numberOfCoinsSoFar, solutionFound,
                                                                         bestFewestCoins, bestNumberOfEachCoinType)
    return bestFewestCoins, bestNumberOfEachCoinType

main()
print "Number of Backtracking Nodes:", backtrackingNodes
```