ExactChange:

This problem rounded the set out perfectly. Only one type of solution will work without timing out. Given the difficulty of a solution, the problem was written kindly. The special cases were handled intuitively, so that the coders were likely to handle them properly by default. In addition, a time-out case was presented.

The solution requires a bit of dynamic programming. Create an array of Boolean whose indices are amounts of money between $0.00 and $500.00 in cents. An element of the array will be set to "true" if that combination is representable by adding up some of the customer's money and subtracting some of the cashier's money.

The key to filling the array efficiently is to set up three loops in the following way:

1. An outside loop that iterates through the different customer denominations (call the denomination amount D)
2. A second loop that iterates through the array backward (call the index I).
3. A third loop that runs when the Ith element of the array is "true" It assigns the value "true" to elements whose indices are of the form I + KD where K is between 0 and A, inclusive and A is the total number of coins the customer has of denomination D.

Loop 2 needs to run backward so that we won't inadvertently mark the value I + KD + K'D "true" where K and K' are both legal numbers of coins of denomination D but K + K' is not.

Once we have done this, we can subtract cashier moneys in the same way, with the only difference being that we will iterate forward through the array because we are subtracting.

This algorithm will fill the array quickly enough. All that is left is to find the smallest index greater than or equal to price and less than or equal to price + $10.00. If no such index exists, returning 0 will handle all the special cases.

A test of the time-out case against this algorithm takes about 5 or 6 seconds.