We learned

* Exhaustive Search (Brute Force)
* Branch and Bounds (Pruning)
* Recursion and Backtracking
* Greedy Algorithm (Kruskal’s, Prim’s, Dijkstra’s Algo.)
  + Make the locally optimal choice at each step with the hope of finding a global optimum.
* Dynamic Programming
  + The Solution to the Money Change Problem is an example of this

***The Money Change Problem***

-Convert some amount of money M (in cents) into given denominations, using the smallest possible number of coins.

**First Simple Solution (Greedy Algorithm)**

M = 40 Cents

US coin denominations: (25, 10, 5, 1) sorted in descending order

40/25 = 1 (quarter)

Remaining = 40 - 25x1 = 15

15/10 = 1 (dime)

Remaining = 15 - 10x1 = 5

5/5 = 1 (nickel)

Remaining = 0 cents

Result : {1 quarter, 1 dime, 1 nickel}

3 Coins.

Why is this greedy? Because you always try out the largest denomination first. This Greedy Algorithm Solution to the problem works for the current US coin system. **In General, however, this algorithm will produce the wrong result for other coin systems.** In history, there was the 20 cent piece for like 1 year. But if there was a 20 cent piece how does algo work.

**What If 20 Cent Piece Solution (Greedy Algorithm)**

{25, 20, 10, 5, 1}

M = 40 Cents

40 / 25 = 1 quarter (15 rem)

15 / 20 = 0 20cent (15 rem)

15 / 10 = 1 dime (5 rem)

5 / 5 = 1 nickel (rem 0)

Answer is (1 quarter, 1 dime, 1 nickel) = *3 coins*.

But this is wrong, as the optimal solution is *2 coins* (2 20cent pieces make 40 cents).

***Greedy Algorithms are Easy to Design, but may not produce the correct/best solution***.

**First Dynamic Programming Solution**

{c1 = 7, c2 = 3, c3 = 1}

M = 77 Cents

BestNumCoin(77) will be one of the following options

* BestNumCoin(77-1), plus a 1-cent coin
* BestNumCoin(77-3), plus a 3-cent coin
* BestNumCoin(77-7), plus a 7-cent coin

Of these three options, the minimum number will be the answer to the Money Change Problem.