Giving Change

Just knowing the minimum number of coins is not as useful as getting the actual list of coins. Next, write another version of the `change` function called `giveChange` that takes the same kind of input as `change` but returns a list whose first item is the minimum number of coins and whose second item is a list of the coins in that optimal solution. Here's an example:

```
>>> giveChange(48, [1, 5, 10, 25, 50])

[6, [25, 10, 10, 1, 1, 1]]

>>> giveChange(48, [1, 7, 24, 42])

[2, [24, 24]]

>>> giveChange(35, [1, 3, 16, 30, 50])

[3, [16, 16, 3]]
```

The order in which the coins are presented in the input list doesn't really matter and, similarly, the order in which your solution reports the coins to use is also unimportant: In other words the solution `[3, [16, 16, 3]]` is the same to us as `[3, [3, 16, 16] ]` or `[3, [16, 3, 16]]`. All of these solutions use the same 3 coins after all!

This problem may seem challenging at first, but keep in mind that once you establish that `giveChange` will **always** return a list of the form `[numberOfCoins, listOfCoins]`, you can modify your `change` function relatively modestly to get the `giveChange` function. First, your base cases must observe the convention and return such a list of the form `[numberOfCoins, listOfCoins]`. Then, when you call `giveChange` recursively, remember that it is returning a list of this form. Your function will need to pick apart that list to get at the number of coins and the list of coins in that solution. Finally, after deciding whether the use-it or lose-it solution is better, you can prepare your list of the form `[numberOfCoins, listOfCoins]` and return that list.