Greedy algorithms

As you could notice, some of the algoritms are iterating processes. They choose one option from a certain amount of possible choices.

For example, in the first pseudocode example that was written in my notes an algorithm choose which coin to return first, which to return secind and so on. Some of the options lead to a right answer, but some are not.

Here is a code example, where `for each` *loop iterates through the whole range of optionsand choose the one that it thinks is correct:*

```
// note: ∑ symbol means sum of numbers from i to m.
BruteForceChange(money, c, d):
    smallestNumberOfCoins = 0
    for each combinations of coins (i_1,...,i_d)
    // from (0,...,0) til (money/c[1],...,money/c[d])
        valueOfCoins = ∑ i_k*c_k // sum of every k from 1 til d
        if valueOfCoins = M:
            numberOfCoins = ∑ i_k // sum coins amount
            if numberOfCoins < smallestNumberOfCoins:
                smallestNumberOfCoins = numberOfCoins
                change = (i_1, i_2, ... ,i_d)
    return change
```

Greedy algorithm chooses the locally optimal option. Eventhough an option that algorithm chose may seem the best at the momnet, sometimes it is actually not. The first version of `Change` code (below) is an example of such an algorithm. It may lead to wrong answer because one particular optin seemed the best at the moment and the algorithm chose it.

`Change` *code example:*

```
Change(money, c, d):
    while money > 0:
        coin = ... // coin with the highest nominal (nominal <= money)
        money = money - coin
```

In many cases a greedy method may seem natural and the best practice, but it may lead to wrong answers.